Data Mining on Human Activity Recognition

Human Activity Recognition Analysis using machine learning methods:

Activities are recorded both in our mobile phones and smart watches via accelerometer monitoring data. These data can then be used in numerous ways to predict all kinds of health problems, bad habits and etc. Finding the exact kind of activity using the accelerometer data is a difficult task, let alone using these designated activities to identify a bad habit or possible risks. I'll be using machine learning techniques to verify these activities based on their accelerometer data and then it can be used to map new examples (predict user activities) and be analyzed further.

Background:

Activity recognition is a significant technology in ubiquitous computing because it can be applied to many real-life, human-centric problems such as eldercare and healthcare. Ubiquitous computing (pervasive) is the growing trend towards embedding processors in everyday objects so they can monitor and pass on information. Successful research has so far focused on recognizing simple human activities.

The Heterogeneity Dataset for Human Activity Recognition from Smartphone and Smartwatches is a dataset planned to benchmark human activity recognition algorithms (classification, automatic data segmentation, sensor fusion, feature extraction, etc) containing sensor heterogeneities.

The files in this archive contain all the samples from the activity recognition experiment. The dataset contains the readings of two motion sensors commonly found in smartphones' recorded while users executed activities scripted in no specific order carrying smartwatches and smartphones.

Data sources:

This is a documentation for the Heterogeneity Dataset for Human Activity Recognition (HHAR) from Smartphones and Smartwatches from the public repository:

https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition+Data+Set

or the personal Website: http://cs.au.dk/~allans/heterogenity/

The data is split into 4 files in total divided by device (phone or watch) and sensor (gyroscope and accelerometer). The files for phones are: Phones_accelerometer.csv, Phones_gyroscope.csv for the accelerometer and gyroscope respectively, and for the Watch_accelerometer.csv, Watch_gyroscope.csv for the accelerometer and gyroscope as well.

Activities: ‘Biking’, ‘Sitting’, ‘Standing’, ‘Walking’, ‘Stair Up’ and ‘Stair down’.

Sensors: Two embedded sensors, i.e., Accelerometer and Gyroscope sampled at the highest frequency possible by the device

Devices: 4 smartwatches (2 LG watches, 2 Samsung Galaxy Gears) 8 smartphones (2 Samsung Galaxy S3 mini, 2 Samsung Galaxy S3, 2 LG Nexus 4, 2 Samsung Galaxy S+)

Recordings: 9 users currently named: a,b,c,d,e,f,g,h,i consistently across all files.

The data set is structured in the following way:

————- Accelerometer Samples ————

All the csv files have the same structure of following columns:

‘Index’, ‘Arrival_Time’, ‘Creation_Time’, ‘x’, ‘y’, ‘z’, ‘User’, ‘Model’, ‘Device’, ‘gt’

And the columns have the following values:

Index: is the row number.

Arrival_Time: The time the measurement arrived to the sensing application

Creation_Time: The timestamp the OS attaches to the sample

X,y,z The values provided by the sensor for the three axes, X,y,z

User: The user this sample originates from, the users are named a to i.

Model: The phone/watch model this sample originates from

Device: The specific device this sample is from. They are prefixed with the model name and then the number, e.g., nexus4_1 or nexus4_2.

Gt: The activity the user was performing: bike sit, stand, walk, stairsup, stairsdown and null

————- Devices and models ————————

The names and models of the devices used in the HAR data set are:

LG-Nexus 4 ‘nexus4_1’

‘nexus4_2’ Saumsung Galaxy S3

‘s3_1’ ‘s3_2’

Samsung Galaxy S3 min: Samsung Galaxy S+:

‘s3mini_1’ ‘s3mini_2’

‘samsungold_1’ ‘samsungold_2’

Algorithms:

In the project, I am using K-means, KNN, LDA, SVM, Naive Bayes, Random Forest, Boosting and Bagging to make the data analysis and evaluate the results with the provided labels. Then establish a predictive model to make further analysis on the activities based on raw data. I also could evaluate the algorithms on their efficiency (time and correctness).

