PART 1
PART 2
PART 3
References

Code

Statistical Learning - Homework 2

Davide Aureli, Andrea Marcocchia and Dario Stagnitto

17/06/2018

PART 1

Introduction

The goal of classification is to build a model of the distribution of class labels in terms of predictor features. We observe i.i.d. data pairs from the joint $D_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ where each tuple is composed by:

- $Y_i \in \{0, 1, \ldots, K-1\}$ which represents the classification variable (with K classes)
- $X_i = (x_1, \ldots, x_p)$ which represents the values of the predictors

In other words a classifier predicts Y given a feature vector X.

From now on we're gonna focus on a **binary classification** problem so $Y_i \in \{0, 1\}$.

The resulting classifier is then used to assign class labels to the testing instances, where the values of the predictor features are known, but the values of the class value not.

There are many classification methods but we're going to talk about the **Naive Bayes classifier**.

Naive Bayes

The Bayes classifiers are a family of algorithms, and the Naive Bayes is a special case in which is assumed that the value of a particular feature is independent from the value of any other feature, given the class variable. Firstly, according to the Bayes' rule, we

PART 1
PART 2
PART 3
References

define the Bayes classifier as follows.

$$P(Y \mid X_i) = \frac{P(X_i \mid Y) \cdot P(Y)}{P(X_i)}$$

with $i \in \{1, \ldots, n\}$.

Given that, X is classified in the class 0 if and only if:

$$f_b(X_i) = \frac{P(Y_i = 0 \mid X_i)}{P(Y_i = 1 \mid X_i)} \geq 1$$

where $f_b(X_i)$ is called a **Bayesian classifier**.

We're not talking about the Naive classifier yet. We need to make a fundamental assumption which marks the Naive classifiers: **the predictors are considered independent to each other conditionally to the labels**.

Let's go to math...

According to what we said above, we can explicit the *Likelihood* using this latter assumption:

$$P(X_i \mid Y = y) = P(x_1, \ldots, x_p \mid y) = \prod_{j=1}^{p} P(x_j \mid y)$$

The resulting classifier is:

$$f_{nb}(X_i) = \frac{P(Y = 0) \cdot \prod_{j=1}^{p} P(x_j \mid 0)}{P(Y = 1) \cdot \prod_{j=1}^{p} P(x_j \mid 1)}$$

where $f_{nb}(X_i)$ is called **Naive Bayes classifier**.

Now the question is:

Is it meaningfull, in real case applications, to consider independence among all the covariates?

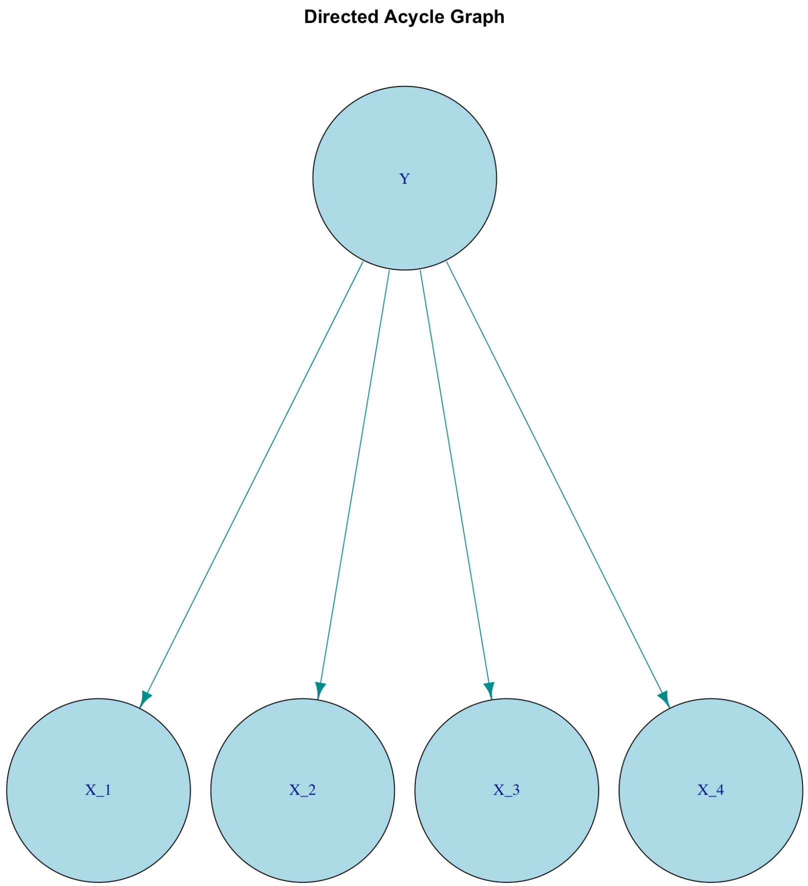
PART 1
PART 2
PART 3
References

Let’s start to represent the basic situation by means a **DAG** (Directed Acycle Graph):

Hide

```
library(igraph)
graph <- make_empty_graph(directed=T)
graph <- graph + vertex(color= 'lightblue',name="Y")+vertex(color= 'lightblue',name=expression(X_1))+vertex(color= 'lightblue',name=expression(X_2))+vertex(color= 'lightblue',name=expression(X_3))+vertex(color= 'lightblue',name=expression(X_4))
graph <- graph + edge(1, 2)+ edge(1, 3)+ edge(1, 4)+ edge(1, 5)
l <-layout.reingold.tilford(graph)
plot(graph,layout = l, vertex.size=60,edge.arrow.size=0.9,edge.color='darkcyan', main="Directed Acycle Graph")
```

PART 1
PART 2
PART 3
References



As shown above we have the node Y which represents the label and (x_1, x_2, x_3, x_4) the features. It is possible deriving the joint distribution as follows:

$$P(Y_i, X_i) = P(Y_i) \cdot \prod_{j=1}^{p} P(x_j \mid Y_i)$$

In this case we have that all attributes are independent given the value of the class variable.

PART 1
PART 2
PART 3
References

This is called *conditional independence*.

But how can be possible that Naive Bayes works well even though the assumption is almost never hold in the real world?

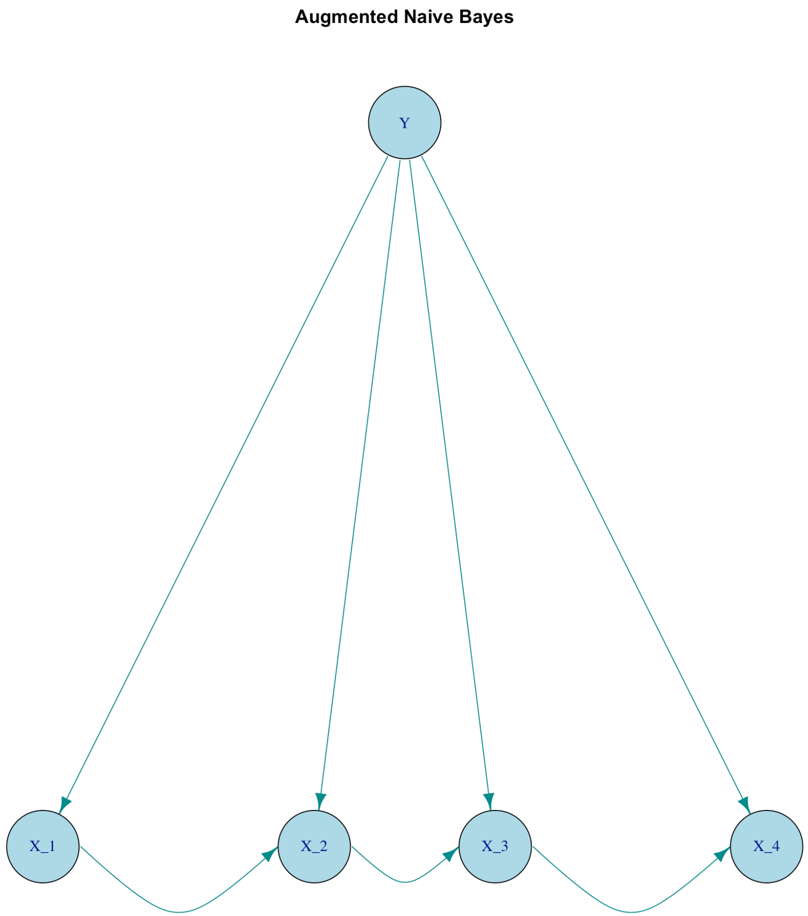
It has been observed that Naive Bayes may still have high accuracy on a dataset in which strong dependences exist among attributes. Proceeding on this way, the structure of our graph changes.

Let’s plot the **ANB** (Augmented Naive Bayes)

Hide

```
library(igraph)
graph <- make_empty_graph(directed=T)
graph <- graph+ vertex(color= 'lightblue',name="Y")+vertex(color= 'lig
htblue',name=expression(X_1))+vertex(color= 'lightblue',name=expressio
n(X_2))+vertex(color= 'lightblue',name=expression(X_3))+vertex(color=
'lightblue',name=expression(X_4))
graph <- graph + edge(1, 2)+ edge(1, 3)+ edge(1, 4)+ edge(1, 5)+ edge(
2, 3)+ edge(3, 4)+ edge(4, 5)
l <-layout.reingold.tilford(graph)
l[,1] <- c(0,-20,-5,5,20)
plot(graph,layout = l,edge.arrow.size=0.9,edge.color='darkcyan',edge.c
urved=c(0,0,0,0,-1,-1,-1,-1),vertex.size=20 ,main="Augmented Naive Bay
es")
```

PART 1
PART 2
PART 3
References



Now, the joint distribution is defined as:

$$P(Y_i, X_i) = P(Y_i) \cdot \prod_{j=1}^{p} P(x_j \mid pa(x_j), Y_i)$$

where $pa(x_j)$ **represents the value of the parents releated to** x_j.

Under which conditions those two graphs are releated to each other?

PART 1
PART 2
PART 3
References

The key point here is that we need to know how the dependencies affect the classification, and under what conditions do not.

First of all… ***Local Dependence Distribution (LDD)***!

For each node, the influence of its parents is quantified by the correspondent conditional probabilities. The dependence between a node and its parents is the local dependence of this node.

The ratio of the conditional probability of the node given its parents over the conditional probability without the parents, reflects how strong the parents affect the feature in each class.

Do math again….

$$LDD^0(x_j \mid pa(x_j)) = \frac{P(x_j \mid pa(x_j), 0)}{P(x_j \mid 0)}$$

LDD^0 reflects the strength of the local dependence of node x_j in class 0, that is measures the influence of x_j's local dependence for the classification into the class 0.

$$LDD^1(x_j \mid pa(x_j)) = \frac{P(x_j \mid pa(x_j), 1)}{P(x_j \mid 1)}$$

The same interpretation is valid fo LDD^1.