Description:

The MHEALTH (Mobile HEALTH) dataset comprises body motion and vital signs recordings for ten volunteers of diverse profile while performing several physical activities. Sensors placed on the subject’s chest, right wrist and left ankle are used to measure the motion experienced by diverse body parts, namely, acceleration, rate of turn and magnetic field orientation. The sensor positioned on the chest also provides 2-lead ECG measurements, which can be potentially used for basic heart monitoring, checking for various arrhythmias or looking at the effects of exercise on the ECG.

Here we import these datasets and combine them together. Then we test the accuracy of activity set labels (the 24th attribute comprising 12 different activities) generated using popular machine learning algorithms mentioned earlier and compare their usability and effectiveness in this case. The detailed process is explained below.

First we collect the data for each of the ten volunteers and perform normalizing, scaling and shuffling. Then we specify the training and test data. After that we start applying the algorithms and assess the results.

```r
set.seed(1)

dt1 <- read.table("./MHEALTHDATASET/mHealth_subject1.log")
# summary(dt1)
dim(dt1)
```

1) Data Prepration

```
## [1] 161280     24
```

```r
dt2 <- read.table("./MHEALTHDATASET/mHealth_subject2.log")
# summary(dt2)
dim(dt2)
```

```
## [1] 130561     24
```

```r
dt3 <- read.table("./MHEALTHDATASET/mHealth_subject3.log")
# summary(dt3)
dim(dt3)
```

```
## [1] 122112     24
```

```r
dt4 <- read.table("./MHEALTHDATASET/mHealth_subject4.log")
# summary(dt4)
dim(dt4)
```

```
## [1] 116736     24
```

```r
dt5 <- read.table("./MHEALTHDATASET/mHealth_subject5.log")
# summary(dt5)
dim(dt5)
```

```
## [1] 119808     24
```

```r
dt6 <- read.table("./MHEALTHDATASET/mHealth_subject6.log")
# summary(dt6)
dim(dt6)
```

```
## [1] 98304    24
```

```r
dt7 <- read.table("./MHEALTHDATASET/mHealth_subject7.log")
# summary(dt7)
dim(dt7)
```

```
## [1] 104448     24
```

```r
dt8 <- read.table("./MHEALTHDATASET/mHealth_subject8.log")
# summary(dt8)
dim(dt8)
```

```
## [1] 129024     24
```

```r
dt9 <- read.table("./MHEALTHDATASET/mHealth_subject9.log")
# summary(dt9)
dim(dt9)
```

```
## [1] 135168     24
```

```
dt10 <- read.table("./MHEALTHDATASET/mHealth_subject10.log")
# summary(dt10)
dim(dt10)
```

```
## [1] 98304    24
```

```
all <- rbind(dt1,dt2,dt3,dt4,dt5,dt6,dt7,dt8,dt9,dt10)

# Name the activity set field name to Y
names(all)[24] <- "Y"
names(all)
```

```
##  [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
## [12] "V12" "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22"
## [23] "V23" "Y"
```

```
all <- all[all$Y != 0, ]     # Remove the null values

all$Y <- as.factor(all$Y)

str(all)
```

```
## 'data.frame':    343195 obs. of  24 variables:
##  $ V1 : num  -9.78 -9.77 -9.86 -9.74 -9.78 ...
##  $ V2 : num  0.557 0.279 0.116 0.177 0.216 ...
##  $ V3 : num  1.198 0.73 0.8 0.89 0.904 ...
##  $ V4 : num  0.00837 -0.02512 0.02512 0.18001 0.0921 ...
##  $ V5 : num  -0.0335 -0.0251 0.0167 0.1298 0.046 ...
##  $ V6 : num  2.65 2.42 2.39 2.38 2.32 ...
##  $ V7 : num  -9.45 -9.53 -9.6 -9.6 -9.54 ...
##  $ V8 : num  0.377 0.402 0.481 0.429 0.4 ...
##  $ V9 : num  -0.21 -0.21 -0.2 -0.2 -0.2 ...
##  $ V10: num  -0.889 -0.889 -0.869 -0.869 -0.869 ...
##  $ V11: num  -0.509 -0.509 -0.507 -0.507 -0.507 ...
##  $ V12: num  0.564 0.568 0.211 0.216 0.568 ...
##  $ V13: num  0.545 0.912 0.548 1.282 0.912 ...
##  $ V14: num  -0.738 -0.886 -1.02 -1.171 -0.886 ...
##  $ V15: num  -2.84 -2.99 -2.88 -2.92 -2.9 ...
##  $ V16: num  -9.06 -9.2 -9.19 -9.17 -9.2 ...
##  $ V17: num  1.82 1.52 1.55 1.54 1.61 ...
##  $ V18: num  -0.0588 -0.0588 -0.0588 -0.0784 -0.0784 ...
##  $ V19: num  -0.934 -0.934 -0.934 -0.934 -0.934 ...
##  $ V20: num  -0.345 -0.345 -0.345 -0.341 -0.341 ...
##  $ V21: num  0.35537 0.71991 0.35537 0.35718 -0.00189 ...
##  $ V22: num  -0.37 0.178 -0.37 -0.189 -0.189 ...
##  $ V23: num  -0.35 0.374 -0.35 -0.352 -0.72 ...
##  $ Y  : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# 'all' is all the data we have. 23 attributes (column 1-23) and 1 label(column 24) with 12 classes.
```

```r
library('som')
```

2) Normalizing and Shuffling the data:

```
## Warning: package 'som' was built under R version 3.2.5
```

```r
all_norm <- normalize(all[,-24], byrow=FALSE)
all_norm <- as.data.frame(all_norm)
all_norm$Y <- all$Y
head(all_norm)
```

```
##           V1         V2        V3           V4          V5        V6
## 1 -0.4022359 0.24927791 0.4632058  0.005629246 -0.02966275 0.2003039
## 2 -0.4012714 0.14993447 0.3618994 -0.034284536 -0.01990204 0.1448720
## 3 -0.4166345 0.09163941 0.3769759  0.025586256  0.02890501 0.1379430
## 4 -0.3955891 0.11339780 0.3964265  0.210185084  0.16067811 0.1354039
## 5 -0.4028147 0.12763310 0.3994865  0.105412390  0.06306983 0.1230883
## 6 -0.3868552 0.12555764 0.3263249 -0.004349140 -0.01502052 0.1956530
##            V7        V8         V9        V10        V11          V12
## 1 -0.07503534 0.1670606 -0.6772897 -0.7806575 -0.6973897  0.003087220
## 2 -0.09020728 0.1709258 -0.6772897 -0.7806575 -0.6973897  0.003145287
## 3 -0.10337937 0.1832555 -0.6570680 -0.7331787 -0.6938641 -0.001523649
## 4 -0.10349474 0.1751689 -0.6570680 -0.7331787 -0.6938641 -0.001464143
## 5 -0.09213021 0.1707075 -0.6570680 -0.7331787 -0.6938641  0.003145287
## 6 -0.09216867 0.1663390 -0.6570680 -0.7331787 -0.6938641  0.005451507
##          V13         V14        V15        V16        V17       V18
## 1 0.05016872 -0.01895734 0.10257009 -0.4986576 -0.1348656 0.2597697
## 2 0.05790493 -0.02679182 0.07717799 -0.5203955 -0.2063836 0.2597697
## 3 0.05024315 -0.03385694 0.09566195 -0.5188297 -0.1987722 0.2597697
## 4 0.06571535 -0.04188403 0.08888959 -0.5158047 -0.2010221 0.2240897
## 5 0.05790493 -0.02679182 0.09367607 -0.5202587 -0.1839325 0.2240897
## 6 0.06173687 -0.01943886 0.10586292 -0.5077176 -0.1256027 0.2240897
##         V19       V20        V21         V22          V23 Y
## 1 -0.958664 -1.406984 0.02595255 -0.05917632 -0.004607498 1
## 2 -0.958664 -1.406984 0.03660655 -0.04107973  0.004148898 1
## 3 -0.958664 -1.406984 0.02595255 -0.05917632 -0.004607498 1
## 4 -0.958664 -1.398626 0.02600545 -0.05318496 -0.004629031 1
## 5 -0.958664 -1.398626 0.01551141 -0.05318793 -0.009083139 1
## 6 -0.958664 -1.398626 0.02605864 -0.04719351 -0.004650564 1
```

```r
# 'all_norm' is the 'all' data frame but normalized.

dt <- all_norm[sample(nrow(all_norm), nrow(all_norm)), ]

# dt is the shuffled format of 'all_norm'
```

```r
dt <- cbind(as.data.frame(lapply(dt[,-24], scale)), Y=dt[,24])
head(dt)
```