Results:

- If the node has no parents $LDD^0 = LDD^1 = 1$
- If $LDD^0 > 1$ then, x_i local dependence within the class, supports the class 0. Otherwise the opposite one. The same situation holds for LDD^1.

What does all this stuff mean?

Well, **when the local dependence derivatives in both classes support the different classifications, they cancel partially each other out**, and the final classification supported by the local dependence is the class with the greater local dependence derivative.

PART 1

PART 2

PART 3

References

Another case is that **the local dependence derivatives in the two classes support the same classification**. So the local dependencies in the two classes work together to support the splitting.

Now we analyze a new measure that helps us to understand and quantify the behaviour of x_i's local dependence on the classification… and again…

$$LDDR(x_j) = \frac{LDD^0(x_j \mid pa(x_j))}{LDD^1(x_j \mid pa(x_j))}$$

where $LDDR$ is the **Local Dependence Derivative Ratio**.

Other Results:

- $LDDR(x_j) = 1$ either if the node has no parents or $LDD^0(x_j \mid pa(x_j)) = LDD^1(x_j \mid pa(x_j))$
- $LDDR(x_j) > 1$ the class at the numerator is the supported one

Now, in an inductive way, we pass from the local to the global dependence.

We can explicit the *Bayes classifier* as:

$$f_b(X_i) = f_{nb}(X_i) \cdot \prod_{j=1}^{p} LDDR(x_j)$$

Let's call for simplicity $\prod_{j=1}^{p} LDDR(x_j) = DF$

From the previous formula we can see that the difference between an *ANB* and the Naive Bayes classifier is determinated by the product of the $LDDR$ which reflects the **global dependence distribution** (i.e. how each local dependence is distributed in each class, and how all local dependencies work together).

Theorem:

Given $X_i = (x_1, x_2, \ldots, x_p), f_b(X_i) = f_{nb}(X_i)$ under zero-one loss if and only if:

- $f_b(X_i) \geq 1 \;\&\; DF(X_i) \leq f_b(X_i)$
- $f_b(X_i) < 1 \;\&\; DF(X_i) > f_b(X_i)$

PART 1

PART 2

PART 3

References

- $f_b(X_i) > 1$ & $DF(X_i) < f_b(X_i)$.

If the distribution of the dependences satisfies certain conditions, then Naive Bayes classifies exactly the same as the underlying *ANB*, even though there may exist strong dependencies among attributes.

Focus on the specific cases that can occur:

- When $DF(X_i) = 1$, the dependencies in *ANB* has no influence on the classification
- $f_b(X_i) = f_{nb}(X_i)$ does not require that $DF(X_i) = 1$. The precise condition is given by the latter theorem.

Let's make an example to better understand the concept:

assuming that we have $f_b(X_i) = \frac{P(Y_i=0|X_i)}{P(Y_i=1|X_i)} = 0.8$, so X_i is assigned to class *0*. Being $f_b \leq 1$, we would have $DF > f_b$.

Using the equation expressed in the theorem before, we can derive $f_{nb}(X_i) = \dfrac{F_b(X_i)}{DF(X_i)}$.

Being $DF > f_b$, the whole ratio will be smaller than 1, and so **the behaviour of the Naive Bayes will be equal to the Bayesian one.**

In order to have a concrete visualization of this claim, we provide a simple example:

let's consider three equiprobable boolean features, described by x_1, x_2 and x_3 where x_1 and x_3 are independent, and x_1 and x_2 completely dependent. Therefore x_2 should be ignored.

Let's consider also two classes, denoted by 1 and 0.

The optimal classification procedure will assign an observation to class 1 if $P(x_1 \mid 1) \cdot P(x_3 \mid 1) - P(x_1 \mid 0) \cdot P(x_3 \mid 0) > 0$, to class 0 if the inequality has the opposite sign, and to an arbitrary class if the two sides are equal.

On the other hand, the Naive Bayesian classifier will take x_2 into account as if it was independent from x_1, and this will be equivalent to counting x_1 twice.

PART 1
PART 2
PART 3
References

Thus, the Naive Bayesian classifier will assign the instance to class 1 if $P(x_1 \mid 1)^2 \cdot P(x_3 \mid 1) - P(x_1 \mid 0)^2 \cdot P(x_3 \mid 0) > 0$, and to 0 otherwise.

Let $P(1 \mid x_1) = p$ and $P(1 \mid x_3) = q$. Then, for the optimal classifier, class 1 should be selected when $pq - (1-p)(1-q) > 0$, which is equivalent to $q > 1 - p$.

On the other hand with the Naive Bayesian classifier, it will be selected when $p^2 q - (1-p)^2(1-q) > 0$,which is equivalent to $q > \dfrac{(1-p)^2}{p^2 + (1-p)^2}$.

Looking at the plot below we can see the behaviour of the two classifiers as defined before.

Hide

PART 1

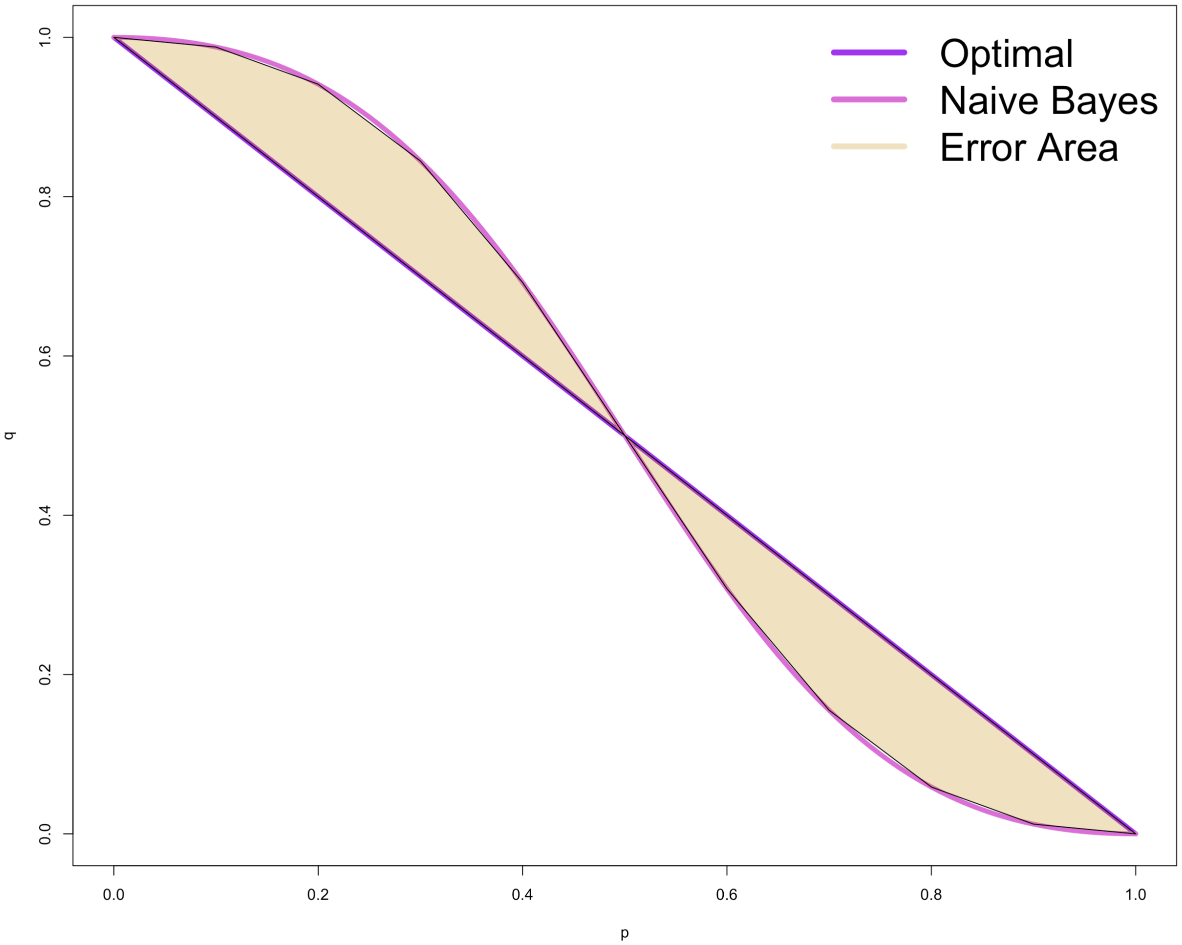
PART 2

PART 3

References

```r
# Define a function for the optimal classifier
optimal_classifier <- function(x)
{
  q = 1 - x
  return(q)
}


# Define a function for the naive bayes classifier
naive_bayes_classifier <- function(x)
{
  q = ((1 - x)^2) / (x^2 + (1 - x)^2)
  return(q)
}


# Plot the results
curve(optimal_classifier(x),col='purple',lwd=5, xlab="p", ylab = "q")
curve(naive_bayes_classifier(x),add=T, col='orchid',lwd=5)
coord = seq(0,1,0.1)
polygon(coord, naive_bayes_classifier(coord),col=rgb(0.8,0.6,0,0.3))
legend(x="topright", legend = c("Optimal", "Naive Bayes", "Error Area"
), col = c("purple","orchid",rgb(0.8,0.6,0,0.3)), lwd = 6, bty = 'n',
cex = 2.5)
```

PART 1
PART 2
PART 3
References



The remarkable fact is that, even though the independence assumption is decisively violated because x_2 is dependent from x_1, the Naive Bayesian classifier disagrees with the optimal procedure only in the two colored regions, while everywhere else it performs the correct classification. This shows that the Naive Bayesian classifier's range of applicability may be much broader than we expected.

For all problems where *p* and *q* does not fall in those two small regions, the Naive Bayesian classifier is effectively optimal.

All these considerations are valid under $L_{0,1}$ loss function.

PART 1
PART 2
PART 3
References

Running time

Ok, the classification seems to work well (at least in theory)…but ***what about the time complexity***?

Also in this compound the Naive Bayes Classifier has an excellent behaviour.

Let's analyze it deeper!

In the general case we assume that each feature has a Normally distributed PDF. This assumption is made also in the current implementation of the *naiveBayes* function in the *R* package *e1071*.

Belonging the likelihood to the exponential family (under gaussian PDF assumption), the Naive Bayes classifier corresponds to a linear classifier in a particular feature space (if the class conditional variance matrices are the same for all classes).

At this point the optimal solution of the problem will be obtained in the optimization of a linear problem which will require less computational time than non-linear problems.

Naive Bayes methods train very quickly since they require only a single pass on the data either to count frequencies (descrete variables) or to compute the normal probability density function (continuous variables under normality assumptions). So there aren't neither iterations or epoches neither backpropagation error or resolutions of an equation matrix.