3) Scaling the data:

```
##           V1          V2         V3          V4          V5          V6
## 1  0.5810420  0.59942693 -1.6155537 -3.44194991  1.15146255  0.08775520
## 2 -0.4184760 -0.08192405 -0.0411355 -0.09415414 -0.10287391 -0.45736693
## 3  0.2362826  0.89167026 -0.7252795 -0.44339287 -0.34202416  0.11022698
## 4 -0.8082726 -0.07768383 -0.7805150 -0.00434914  0.01426313 -0.03559041
## 5 -0.2212276 -0.14388778 -0.4166156  1.18307950  0.31198545 -0.18570284
## 6 -0.6027288  0.81476036 -1.0376519 -0.19393919 -0.12728152  0.24951878
##           V7          V8         V9         V10         V11         V12
## 1  0.5253620 -1.00640995 -0.8147450  0.88526211 -1.72929527  0.20851179
## 2 -0.1165130  0.35305324 -1.1745518 -0.04696828  1.30992075 -0.01845476
## 3 -0.5233287  0.24843114 -0.5681187 -0.41812599 -0.98010875 -0.73393329
## 4 -0.5950541 -0.16029359 -0.3700332 -3.47799753 -1.56675477 -0.06303067
## 5  0.2948216 -0.86584738  0.5517234 -0.43537844 -0.06127641  0.08094520
## 6 -0.5550571 -0.01003085  0.4870489 -1.60070093  0.51830028  0.01498634
##           V13         V14         V15        V16        V17         V18
## 1  0.03138282  0.29321674  0.83065816 -0.5264152  0.9998718 -0.06850719
## 2  0.05438460  0.02085753  0.08941576 -1.0316467  0.4440039 -0.94983372
## 3 -0.27451485 -0.27186003  0.80149797 -0.5727945  1.5970995 -0.17911203
## 4 -0.04934838  0.30194760 -0.07176300 -0.5573955 -0.2865900  0.46672338
## 5  0.16883943 -0.01296480  1.20858637 -0.4323190  1.4560741  1.52645524
## 6 -0.20533958  0.60496645 -0.06801190 -1.1842681 -0.1339800 -1.04973838
##          V19        V20        V21         V22         V23 Y
## 1 -1.2180356 -0.3579360 -0.3755633 -0.37004271  0.07618560 8
## 2 -0.4588095  0.1477869 -1.0436083  1.21936736 -1.46801669 7
## 3 -1.1466539 -0.2074680  0.5233369 -0.18297482  0.05673796 9
## 4 -2.4996104  0.1770311 -0.0935687  0.07182568 -0.08468806 5
## 5 -0.3423110  1.2888536 -0.1762925 -0.38217072 -0.06623966 9
## 6 -0.1919703  0.8415803 -0.4436596  0.23811449 -0.28471830 5
```

```r
# remove the previous variables:

rm(dt1,dt2,dt3,dt4,dt5,dt6,dt7,dt8,dt9,dt10,all,all_norm)
```

```r
ntr <- 40000 #training
nte <- 10000 #test

train1 <- dt[1:ntr,]
dim(train1)
```