So in this case it's not true that "**Who goes slow and steady wins**"…in fact the Naive Bayes Classifier is one of the faster methods, and at the same time it has a good accuracy in the classification.

But…***how is the classification accuracy evaluated***?

We define the *risk function* as the expected value of the *loss function*. For binary classification problem usually the ***0-1 loss*** is used.

This is the risk function formula:

$$R(f) = \mathbb{E}_{(Y,X)}\left[\mathbb{I}\Big(Y \neq f(X)\Big)\right]$$

PART 1
PART 2
PART 3
References

The *0-1 loss function* doesn't penalyze an inaccurate probability estimate, until than greater probability is assigned to the correct label (e.g. if you're classifing something that belongs to the *0* class with probability *0.51* or *0.99*, there is no difference in the loss function result).

PART 2

Load the required packages.

Hide

```
library(plot3D)
library(plotly)
library(caret)
library(readr)
library(ks)
library(MASS)
library(e1071)
library(tidyr)
library(plotrix)
library(doParallel)
library(rpart)
library(RCurl)
library(FactoMineR)
library(foreach)
```

Register the parameter for parallel computing, that will be used later.

Hide

```
cl <- makeCluster(3)
registerDoParallel(cl,cores=2)
```

Hide

```
set.seed(666)
```

PART 1
PART 2
PART 3
References

Load data

Let's start!

The original dataset (https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities) contains 19 activities performed by 8 people that wear sensor units on the chest (T), arms (RA and LA) and legs (RL and LL).

For this part of homework, we use just **4 activities** (walking, stepper, cross trainer, jumping) performed by a single person (the first one in the original dataset) and as covariates the measurements taken by all the **9 sensors** (x, y, z accelerometers, x, y, z gyroscopes, x, y, z magnetometers) on each of the 5 units for a total of 45 features.

Hide

```
#load("/Users/andreamarcocchia/Desktop/Statistical learning/HW2/daily-sport.RData")
load("/Users/Dario/Desktop/HW2 - Brutti/daily-sport.RData")
```

First of all, take a look at the structure of the *dailysport* dataset.

Hide

```
str(dailysport)
```

PART 1
PART 2
PART 3
References


```
## 'data.frame':    30000 obs. of  46 variables:
##  $ id      : Factor w/ 4 levels "crosstr","jumping",..: 4 4 4 4 4 4 
4 4 4 4 ...
##  $ T-xAcc  : num  8.72 8.62 8.93 9.07 9.05 ...
##  $ T-yAcc  : num  2.14 2.05 2.05 2.08 1.98 ...
##  $ T-zAcc  : num  3.83 3.45 3.28 3.23 3.41 ...
##  $ T-xGyro : num  0.294 0.23 0.209 0.197 0.238 ...
##  $ T-yGyro : num  -0.252 -0.26 -0.344 -0.241 -0.098 ...
##  $ T-zGyro : num  -0.1506 -0.13245 -0.10109 -0.06141 -0.00907 ...
##  $ T-xMag  : num  -0.577 -0.586 -0.596 -0.605 -0.61 ...
##  $ T-yMag  : num  0.1086 0.0974 0.0896 0.0825 0.075 ...
##  $ T-zMag  : num  -0.697 -0.69 -0.684 -0.675 -0.672 ...
##  $ RA-xAcc : num  8.59 8.5 9.09 9.36 9.34 ...
##  $ RA-yAcc : num  3.51 3.77 3.8 3.77 3.94 ...
##  $ RA-zAcc : num  1.58 1.64 1.62 1.68 1.83 ...
##  $ RA-xGyro: num  0.492 0.252 0.168 0.25 0.269 ...
##  $ RA-yGyro: num  -0.224 -0.381 -0.421 -0.352 -0.276 ...
##  $ RA-zGyro: num  0.532 0.65 0.706 0.597 0.373 ...
##  $ RA-xMag : num  -0.532 -0.55 -0.574 -0.594 -0.614 ...
##  $ RA-yMag : num  -0.476 -0.472 -0.461 -0.452 -0.443 ...
##  $ RA-zMag : num  -0.594 -0.582 -0.571 -0.559 -0.545 ...
##  $ LA-xAcc : num  9.22 8.47 8.49 8.43 8.08 ...
##  $ LA-yAcc : num  -2.77 -2.76 -3.02 -3.07 -3.59 ...
##  $ LA-zAcc : num  3.06 2.72 2.51 2.54 3.06 ...
##  $ LA-xGyro: num  0.301 0.0294 -0.0944 -0.6035 -1.0688 ...
##  $ LA-yGyro: num  -0.0141 -0.0514 -0.0233 0.1012 0.1644 ...
##  $ LA-zGyro: num  -0.488 -0.379 -0.342 -0.325 -0.265 ...
##  $ LA-xMag : num  -0.484 -0.492 -0.503 -0.511 -0.52 ...
##  $ LA-yMag : num  0.727 0.717 0.712 0.709 0.711 ...
##  $ LA-zMag : num  -0.256 -0.262 -0.261 -0.251 -0.229 ...
##  $ RL-xAcc : num  -11.65 -10.9 -10.09 -8.85 -8.46 ...
##  $ RL-yAcc : num  0.112 -1.163 -1.985 -0.721 0.802 ...
##  $ RL-zAcc : num  0.464 1.118 1.285 1.398 1.577 ...
##  $ RL-xGyro: num  -1.69 -1.268 -0.476 -0.305 -0.278 ...
##  $ RL-yGyro: num  0.411 0.437 0.148 0.195 0.235 ...
##  $ RL-zGyro: num  1.55 1.68 1.57 1.55 1.36 ...
##  $ RL-xMag : num  0.794 0.763 0.727 0.689 0.646 ...
```

PART 1

PART 2

PART 3

References

```
##  $ RL-yMag : num  -0.427 -0.485 -0.54 -0.587 -0.636 ...
##  $ RL-zMag : num  0.192 0.177 0.171 0.166 0.164 ...
##  $ LL-xAcc : num  -9.74 -9.53 -9.97 -9.66 -9.77 ...
##  $ LL-yAcc : num  -0.434 -0.495 1.045 -0.436 -0.523 ...
##  $ LL-zAcc : num  -1.46 -1.55 -1.27 -1.04 -1.66 ...
##  $ LL-xGyro: num  -0.291 -0.348 -0.417 -0.234 -0.344 ...
##  $ LL-yGyro: num  0.0618 0.0619 0.0789 -0.0764 0.0258 ...
##  $ LL-zGyro: num  0.32 0.291 0.297 0.478 0.407 ...
##  $ LL-xMag : num  0.8 0.805 0.811 0.817 0.823 ...
##  $ LL-yMag : num  0.437 0.43 0.423 0.412 0.397 ...
##  $ LL-zMag : num  -0.166 -0.159 -0.149 -0.144 -0.143 ...
```

Hide

```
summary(dailysport)
```

PART 1

PART 2

PART 3

References


```
##        id            T-xAcc            T-yAcc             T-zAcc

##  crosstr:7500   Min.   :-11.526   Min.   :-14.0190   Min.   :-8.363

##  jumping:7500   1st Qu.:  6.584   1st Qu.: -0.3961   1st Qu.: 1.420

##  stepper:7500   Median :  8.751   Median :  0.3336   Median : 2.848

##  walking:7500   Mean   :  9.227   Mean   :  0.3792   Mean   : 3.029

##                 3rd Qu.: 10.936   3rd Qu.:  1.3094   3rd Qu.: 3.972

##                 Max.   : 70.835   Max.   : 14.0120   Max.   :33.093

##     T-xGyro             T-yGyro             T-zGyro
##  Min.   :-4.696900   Min.   :-9.85770   Min.   :-2.1615000
##  1st Qu.:-0.330675   1st Qu.:-0.26165   1st Qu.:-0.1770200
##  Median : 0.016853   Median : 0.02464   Median :-0.0009475
##  Mean   : 0.001798   Mean   : 0.01580   Mean   : 0.0014212
##  3rd Qu.: 0.359052   3rd Qu.: 0.33622   3rd Qu.: 0.1753525
##  Max.   : 4.530300   Max.   : 5.59380   Max.   : 1.9199000
##      T-xMag             T-yMag              T-zMag             RA-xAcc

##  Min.   :-0.9246   Min.   :-0.58331   Min.   :-0.8606   Min.   :-2
5.2760
##  1st Qu.:-0.7513   1st Qu.:-0.15989   1st Qu.:-0.6281   1st Qu.: -
0.1473
##  Median :-0.6952   Median :-0.06689   Median :-0.4968   Median :
2.7679
##  Mean   :-0.7000   Mean   : 0.02185   Mean   :-0.4154   Mean   :
3.1426
##  3rd Qu.:-0.6171   3rd Qu.: 0.25817   3rd Qu.:-0.2799   3rd Qu.:
7.7249
##  Max.   :-0.4142   Max.   : 0.56106   Max.   : 0.2717   Max.   : 2
1.5890
##     RA-yAcc            RA-zAcc            RA-xGyro
##  Min.   :-10.587   Min.   :-16.557   Min.   :-7.75480
```

PART 1

PART 2

PART 3

References


```
##  1st Qu.:  2.839   1st Qu.:  1.847   1st Qu.:-0.43318
##  Median :  4.864   Median :  3.967   Median : 0.01838
##  Mean   :  6.086   Mean   :  4.166   Mean   : 0.01600
##  3rd Qu.:  7.529   3rd Qu.:  6.468   3rd Qu.: 0.47524
##  Max.   : 51.797   Max.   : 39.010   Max.   : 9.55990
##     RA-yGyro           RA-zGyro           RA-xMag
##  Min.   :-6.432800   Min.   :-4.343000   Min.   :-0.94995
##  1st Qu.:-0.365402   1st Qu.:-0.654562   1st Qu.:-0.46215
##  Median :-0.014138   Median :-0.017570   Median :-0.22478
##  Mean   :-0.009334   Mean   : 0.002844   Mean   :-0.25362
##  3rd Qu.: 0.362628   3rd Qu.: 0.635118   3rd Qu.:-0.01952
##  Max.   : 4.853700   Max.   : 5.177800   Max.   : 0.93707
##     RA-yMag           RA-zMag            LA-xAcc            LA-yAcc

##  Min.   :-1.1059   Min.   :-1.1374   Min.   :-31.6400   Min.   :-5
7.939
##  1st Qu.:-0.7037   1st Qu.:-0.6716   1st Qu.:  0.6757   1st Qu.: -
6.833
##  Median :-0.5470   Median :-0.5559   Median :  4.3583   Median : -
4.362
##  Mean   :-0.4846   Mean   :-0.5029   Mean   :  4.0236   Mean   : -
5.387
##  3rd Qu.:-0.2958   3rd Qu.:-0.3964   3rd Qu.:  8.1350   3rd Qu.: -
2.387
##  Max.   : 0.3295   Max.   : 0.3614   Max.   : 50.9240   Max.   : 3
0.246
##     LA-zAcc            LA-xGyro            LA-yGyro
##  Min.   :-23.7440   Min.   :-8.833200   Min.   :-8.381800
##  1st Qu.:  0.7575   1st Qu.:-0.469542   1st Qu.:-0.382228
##  Median :  3.2938   Median :-0.004280   Median :-0.024567
##  Mean   :  2.9418   Mean   :-0.007128   Mean   :-0.004787
##  3rd Qu.:  6.2104   3rd Qu.: 0.426720   3rd Qu.: 0.368470
##  Max.   : 24.6540   Max.   :18.809000   Max.   : 4.358900
##     LA-zGyro            LA-xMag             LA-yMag
##  Min.   :-12.935000   Min.   :-0.93608   Min.   :-0.9267
##  1st Qu.: -0.624395   1st Qu.:-0.49630   1st Qu.: 0.3048
##  Median :  0.027278   Median :-0.03208   Median : 0.4356
##  Mean   : -0.008746   Mean   :-0.14015   Mean   : 0.3854
```

PART 1
PART 2
PART 3
References

```
##  3rd Qu.:  0.622652   3rd Qu.: 0.21392   3rd Qu.: 0.5705
##  Max.   :  5.679100   Max.   : 0.89720   Max.   : 0.8499
##     LA-zMag           RL-xAcc            RL-yAcc            RL-zAcc

##  Min.   :-1.1028   Min.   :-89.704   Min.   :-52.727   Min.   :-47.
9250
##  1st Qu.:-0.6849   1st Qu.:-11.917   1st Qu.: -1.034   1st Qu.: -1.
1442
##  Median :-0.4257   Median : -9.704   Median :  1.115   Median : -0.
1701
##  Mean   :-0.2640   Mean   :-10.051   Mean   :  1.084   Mean   : -0.
2483
##  3rd Qu.: 0.1127   3rd Qu.: -7.624   3rd Qu.:  3.362   3rd Qu.:  0.
8341
##  Max.   : 0.7589   Max.   :  3.008   Max.   : 80.427   Max.   : 17.
0360
##     RL-xGyro           RL-yGyro            RL-zGyro
##  Min.   :-5.14820   Min.   :-2.081100   Min.   :-3.185800
##  1st Qu.:-0.52341   1st Qu.:-0.198860   1st Qu.:-1.127500
##  Median : 0.04027   Median : 0.008418   Median :-0.064018
##  Mean   : 0.01691   Mean   : 0.012355   Mean   :-0.003762
##  3rd Qu.: 0.54979   3rd Qu.: 0.219955   3rd Qu.: 1.074200
##  Max.   : 6.01390   Max.   : 3.311100   Max.   : 3.665900
##     RL-xMag           RL-yMag            RL-zMag            LL-xAcc

##  Min.   :0.3556   Min.   :-0.7856   Min.   :-0.62804   Min.   :-93.
940
##  1st Qu.:0.4882   1st Qu.:-0.2595   1st Qu.:-0.32134   1st Qu.:-11.
757
##  Median :0.6394   Median :-0.1207   Median :-0.12695   Median : -9.
556
##  Mean   :0.6510   Mean   :-0.1197   Mean   :-0.01181   Mean   : -9.
989
##  3rd Qu.:0.8168   3rd Qu.:-0.0391   3rd Qu.: 0.31716   3rd Qu.: -7.
699
##  Max.   :1.0715   Max.   : 0.7485   Max.   : 0.57962   Max.   :  3.
592
##     LL-yAcc            LL-zAcc             LL-xGyro
```

PART 1

PART 2

PART 3

References

```
##  Min.   :-95.8980   Min.   :-27.1510   Min.   :-7.15920
##  1st Qu.: -4.1598   1st Qu.: -1.4636   1st Qu.:-0.56632
##  Median : -1.7903   Median : -0.3309   Median :-0.06929
##  Mean   : -1.7952   Mean   : -0.4361   Mean   :-0.03791
##  3rd Qu.:  0.4201   3rd Qu.:  0.7115   3rd Qu.: 0.48713
##  Max.   : 49.4040   Max.   : 33.0620   Max.   :10.38100
##     LL-yGyro            LL-zGyro            LL-xMag
##  Min.   :-3.605000   Min.   :-4.115000   Min.   :0.2951
##  1st Qu.:-0.227410   1st Qu.:-1.116200   1st Qu.:0.4771
##  Median : 0.004304   Median : 0.018875   Median :0.6287
##  Mean   : 0.004683   Mean   :-0.006432   Mean   :0.6244
##  3rd Qu.: 0.231900   3rd Qu.: 1.148025   3rd Qu.:0.7691
##  Max.   : 4.207600   Max.   : 3.526500   Max.   :1.0377
##     LL-yMag            LL-zMag
##  Min.   :-0.7177   Min.   :-0.52303
##  1st Qu.: 0.1047   1st Qu.:-0.27879
##  Median : 0.2789   Median : 0.01363
##  Mean   : 0.2626   Mean   :-0.02575
##  3rd Qu.: 0.4749   3rd Qu.: 0.14591
##  Max.   : 1.0249   Max.   : 0.62156
```

As we can see in the summary, our dataset is composed by 7500 for each activity made by the selected person. So the four classes are numerically equidistributed.

Now we want to reduce the dataset into **two activities** (*jumping*, *crosstr*) and only **three sensors**. We decide to use the *x-y-z* sensor from the right-leg accelerometer.

Hide

```
data_reduce <- subset(dailysport,dailysport$id == "jumping" | dailyspo
rt$id == "crosstr")
ds.small <- data_reduce[,c(1,29:31)]
ds.small <- droplevels(ds.small)
```

Let's start with plot! For simplicity we plot only 500 points for each class.

We start creating two datasets:

PART 1
PART 2
PART 3
References

1. jumping
2. cross training

Hide

```
jump <- ds.small[ds.small$id == "jumping",]
head(jump)
```

```
##              id   RL-xAcc  RL-yAcc RL-zAcc
## 22501 jumping  -0.32286 -0.63404  1.3314
## 22502 jumping  -4.98630  2.61600 -1.8274
## 22503 jumping -38.33700 15.74600 -2.0916
## 22504 jumping -13.17800 -9.93740  4.0246
## 22505 jumping -26.50700 -6.85620 -2.8498
## 22506 jumping -20.40300 -2.82580 -3.0242
```

Hide

```
cross <- ds.small[ds.small$id == "crosstr",]

head(cross)
```

```
##              id  RL-xAcc   RL-yAcc   RL-zAcc
## 15001 crosstr -11.5280  1.201200 -0.072049
## 15002 crosstr -10.8800 -0.002383 -0.365320
## 15003 crosstr -10.7760  1.952300  0.377370
## 15004 crosstr  -9.6741  4.190000  0.742190
## 15005 crosstr  -8.2311  2.797600  1.500400
## 15006 crosstr  -9.1227 -0.911080  1.614100
```

In the next chunk we pick 1000 observations: 500 from *jumping* and 500 from *cross training* activity.

Hide

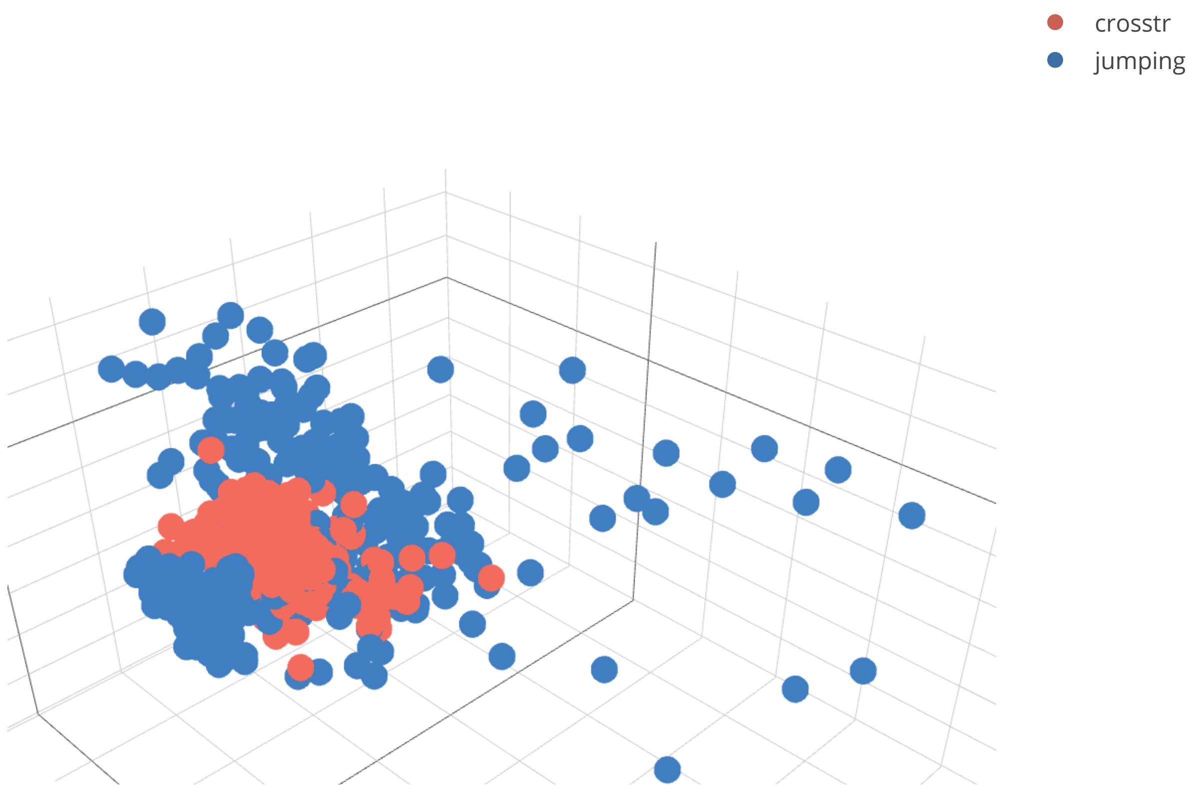
PART 1
PART 2
PART 3
References

```
# Randomly select the indexes
idxxx <- sample(1:length(jump$id),size = 500,replace=F)

# Create the new dataset
new_data=rbind(jump[idxxx,],cross[idxxx,])
```

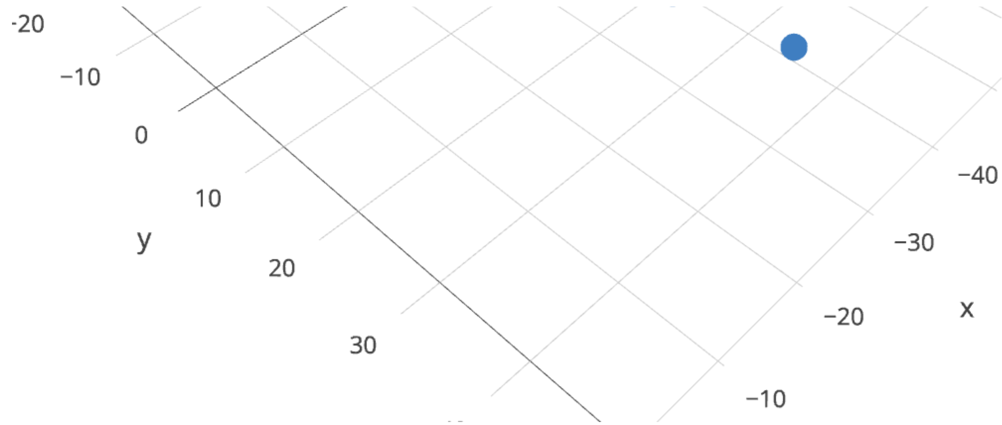
Here we represent the point cloud using the *plot_ly* function, in order to obtain an interactive plot.