4) Specifying Training and Test data

```
## [1] 40000    24
```

```
test1 <- dt[(ntr+1):(ntr+nte),]
dim(test1)
```

```
## [1] 10000    24
```

```
k <- 12
trails <- 40
kmean_cluster <- kmeans(rbind(train1[, -24], test1[, -24]), k, nstart=trails)
str(kmean_cluster)
```

5) Applying Kmeans

```
## List of 9
##  $ cluster     : Named int [1:50000] 9 8 9 9 5 3 3 7 3 8 ...
##   ..- attr(*, "names")= chr [1:50000] "1" "2" "3" "4" ...
##  $ centers     : num [1:12, 1:23] -1.4567 -0.0101 -0.302 -0.2102 0.0754 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:12] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:23] "V1" "V2" "V3" "V4" ...
##  $ totss       : num 1146324
##  $ withinss    : num [1:12] 74807 56549 64723 40939 20508 ...
##  $ tot.withinss: num 678698
##  $ betweenss   : num 467626
##  $ size        : int [1:12] 1664 1090 7005 6714 4143 1720 3521 7257 7827 4401 ...
##  $ iter        : int 8
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
head(kmean_cluster$cluster)
```

```
## 1 2 3 4 5 6
## 9 8 9 9 5 3
```

```
table(pred=kmean_cluster$cluster, truth=c(train1$Y, test1$Y))
```

```
##     truth
## pred    1    2    3    4    5    6    7    8    9   10   11   12
##   1     0    0    0    0    2    0    1    0    0  774  778  109
##   2     0    0    0   30    9    0    0    0    0  353  630   68
##   3   876  874    0 2644 1171  329  334  163    0  180  151  283
##   4   469 1809    0  160 1551  275  431 1436  453    7    2  121
##   5     0    0    0    0    8  397   58    0 3622    6   20   32
##   6     0    0    0    5    3    0    9    0    0  800  764  139
##   7     0  922    0    0   44    0 2321    0    0   74   41  119
##   8  2189  911    0   20 1007 1295  881  679    0  133   13  129
##   9   874    0    0 1202  731 1803  215 1988  401  297    4  312
##   10    0    0 4383    0    8    0    0    0    0    1    3    6
##   11    0    0    0    1    0    0    0    0 1222 1271  121
##   12    0    0    0  492   10    0    0    0  715  784   42
```

```r
agreement <- kmean_cluster$cluster == c(train1$Y, test1$Y)
table(agreement) / 5
```

```
## agreement
##  FALSE   TRUE
## 9023.4  976.6
```

10% accurate

```r
library(class)

k <- 1

knn.m1 <- knn(train = train1[, -24], test = test1[, -24], train1$Y, k)
head(knn.m1)
```

6) Applying KNN

```
## [1] 7 8 2 4 3 6
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

```r
table(pred=knn.m1, truth=test1$Y)
```

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##    1 878   0   0   0   2   0   0   1   0   0   0   0
##    2   0 911   0   0   0   0   0   0   0   0   0   0
##    3   0   0 871   0   0   0   0   0   0   0   0   0
##    4   0   0   0 964  18   0   0   0   0   3   0   8
##    5   0   0   0   1 860   2   1   0   0   0   0   9
##    6   0   0   0   0   4 819   1   3   0   0   0   0
##    7   1   0   0   0   1   6 829   1   0   0   0   0
##    8   0   0   0   0  18   3   1 855   0   0   0   0
##    9   0   0   0   0   1   0   0   0 886   0   0   0
##   10   0   0   0   0   2   0   0   0   0 836  56  28
##   11   0   0   0   0   0   0   0   0   0  11 853   6
##   12   0   0   0   1   1   0   0   0   0   1   0 247
```

```r
agreement <- knn.m1 == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##   191  9809
```

98% accurate

```r
library(MASS)
```

7) Applying LDA

```
## Warning: package 'MASS' was built under R version 3.2.4
```

```r
lda.m1 <- lda(Y ~., data=train1)

lda.m1.p <- predict(lda.m1, test1[, -24])
head(lda.m1.p$class)
```

```
## [1] 7 8 2 4 3 6
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

```r
table(pred=lda.m1.p$class, truth=test1$Y)
```

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##   1  612   0   0 108 113 157 154 157   0   1   1  12
##   2    0 666   0   3  32   2  99   0   0  60  87  35
##   3    0   0 871   0   3   0   0   0   0   0   1   1
##   4  185   0   0 627 158  45  58  38   0  14  12  13
##   5   82  43   0 180 411  44  23 130   3  23  33   6
##   6    0   0   0   0  32 543   0  38   0   0  22  15
##   7    0 177   0   0  15   0 489   0   0   5  13  19
##   8    0  25   0  43 114  39   4 494 145  26  26  14
##   9    0   0   0   0   6   0   5   0 738   3  13   7
##   10   0   0   0   1  15   0   0   1   0 497  90  70
##   11   0   0   0   4   4   0   0   0   0 133 586  52
##   12   0   0   0   0   4   0   0   2   0  89  25  54
```

```r
agreement <- lda.m1.p$class == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##  3412  6588
```

66% accurate

```
library('e1071')
```

8) Applying SVM

```
## Warning: package 'e1071' was built under R version 3.2.5
```

```
svm.fit <- svm(as.factor(Y)~., data=train1, kernal="radial", cost=10, scale=TRUE)
head(svm.fit$index) #lists the support vectors
```

```
## [1]  19  25  36  68  92 103
```

```
summary(svm.fit)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = train1, kernal = "radial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.04347826
##
## Number of Support Vectors:  8263
##
##  ( 872 627 134 1033 649 108 1293 1657 22 399 689 780 )
##
##
## Number of Classes:  12
##
## Levels:
##  1 2 3 4 5 6 7 8 9 10 11 12
```

```
ypred <- predict(svm.fit, test1[, -24])
head(ypred)
```

```
## 40001 40002 40003 40004 40005 40006
##     7     8     2     4     3     6
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

```
table(pred=ypred, truth=test1$Y)
```

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##   1  879   0   0   0   0   2   0   2   0   0   0   0
##   2    0 911   0   0   0   0   0   0   0   0   0   0
##    3   0   0 871   0   0   0   0   0   0   0   0   0
##    4   0   0   0 963  16   0   0   0   0   0   0   1
##    5   0   0   0   1 876   2   2   2   1   0   0   0
##    6   0   0   0   0   1 821   4   5   0   0   0   0
##    7   0   0   0   0   1   1 823   5   0   0   0   0
##    8   0   0   0   1   9   4   3 845   2   0   0   0
##    9   0   0   0   0   1   0   0   0 883   0   0   0
##   10   0   0   0   0   0   0   0   0   0 833  23   4
##   11   0   0   0   0   0   0   0   1   0  17 886  10
##   12   0   0   0   1   3   0   0   0   0   1   0 283
```

```
agreement <- ypred == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##   126  9874
```

98% accurate

```


model1 <- naiveBayes(Y~., data=train1)
ypred_naivbs1 <- predict(model1, test1[,-24])

table(pred=ypred_naivbs1, truth=test1$Y)
```

9) Applying Naive Bayes Method

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##    1 855   0   0   0   0  11  11  10   0   0   0   0
##    2   3 896   0   0   0   0   1   0   0   0   0   0
##    3   0   0 869   0   0   0   0   0   0   0   0   0
##    4   0   0   0 821 130   1   0  10   1   1   0   2
##    5   1   0   0  87 594   4   1  61   1   0   0   0
##    6  10   0   0  13  23 744   8 171   0   0   0   0
##    7  10   2   0   0   1  58 806  25   0   0   0   0
##    8   0   9   0  30 115  12   0 571  52   0   0   1
##    9   0   0   0   0   2   0   0   1 832   0   0   0
##   10   0   0   0   3  25   0   0   9   0 618 158 117
##   11   0   2   2   8   6   0   1   0   0 176 706  57
##   12   0   2   0   4  11   0   4   2   0  56  45 121
```