Hide

```
plot_ly(x = new_data$`RL-xAcc`, y = new_data$`RL-yAcc`, z = new_data$`
RL-zAcc`, color = new_data$id , colors = c('#BF382A', '#0C4B8E'), alph
a = 0.8) %>% add_markers()
```



PART 1
PART 2
PART 3
References



What a mess! Seems to be hard to classify this points…

Now we are going to split our dataset in **train** and **test**:

Hide

```
# Define the number of elements in the test
# In this case is the 30%
n_test <- round(0.3*nrow(ds.small))

# Randomly select the elements that will belong to test set
idx_test <- sort(sample(1:nrow(ds.small), n_test))

head(idx_test)
```

```
## [1]  9 11 13 16 36 38
```

Hide

PART 1

PART 2

PART 3

References

```
# Create the test set
ds.test <- ds.small[idx_test,]

# Create the train set
ds.train <- ds.small[setdiff(1:nrow(ds.small),idx_test),]
```

Last operations before starting the predictions…in the next chunk we want to scale the data:

Hide

```
test_scalato <- ds.test
# Scale all the numeric variables (all except the first one)
test_scalato[,2:4] <- scale(ds.test[,2:4])

train_scalato <- ds.train
train_scalato[,2:4] <- scale(ds.train[,2:4])
```

Ok, we have all the elements to start the predictions.

We'll try different methods, in order to obtain the best result.

The first method that we apply is the **LDA (Linear Discriminant Analysis)** that is a method used in statistics and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects.

The resulting combination may be used as a linear classifier or, more commonly, for dimensionality reduction before later classification. In this project we'll use it as a linear classifier.

It is important to understand the assumptions for its implementation. LDA assumes that the conditional probability density functions (PDF) $p(\bar{x} \mid y = 0)$ and $p(\bar{x} \mid y = 1)$ are both normally distributed with mean and covariance parameters equal to $(\bar{\mu}_0, \Sigma_0)$ and $(\bar{\mu}_1, \Sigma_1)$.

PART 1
PART 2
PART 3
References

Moreover LDA makes the additional simplifying **homoscedasticity assumption**, assuming that the variance for each feature is the same in each classes. It's called **pooled variance estimate**.

Some problems could arise due to the presence of outliers.

In the LDA method the size of the smallest group must be larger than the number of predictor variables, as in our case (7500 observations in each class and a smaller number of variables).

In the next chunk we train the algorithm in the train set and we check the results (**accuracy**) in the train set itself. We'll use scaled data, to avoid problems related to the computation of the pooled variance.

Hide

```
# Estimate the parameter in the train set
lda.out <- lda(id ~ . , data = train_scalato)

# Predict the result
pred.tr <- predict(lda.out, train_scalato)

# Estimate the error
mean(pred.tr$class == train_scalato$id)
```

```
## [1] 0.5938095
```

Now it's time to check how our classifier works in another dataset…this is why we have the **test** one!

We have to remember that the train gives us an optimistic evaluation of our model, so we need to verify how good the model is on the test set.

To summarize the results we use the *confusionMatrix* function provided by *caret* package.

Let's check:

PART 1
PART 2
PART 3
References

Hide

```
# Predict the results for the test set
pred.te <- predict(lda.out, test_scalato[,-1])

# Print the confusion matrix
confusionMatrix(pred.te$class, test_scalato$id)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping
##    crosstr    1380     953
##    jumping     856    1311
##
##                Accuracy : 0.598
##                  95% CI : (0.5835, 0.6124)
##     No Information Rate : 0.5031
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.1962
##  Mcnemar's Test P-Value : 0.024
##
##             Sensitivity : 0.6172
##             Specificity : 0.5791
##          Pos Pred Value : 0.5915
##          Neg Pred Value : 0.6050
##              Prevalence : 0.4969
##          Detection Rate : 0.3067
##    Detection Prevalence : 0.5184
##       Balanced Accuracy : 0.5981
##
##        'Positive' Class : crosstr
##
```

We notice that LDA performance is not so satisfactory and probably the assumptions aren’t the better choice for our situation.

PART 1
PART 2
PART 3
References

Howewer we can mantain the **gaussian distributed PDF assumption**, and try to use a simpler model…or better *Naive*!

Let's now focus on the **Naive Bayes Classifier**. In the next chunk we train the algorithm on the train set:

Hide

```
bayes.tr <- naiveBayes(id ~ . , data = ds.train,type = "raw")

pred.baye.tr <- predict(bayes.tr, ds.train[,-1])

mean((pred.baye.tr == ds.train$id))
```

```
## [1] 0.8935238
```

At a glance we obtain a better result comparing with the *LDA*.

Let's check how it goes in the test:

Hide

```
# Test the naive bayes prediction on the test set

pred.baye.te <- predict(bayes.tr, ds.test[,-1])

confusionMatrix(pred.baye.te, ds.test$id)
```

PART 1
PART 2
PART 3
References

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping
##    crosstr    2099     329
##    jumping     137    1935
##
##                Accuracy : 0.8964
##                  95% CI : (0.8872, 0.9052)
##     No Information Rate : 0.5031
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.793
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9387
##             Specificity : 0.8547
##          Pos Pred Value : 0.8645
##          Neg Pred Value : 0.9339
##              Prevalence : 0.4969
##          Detection Rate : 0.4664
##    Detection Prevalence : 0.5396
##       Balanced Accuracy : 0.8967
##
##        'Positive' Class : crosstr
##
```

The result obtained through the test set, using the Naive bayes, is improved of about 30 percentage points.

Confidence Interval

Now we have to estimate the 1-dimensional class–conditional densities of each of the three covariates using histograms or kernels and build 95% **bootstrap confidence bands** around them.

PART 1
PART 2
PART 3
References

Let’s start estimating the KDE using the function *density* for each of the three variables within the two classes:

Hide

```
jump_x_density <- density(jump$`RL-xAcc`)
jump_y_density <- density(jump$`RL-yAcc`)
jump_z_density <- density(jump$`RL-zAcc`)

cross_x_density <- density(cross$`RL-xAcc`)
cross_y_density <- density(cross$`RL-yAcc`)
cross_z_density <- density(cross$`RL-zAcc`)
```

Now we define a function to build a boostrap confidence bands:

Hide

PART 1
PART 2
PART 3
References

```
ConfidenceBands_density <- function(dataset, from = -100, to = 100, n
 = 1000, evalpoints = 1000, B = 2000, coord = c("x","y","z"))
{
  # Iterate over the variables in the dataset (excluded the first one,
 that is the id)
  for (elemento in 2:ncol(dataset))
  {
    # Save the column values
    x <- dataset[,elemento]
    xax <- seq(from,to,length.out = evalpoints)

    # Estimate the Kernel density
    d <- density(x,n=evalpoints,from=from,to=to)

    # Create an empty matrix in which will be stored the bootsrapping
 sample
    # Number of columns is equal to the number of bootstrap iteration
    estimates <- matrix(NA,nrow=evalpoints,ncol=B)

    # Iterate over each bootstrap repetition
    for (b in 1:B)
      {
        # Sample from the observation in the selected column, with rep
etition
        xstar <- sample(x,replace=TRUE)

        # Evaluate the density estimation for the bootstrap sample
        dstar <- density(xstar,n=evalpoints,from=from,to=to)

        # Fill the column with the y values, associated with xstar
        estimates[,b] <- dstar$y
      }

    # Set the 95% probability using the prameter probs
    ConfidenceBands <- apply(estimates, 1, quantile, probs = c(0.025,
0.975))
```

PART 1
PART 2
PART 3
References

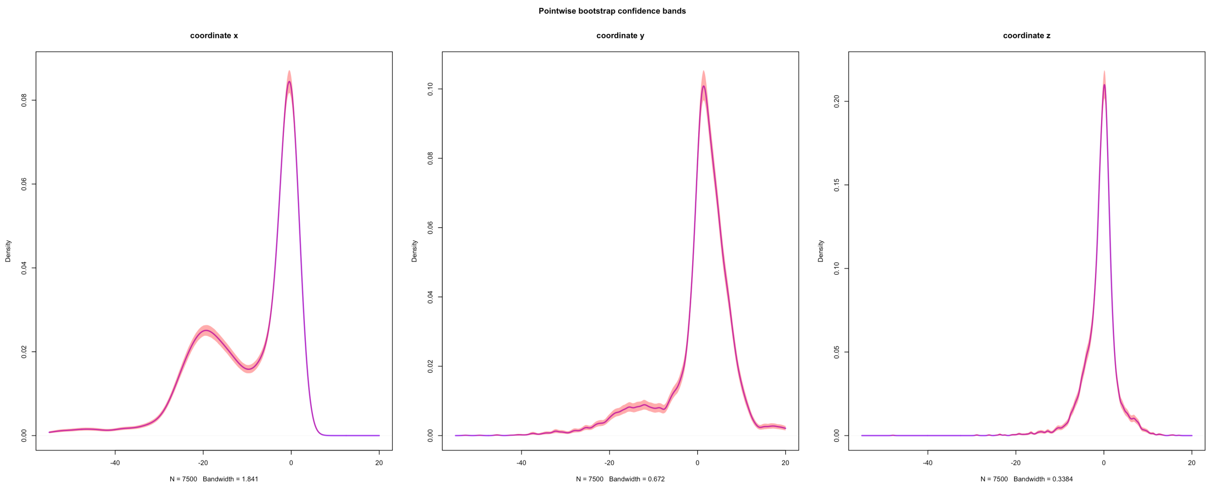
```
    # Plot
    plot(d,lwd=2,col="purple",ylim=c(0,max(ConfidenceBands[2,]*1.01)),
main=paste("coordinate", coord[elemento - 1]))
    xshade <- c(xax,rev(xax))
    yshade <- c(ConfidenceBands[2,],rev(ConfidenceBands[1,]))
    polygon(xshade,yshade, border = NA,col=adjustcolor("red", 0.4))
  }