```
agreement <- ypred_naivbs1 == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##  1567  8433
```

84% accurate

```
library(randomForest)
```

10) Applying Random Forest

```
## Warning: package 'randomForest' was built under R version 3.2.5
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
fit <- randomForest(Y~., data=train1, importance=TRUE, ntree=200)
ypred_rf1 <- predict(fit, test1[, -24])
head(ypred_rf1)
```

```
## 40001 40002 40003 40004 40005 40006
##     7     8     2     4     3     6
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

```
table(pred=ypred_rf1, truth=test1$Y)
```

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##   1  879   0   0   0   0   3   0   0   0   0   0   0
##   2    0 911   0   0   0   0   0   0   0   0   0   0
##   3    0   0 871   0   0   0   0   0   0   0   0   0
##   4    0   0   0 966   9   0   0   0   0   0   0   0
##   5    0   0   0   0 891   1   0   4   0   0   0   0
##   6    0   0   0   0   1 823   0   0   0   0   0   0
##   7    0   0   0   0   0   1 832   0   0   0   0   0
##   8    0   0   0   0   2   2   0 856   1   0   0   2
##   9    0   0   0   0   0   0   0   0 885   0   0   0
##   10   0   0   0   0   3   0   0   0   0 821  13  11
##   11   0   0   0   0   1   0   0   0   0  30 896  13
##   12   0   0   0   0   0   0   0   0   0   0   0 272
```

```
agreement <- ypred_rf1 == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##    97  9903
```

99% accurate

```r
library(ipred)
```

11) Applying Bagging method

```
## Warning: package 'ipred' was built under R version 3.2.5
```

```r
fit1 <- bagging(Y~., data=train1)
ypred_bg1 <- predict(fit1, test1[, -24], type="class")
head(ypred_bg1)
```

```
## [1] 7 8 2 4 3 6
## Levels: 1 10 11 12 2 3 4 5 6 7 8 9
```

```r
ypred_bg1 <- factor(ypred_bg1, levels = c(1,2,3,4,5,6,7,8,9,10,11,12))

table(pred=ypred_bg1, truth=test1$Y)
```

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##   1  878   0   0   0   2   2   1   2   0   0   0   0
##   2    0 911   0   0   2   1   1   0   0   0   0   0
##   3    0   0 871   0   0   0   0   0   0   0   0   0
##   4    0   0   0 944  21   0   0   6   1   3   0   1
##   5    1   0   0  15 838   0   0  10   3   3   0   6
##   6    0   0   0   2   7 815   1  12   0   0   0   2
##   7    0   0   0   1   4   8 824   3   0   0   1   0
##   8    0   0   0   2  20   4   4 826   3   1   0   4
##   9    0   0   0   0   3   0   0   0 879   1   1   1
##   10   0   0   0   2   5   0   1   0   0 803  35  30
##   11   0   0   0   0   1   0   0   0   0  37 868  17
##   12   0   0   0   0   4   0   0   1   0   3   4 237
```

```r
agreement <- ypred_bg1 == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##   306  9694
```

96% accurate

```r
library(adabag)
```

12) Applying Boosting method

```
## Warning: package 'adabag' was built under R version 3.2.5

## Loading required package: rpart

## Loading required package: mlbench

## Warning: package 'mlbench' was built under R version 3.2.5

## Loading required package: caret

## Warning: package 'caret' was built under R version 3.2.5

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.5

## 
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
## 
##     margin

## 
## Attaching package: 'adabag'

## The following object is masked from 'package:ipred':
## 
##     bagging
```

```r
model1 <- boosting(Y~., data=train1, boos = FALSE, mfinal = 10)
ypred_boost1 <- predict(model1, test1[, -24])

table(pred=as.integer(ypred_boost1$class), truth=test1$Y)
```

```
##     truth
## pred   1   2   3   4   5   6   7   8   9  10  11  12
##    1 869   2   0  27  35 102 112  92   1   0   1   1
##    2   0 908   0   4  18   4  20   2   2  17   6  11
##    3   0   0 871   0   0   0   1   0   0   5  28   5
##    4   1   0   0 778 122  15   1  92   1  47  21  13
##    5   9   0   0  64 543  50   2 149   1   7   3   4
##   6     0   0   0  36  46 622   2  66   2   4   2  20
##   7     0   0   0   8  12   8 666   2  13   2   6   9
##   8     0   0   0   8  81  27   0 408   7   4   3   9
##   9     0   0   0   3  13   2   7  47 857   4   6   6
##   10    0   0   0   1   8   0  16   2   2 565 123 132
##   11    0   1   0  37  28   0   5   0   0 195 710  78
##   12    0   0   0   0   1   0   0   0   0   1   0  10
```

```
agreement <- ypred_boost1$class == test1$Y
table(agreement)
```

```
## agreement
## FALSE  TRUE
##  2193  7807
```

78% accurate

Interpretation:

In supervised learning, one set of observations, called inputs, is assumed to be the cause of another set of observations, called outputs, while in unsupervised learning all observations are assumed to be caused by a set of latent variables. Therefore, as we saw in the results for Kmeans, unsupervied learning is not efficient for this kind of prediction.

The standard implementation of the LDA model assumes a Gaussian distribution of the input variables. Also we have to Consider removing outliers from our data. These can reorient the basic statistics used to separate classes in LDA such as the mean and the standard deviation. LDA also assumes that each input variable has the same variance. It is almost always a good idea to standardize our data before using LDA so that it has a mean of 0 and a standard deviation of 1. Some of these or maybe all of them could be the reasons for the less efficient performance of LDA in our case.

Naive Base has the strong assumption of independency between the features. This in fact is not exactly true and even not so close. The reason is that some sensors' values impose some change in the other sensors and this will be against independency of attributes. Therefore we will expect a lower accuracy in the performance of the classifier. The results indicate that we have only 84% accuracy in the test set as opposed to 98% accuracy of KNN and SVM.

Baysian classifier could be very useful in problems with almost independent features or problems with many features that with other sort of conventional classifiers, The problem would be very time consuming and computationally expensive like spam recgnition for emails.

Best algorithms in order: Random Forest, KNN, SVM, Bagging, Boosting

KNN tends to perform very well with a lot of data points. On the minus side KNN needs to be carefully tuned, the choice of K and the metric (distance) to be used are critical. KNN is also sensitive to outliers and removing them before using KNN tends to improve results. The data I used was pretty clean, so it made KNN's outcome as good as SVM's.

When you have a limited set of points in many dimensions, SVM tends to be very efficient because it should be able to find the linear separation that exists. SVM is good with outliers as it will only use the most relevant points to find a linear separation (support vectors). SVM also needs to be tuned, the cost and the use of a kernel and its parameters are critical to the algorithm. In the data we used, there were both many data points and many dimensions, therefore easing the prediction for both KNN and SVM.

Bagging and Boosting do a very good job on classifying the activities based on the sensor data. In fact after Random Forest, KNN and SVM, Bagging and Boosting were the most successful classifiers in my dataset.

Random Forest can very well handle high dimensional spaces as well as large number of training examples. It also deals really well with uneven data sets that have missing variables. Another advantage of these algorithms is that you don't have to worry about tuning a bunch of parameters like you do with SVMs, so they seem to be quite popular these days. It's constructed using bagging or boosting, so all the more reason to beat all the other algorithms here.

References:

1) UCI Machine Learning Repository:

https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition

2) A. Stisen, H. Blunck, S. Bhattacharya, T. Prentow, M. Kjærgaard, A. Dey, T. Sonne, M. Møller Jensen, "Smart Devices are Different: Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition", Proc. 13th ACM Conference on Embedded Networked Sensor Systems (SenSys), Seoul, Korea, 2015