  # Add the title
  title("Pointwise bootstrap confidence bands", outer = T)

}
```

Let’s take a look!

Here we plot the confidence bands for three variables associated to the *jumping* class:

Hide

```
par(mfrow=c(1,3), oma=c(0,0,2,0))
ConfidenceBands_density(dataset = jump,from = -55, to = 20)
```

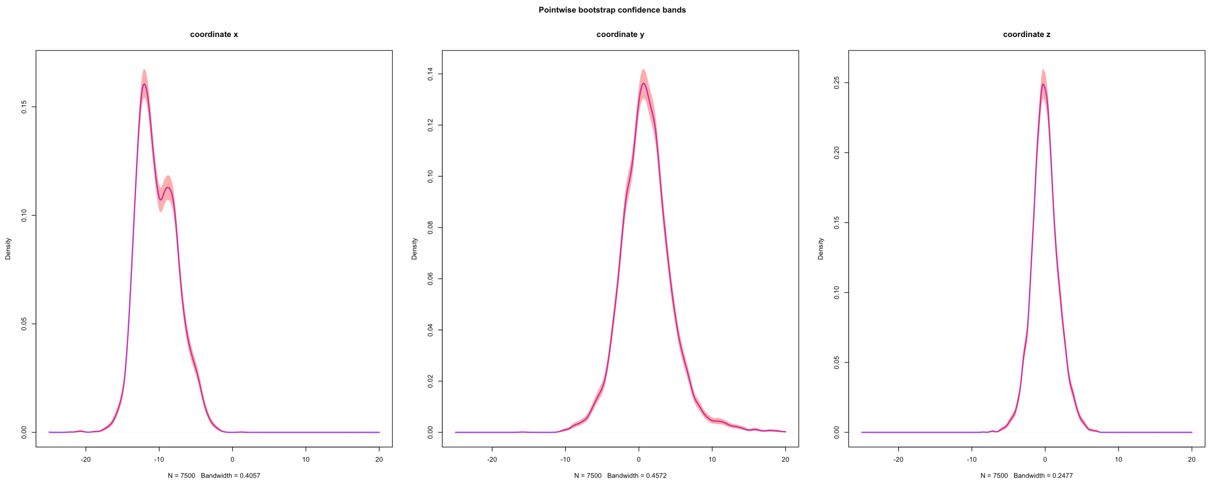


And here there are the plots for the *cross train* activity:

PART 1
PART 2
PART 3
References

Hide

```
par(mfrow=c(1,3), oma=c(0,0,2,0))
ConfidenceBands_density(dataset = cross,from = -25, to = 20)
```



Hide

```
par(mfrow=c(1,1), oma=c(0,0,0,0))
```

Analyzing the plot above, it’s possible to see that the x variable, in both classes, seems not to follow the behaviour of a gaussian distribution.

In the peaks the confidence bands are larger, and so the density estimation is less precise.

Naive Bayes Classifier hand-made

Well, now it’s time to make a first-hand experience with Naive Bayes!

In this part of the project we **implement our version of the Naive Bayes Classifier**. In the *naiveBayes* function is assumed that all the features have a Normally distributed PDF, while we decide to use a **non-parametric approach**.

PART 1
PART 2
PART 3
References

In fact we estimate the features' PDF using a **Kernel Density Estimator** (function *density*); as we have seen before the assumption about gaussian PDF is not always respected.

We implement our classifier with two functions:

- *naive_bayes2* ⇒ function that requires in input the dataframe in which the model has to be trained, the kernel type (gaussian kernel as default) and the column position in which are stored the labels in the dataframe (first column as default). The output of this function is a list whose values are the estimations that will be used in the other function.

- *predizione* ⇒ function that requires in input the list of values obtained with *naive_bayes2*, the test dataset and a value to use as *pseudocount*. In this function new observations are classified. The output is a list of labels, one for each row in the test dataset.

Just one more detail: **if a given class and feature value never occur together in the training data**, then the frequency-based probability estimate will be zero.

This is an issue because the Likelihood will be equal to 0 so all the informations will be wiped out. Therefore, it is often desirable to incorporate a small-sample correction, called **pseudocount**, in all probability estimates such that no probability is ever set to be exactly zero.

A regularizing way for *Naive Bayes* is called **Laplace smoothing**, and it's when the pseudocount is one. In our implementation we don't set the *pseudocount* value equal to 1, but we want to penalyze the likelihood when this situation occurs, by using a number really close to 0. We find the smaller value that *R* is able to represent, and we use as *pseudocount* a number a bit bigger.

Here we have the *naive_bayes2* function:

Hide

PART 1
PART 2
PART 3
References

```
naive_bayes2 <- function(train, kern = "gaussian", pos = 1)
{
  # Check if the kernel in input is supported by the density function
  "%ni%" <-  Negate("%in%")
  if (kern %ni% c("gaussian", "epanechnikov", "rectangular", "triangul
ar", "biweight", "cosine", "optcosine"))
  {
    stop("Invalid name for Kernel")
  }

  # Check if the train dataset is a dataframe
  if (is.data.frame(train)==F)
  {
    stop("Invalid structure for train. Required data frame")
  }

  if (pos != 1)
  {
    if (pos==ncol(train))
    {
      train <- train[,c(pos, 1:ncol(train)-1)]
    }
    else
    {
      train <- train[,c(pos,1:(pos-1),(pos+1):ncol(train))]
    }
  }

  # Store in a vector all the labels
  id <- train[,1]

  # Store all the different classes that appear in the dataset
  label <- levels(id)

  # Create an empty list that will be use to store final result
  biggg <- list()
```

PART 1
PART 2
PART 3
References

```
  # Start a loop that iterate over the different labels
  for (i in 1:length(label))
  {
    # Select all the elements that belong to a particular class
    aux <- train[train[,1]==label[i],]

    # Evaluate the ratio between the number of element in the class ov
er the total number of observations
    prop <- length(aux[,1])/length(train[,1])
    indice <- 1
    listone <- list()

    # Start a loop over the variables
    for (j in 2:ncol(aux))
    {
      # Estimate the density for a certain column
      density_estimation <- density(aux[,j], kernel = kern)

      # Store in a list the values of x and y obtained with the densit
y in the previous operation
      lll <- list(x = density_estimation$x, y = density_estimation$y)

      # Add lll to listone in position index
      listone[[indice]] <- lll

      indice <- indice + 1
    }

    # Create a list with two elements:
    # listone --> inside listone there are other lists, as many as the
 number of variables
    # prop --> a number that represents the proportion of observations
 that belong to this class
    auxiliar_55 <- list(val=listone, prop = prop)

    # the biggg list will have as many elements as many the class are
    biggg[[i]] <- auxiliar_55
```

PART 1
PART 2
PART 3
References

```
  }
  return(biggg)
}
```

…and here the *predizione* function:

Hide

PART 1
PART 2
PART 3
References

```
predizione <- function(lista_par, test_set, pseudocount = 2.225074e-20
8)
{
  # Check if lista_par is a list object
  if (typeof(lista_par) != "list")
  {
    stop("Invalid type for lista_par")
  }

  # Check if the test_set dataset is a dataframe
  if (is.data.frame(test_set)==F)
  {
    stop("Invalid structure for test_set. Required data frame")
  }

  final_ris <- NULL

  # Store the number of variables in num_var
  # The number of variables has to be the same as the number of variab
les inside lista_par
  num_var <- length(test_set[1,]) - 1

  # Store all the different classes that appear in the dataset
  label <- levels(test_set[,1])

  # Iterate over the number of rows in the dataset (test_set)
  for (datass in 1:nrow(test_set))
  {
    ris <- NULL

    # Iterate over the number of labels (same as the length of lista_p
ar)
    for (i in 1:length(lista_par))
    {
      lik <- NULL

      # Iterate over the variables
```

PART 1
PART 2
PART 3
References

```
      for (j in 1:num_var)
      {
        mom <- approx(lista_par[[i]][[1]][[j]]$x ,lista_par[[i]][[1]]
[[j]]$y , xout = test_set[datass,][j+1])$y

        if (is.na(mom))
        {
          lik[j] <-  pseudocount
        }

        else
        {
          lik[j] <-  mom
        }
      }

      # Insert for a certain row the values of a certain class
      # For values we mean the product between the prior and the likel
ihood
      ris[i] <- prod(lik) * lista_par[[i]][[2]]
    }

    # Find the class with the bigger value (max probability)
    massimo <- which.max(ris)

    # Extract the corresponding label
    final_ris[datass] <- label[massimo]
  }

  return(final_ris)
}
```

Ok, the most is done. Now we have to use our functions! Let's start to train the model on the *ds.train* dataframe, and then test it on the *ds.test* dataframe.

Hide

PART 1
PART 2
PART 3
References

```
# Train the model
wer <- naive_bayes2(ds.train)

# Make the prediction on the test set
ret <- predizione(wer,ds.test)

# Evaluate the performance
ww <- NULL
for (i in 1:length(ret))
{
  ww[i] <- ret[i] == ds.test$id[i]
}

mean(ww)
```

```
## [1] 0.9148889
```

Great! Our model performs very well, even better than the *naiveBayes* function from *e1071* packege. Probably this result is reached because we use a non-parametric approach, and so our density estimation fits better than the one under normality assumption.

PART 3

Up to now we have worked on two activities and three variables. From now on we will use the entire dataset. So our data will be composed by:

- 4 activities
- 1 class variable (*id*)
- 45 predictors

Now let's try the classification on the whole dataset. We start using **Naive Bayes** from the package.

Hide

PART 1
PART 2
PART 3
References

```
# Define the number of elements in the test
# In this case is the 30%
n_test <- round(0.3*nrow(dailysport))

# Randomly select the elements that will belong to test set
idx_test <- sort(sample(1:nrow(dailysport), n_test))

# Create the test set
daily_test <- dailysport[idx_test,]

# Create the train set
daily_train <- dailysport[setdiff(1:nrow(dailysport),idx_test),]
```

Hide

```
tr.bayes <- naiveBayes(id~., data = daily_train)

tr.prediction <- predict(tr.bayes, daily_test[,-1])

confusionMatrix(tr.prediction, daily_test$id)
```

PART 1
PART 2
PART 3
References

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    2261       0       0       0
##    jumping       0    2251       0       0
##    stepper       3       0    2247       0
##    walking       0       0       0    2238
##
## Overall Statistics
##
##                Accuracy : 0.9997
##                  95% CI : (0.999, 0.9999)
##     No Information Rate : 0.2516
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9996
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9987         1.0000         1.0000
## Specificity                  1.0000         1.0000         0.9996
## Pos Pred Value               1.0000         1.0000         0.9987
## Neg Pred Value               0.9996         1.0000         1.0000
## Prevalence                   0.2516         0.2501         0.2497
## Detection Rate               0.2512         0.2501         0.2497
## Detection Prevalence         0.2512         0.2501         0.2500
## Balanced Accuracy            0.9993         1.0000         0.9998
##                      Class: walking
## Sensitivity                  1.0000
## Specificity                  1.0000
## Pos Pred Value               1.0000
## Neg Pred Value               1.0000
## Prevalence                   0.2487
## Detection Rate               0.2487
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence          0.2487
## Balanced Accuracy             1.0000
```

Perfect prediction!

Let's try to do something different.

We're going to apply the **PCA (Principal Components Analysis)** method in order to reduce the variable dimension, and let's check if the accuracy of the model is as good as the previous one.

To perform the Principal Component Analysis we use the function *PCA* from package *FactoMineR*.

You may ask…***why do we use PCA instead of LDA to make dimensionality reduction***?

PCA creates a new space defining axes orthogonal and maximizes the variance between the variables. On the other hand LDA maximizes the distance of the labels but the new axes are not orthogonal.

Remember that the core assumption of Naive Bayes is the independence between features, so applying PCA is a good idea.

Let's start the dimensionality reduction using PCA:

Hide

```
data_pca <- PCA(dailysport, ncp = 11, quali.sup = 1, graph = F)
```

Check the results…***how many components do we have to grab?*** For sure all the variability is explained taking all the factors.

Let's analyze the summary:

Hide

```
summary(data_pca)
```

PART 1
PART 2
PART 3
References

```
##
## Call:
## PCA(X = dailysport, ncp = 11, quali.sup = 1, graph = F)
##
##
## Eigenvalues
##                        Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.
6
## Variance               6.640   6.199   4.716   3.161   2.615   2.57
4
## % of var.             14.755  13.776  10.481   7.024   5.811   5.71
9
## Cumulative % of var.  14.755  28.531  39.011  46.035  51.847  57.56
6
##                        Dim.7   Dim.8   Dim.9  Dim.10  Dim.11  Dim.1
2
## Variance               2.238   1.939   1.382   1.323   1.118   0.95
4
## % of var.              4.974   4.309   3.070   2.939   2.484   2.11
9
## Cumulative % of var.  62.540  66.849  69.920  72.859  75.343  77.46
2
##                       Dim.13  Dim.14  Dim.15  Dim.16  Dim.17  Dim.1
8
## Variance               0.916   0.840   0.751   0.720   0.611   0.56
6
## % of var.              2.036   1.867   1.668   1.600   1.359   1.25
8
## Cumulative % of var.  79.498  81.365  83.033  84.632  85.991  87.24
9
##                       Dim.19  Dim.20  Dim.21  Dim.22  Dim.23  Dim.2
4
## Variance               0.541   0.486   0.465   0.425   0.387   0.36
4
## % of var.              1.203   1.081   1.033   0.944   0.860   0.80
9
## Cumulative % of var.  88.451  89.532  90.565  91.509  92.369  93.17
```

PART 1
PART 2
PART 3
References

```
8
##                        Dim.25  Dim.26  Dim.27  Dim.28  Dim.29  Dim.3
0
## Variance                0.351   0.328   0.294   0.277   0.240   0.21
4
## % of var.               0.781   0.730   0.652   0.615   0.534   0.47
6
## Cumulative % of var.  93.958  94.688  95.340  95.955  96.489  96.96
5
##                        Dim.31  Dim.32  Dim.33  Dim.34  Dim.35  Dim.3
6
## Variance                0.208   0.185   0.175   0.127   0.114   0.10
3
## % of var.               0.463   0.411   0.388   0.283   0.254   0.22
9
## Cumulative % of var.  97.428  97.839  98.228  98.511  98.765  98.99
4
##                        Dim.37  Dim.38  Dim.39  Dim.40  Dim.41  Dim.4
2
## Variance                0.092   0.083   0.077   0.061   0.048   0.04
2
## % of var.               0.205   0.185   0.171   0.136   0.107   0.09
4
## Cumulative % of var.  99.200  99.385  99.556  99.692  99.799  99.89
3
##                        Dim.43  Dim.44  Dim.45
## Variance                0.020   0.017   0.011
## % of var.               0.045   0.038   0.024
## Cumulative % of var.  99.938  99.976 100.000
##
## Individuals (the 10 first)
##                   Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2
Dim.3
## 1            |  4.671 | -0.151  0.000  0.001 |  0.253  0.000  0.003 | -
0.418
## 2            |  4.499 | -0.160  0.000  0.001 |  0.275  0.000  0.004 | -
0.417
## 3            |  4.251 | -0.118  0.000  0.001 |  0.247  0.000  0.003 | -
```

PART 1
PART 2
PART 3
References

```
0.324
## 4         |  4.237 | -0.114  0.000  0.001 |  0.314  0.000  0.005 | -
0.486
## 5         |  4.299 | -0.115  0.000  0.001 |  0.259  0.000  0.004 | -
0.500
## 6         |  4.227 | -0.136  0.000  0.001 |  0.139  0.000  0.001 | -
0.475
## 7         |  4.157 | -0.138  0.000  0.001 |  0.016  0.000  0.000 | -
0.558
## 8         |  4.521 | -0.073  0.000  0.000 | -0.042  0.000  0.000 | -
0.625
## 9         |  4.964 | -0.226  0.000  0.002 |  0.111  0.000  0.000 | -
0.385
## 10        |  4.762 | -0.227  0.000  0.002 |  0.086  0.000  0.000 | -
0.390
##              ctr   cos2
## 1          0.000  0.008 |
## 2          0.000  0.009 |
## 3          0.000  0.006 |
## 4          0.000  0.013 |
## 5          0.000  0.014 |
## 6          0.000  0.013 |
## 7          0.000  0.018 |
## 8          0.000  0.019 |
## 9          0.000  0.006 |
## 10         0.000  0.007 |
##
## Variables (the 10 first)
##              Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3
ctr
## T-xAcc    |  0.098  0.144  0.010 | -0.442  3.152  0.195 |  0.798 13.
509
## T-yAcc    |  0.239  0.860  0.057 |  0.080  0.104  0.006 |  0.097  0.
198
## T-zAcc    |  0.259  1.014  0.067 | -0.479  3.708  0.230 |  0.621  8.
168
## T-xGyro   | -0.022  0.008  0.001 | -0.038  0.023  0.001 | -0.115  0.
281
```

PART 1
PART 2
PART 3
References

```
## T-yGyro  |  0.004  0.000  0.000 | -0.069  0.077  0.005 |  0.043  0.
039
## T-zGyro  | -0.019  0.006  0.000 | -0.096  0.150  0.009 | -0.016  0.
006
## T-xMag   | -0.291  1.279  0.085 | -0.630  6.410  0.397 | -0.330  2.
308
## T-yMag   |  0.408  2.506  0.166 | -0.631  6.422  0.398 | -0.466  4.
603
## T-zMag   |  0.891 11.957  0.794 |  0.132  0.280  0.017 |  0.014  0.
004
## RA-xAcc  |  0.372  2.085  0.138 |  0.406  2.657  0.165 |  0.159  0.
535
##            cos2
## T-xAcc    0.637 |
## T-yAcc    0.009 |
## T-zAcc    0.385 |
## T-xGyro   0.013 |
## T-yGyro   0.002 |
## T-zGyro   0.000 |
## T-xMag    0.109 |
## T-yMag    0.217 |
## T-zMag    0.000 |
## RA-xAcc   0.025 |
##
## Supplementary categories
##                 Dist      Dim.1     cos2   v.test      Dim.2     cos
2
## crosstr  |    2.794 |   -2.032    0.529  -78.853 |   -0.523    0.03
5
## jumping  |    3.887 |    1.895    0.238   73.551 |   -2.876    0.54
7
## stepper  |    3.055 |   -2.119    0.481  -82.252 |    1.604    0.27
6
## walking  |    3.189 |    2.256    0.501   87.554 |    1.795    0.31
7
##            v.test      Dim.3     cos2   v.test
## crosstr   -21.012 |    0.281    0.010   12.935 |
## jumping  -115.496 |   -1.698    0.191  -78.196 |
```

PART 1
PART 2
PART 3
References

```
## stepper    64.429 |    0.951    0.097   43.774 |
## walking    72.079 |    0.467    0.021   21.487 |
```

There are many ways to decide the number of factors to take. Let's see three of them:

- **Value of the eigenvalues bigger than one**

Hide

```
head(data_pca$eig, 13)
```

PART 1

PART 2

PART 3

References


```
##         eigenvalue percentage of variance
## comp 1   6.6396393              14.754754
## comp 2   6.1990901              13.775756
## comp 3   4.7163879              10.480862
## comp 4   3.1607857               7.023968
## comp 5   2.6150947               5.811322
## comp 6   2.5737687               5.719486
## comp 7   2.2383847               4.974188
## comp 8   1.9390767               4.309059
## comp 9   1.3816916               3.070426
## comp 10  1.3225830               2.939073
## comp 11  1.1179952               2.484434
## comp 12  0.9535928               2.119095
## comp 13  0.9161916               2.035981
##         cumulative percentage of variance
## comp 1                           14.75475
## comp 2                           28.53051
## comp 3                           39.01137
## comp 4                           46.03534
## comp 5                           51.84666
## comp 6                           57.56615
## comp 7                           62.54034
## comp 8                           66.84940
## comp 9                           69.91982
## comp 10                          72.85889
## comp 11                          75.34333
## comp 12                          77.46242
## comp 13                          79.49840
```

In this case the “right” choice would be 11 factors.

- **Elbow method on screeplot (barplot of eigenvalues)**

Hide

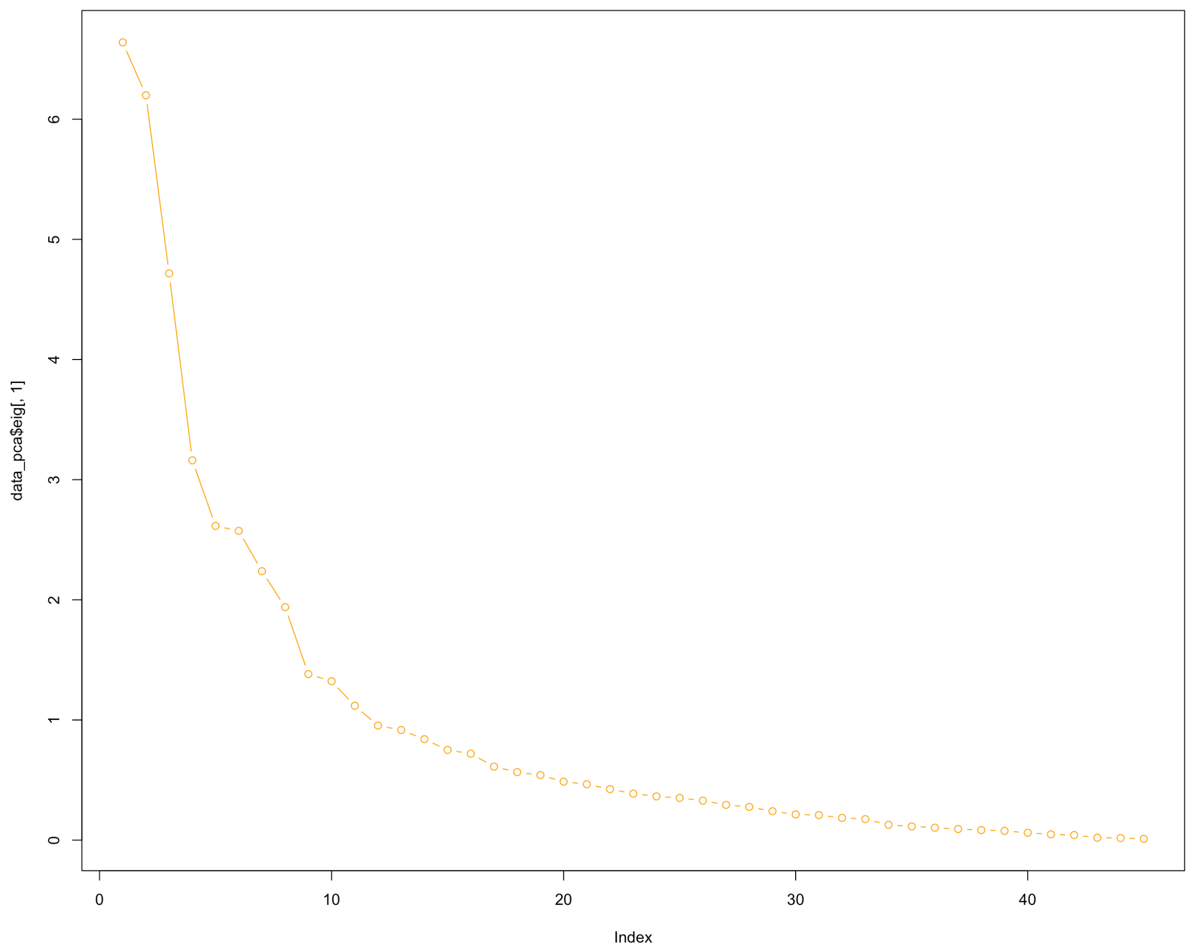
```
plot(data_pca$eig[,1],type='b', col='orange')
```

PART 1

PART 2

PART 3

References



As shown above the number of factors seems to be around 9-11.

- **Percentage of explained variance**

Hide

```
head(data_pca$eig, 13)
```

PART 1

PART 2

PART 3

References

```
##         eigenvalue percentage of variance
## comp 1   6.6396393              14.754754
## comp 2   6.1990901              13.775756
## comp 3   4.7163879              10.480862
## comp 4   3.1607857               7.023968
## comp 5   2.6150947               5.811322
## comp 6   2.5737687               5.719486
## comp 7   2.2383847               4.974188
## comp 8   1.9390767               4.309059
## comp 9   1.3816916               3.070426
## comp 10  1.3225830               2.939073
## comp 11  1.1179952               2.484434
## comp 12  0.9535928               2.119095
## comp 13  0.9161916               2.035981
##         cumulative percentage of variance
## comp 1                           14.75475
## comp 2                           28.53051
## comp 3                           39.01137
## comp 4                           46.03534
## comp 5                           51.84666
## comp 6                           57.56615
## comp 7                           62.54034
## comp 8                           66.84940
## comp 9                           69.91982
## comp 10                          72.85889
## comp 11                          75.34333
## comp 12                          77.46242
## comp 13                          79.49840
```

This method is really discretional and there aren't right or wrong decisions. However using **11 factors** the 75% of the total variance is explained, so we decide to work with 11 factors.

Let's create the new dataframe according to the new 11 variables.

Hide

PART 1
PART 2
PART 3
References

```
new_data_pca <- as.data.frame(data_pca$ind[[1]])

# Add the labels
new_data_pca <- cbind(new_data_pca, dailysport$id)

# Shift the id_column position
new_data_pca <- new_data_pca[,c(12,1:11)]
```

Now we split data in train and test:

Hide

```
n_test <- round(0.3*nrow(new_data_pca))

idx_test <- sort(sample(1:nrow(new_data_pca), n_test))

test_PCA <- new_data_pca[idx_test,]
train_PCA <- new_data_pca[setdiff(1:nrow(new_data_pca),idx_test),]
```

Prediction time....again

Let's start using the **Naive Bayes** method on the reduced (11 variables) dataset:

Hide

```
tr.pca.bayes <- naiveBayes(`dailysport$id`~., data = train_PCA)

tr.pca.prediction <- predict(tr.pca.bayes, train_PCA[,-1])

confusionMatrix(tr.pca.prediction, train_PCA$`dailysport$id`)
```

PART 1

PART 2

PART 3

References


```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    5236       0      20       0
##    jumping       4    5210       1       3
##    stepper      29       0    5096       2
##    walking       5       4     174    5216
##
## Overall Statistics
##
##                Accuracy : 0.9885
##                  95% CI : (0.9869, 0.9899)
##     No Information Rate : 0.252
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9846
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9928         0.9992         0.9631
## Specificity                  0.9987         0.9995         0.9980
## Pos Pred Value               0.9962         0.9985         0.9940
## Neg Pred Value               0.9976         0.9997         0.9877
## Prevalence                   0.2511         0.2483         0.2520
## Detection Rate               0.2493         0.2481         0.2427
## Detection Prevalence         0.2503         0.2485         0.2441
## Balanced Accuracy            0.9958         0.9994         0.9806
##                      Class: walking
## Sensitivity                  0.9990
## Specificity                  0.9884
## Pos Pred Value               0.9661
## Neg Pred Value               0.9997
## Prevalence                   0.2486
## Detection Rate               0.2484
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence         0.2571
## Balanced Accuracy            0.9937
```

And now test the model:

Hide

```
te.pca.prediction <- predict(tr.pca.bayes, test_PCA[,-1])

confusionMatrix(te.pca.prediction, test_PCA$`dailysport$id`)
```

PART 1
PART 2
PART 3
References

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    2214       0      15       0
##    jumping       1    2283       0       2
##    stepper      10       0    2130       0
##    walking       1       3      64    2277
##
## Overall Statistics
##
##                Accuracy : 0.9893
##                  95% CI : (0.987, 0.9914)
##     No Information Rate : 0.254
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9858
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9946         0.9987         0.9642
## Specificity                  0.9978         0.9996         0.9985
## Pos Pred Value               0.9933         0.9987         0.9953
## Neg Pred Value               0.9982         0.9996         0.9885
## Prevalence                   0.2473         0.2540         0.2454
## Detection Rate               0.2460         0.2537         0.2367
## Detection Prevalence         0.2477         0.2540         0.2378
## Balanced Accuracy            0.9962         0.9991         0.9814
##                      Class: walking
## Sensitivity                  0.9991
## Specificity                  0.9899
## Pos Pred Value               0.9710
## Neg Pred Value               0.9997
## Prevalence                   0.2532
## Detection Rate               0.2530
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence          0.2606
## Balanced Accuracy             0.9945
```

Analyzing the confusion matrix, we can see that the accuracy is a bit smaller than the one obtained in the whole dataset, but the advantage here is that we have used only 11 variables.

In the next chunk we'll try our algorithm:

Hide

```
our_pca <- naive_bayes2(train_PCA)

our_pca_pred <- predizione(our_pca, test_PCA)

# Create the factors from the output list, in order to apply the confu
sionMatrix function
levels_predizione <- as.factor(our_pca_pred)

confusionMatrix(levels_predizione, test_PCA$`dailysport$id`)
```

PART 1
PART 2
PART 3
References

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    2211       0      14       0
##    jumping       1    2286       0       0
##    stepper       6       0    2144       0
##    walking       8       0      51    2279
##
## Overall Statistics
##
##                Accuracy : 0.9911
##                  95% CI : (0.9889, 0.9929)
##     No Information Rate : 0.254
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9881
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9933         1.0000         0.9706
## Specificity                  0.9979         0.9999         0.9991
## Pos Pred Value               0.9937         0.9996         0.9972
## Neg Pred Value               0.9978         1.0000         0.9905
## Prevalence                   0.2473         0.2540         0.2454
## Detection Rate               0.2457         0.2540         0.2382
## Detection Prevalence         0.2472         0.2541         0.2389
## Balanced Accuracy            0.9956         0.9999         0.9848
##                      Class: walking
## Sensitivity                  1.0000
## Specificity                  0.9912
## Pos Pred Value               0.9748
## Neg Pred Value               1.0000
## Prevalence                   0.2532
## Detection Rate               0.2532
```

PART 1

PART 2

PART 3

References

```
## Detection Prevalence          0.2598
## Balanced Accuracy             0.9956
```

Ok, our classifier seems to be great!

However we have to keep in mind that we're working with the informations which come from one specific person so the activies are performed always in the same way.

What happens if we improve our dataset adding data from another person?

To obtain this data we explore the *UCI Machine Learning Repository* (https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities) and we download the data folder. It would be useless let you change the path to make the code runnable, so we create the new dataset and we store it in our *Github* repository.

The next chunk of code is not going to be run, but we think it could be usefull to understand how we create the new dataset:

Hide

PART 1
PART 2
PART 3
References

```
# Initialize an empty dataframe
new_person <- data.frame()

# Iterate over the four activities
for (activity in c(9,13,14,18))
{
  if (activity==9){
    activitaa = paste("a0",activity,sep = "")
  }
  else{
    activitaa = paste("a",activity,sep = "")
  }

  # Iterate over all the sensors
  for (senso in 1:60)
  {
    if (senso %in% 1:9){
      aux=paste("s0",senso,sep = "")
    }
    else{
      aux=paste("s",senso,sep = "")

    }

    # Create the full path for each combination
    path = paste("/Users/andreamarcocchia/Desktop/Statistical learnin
g/HW2/data/",activitaa,"/p5/",aux,".txt", sep="")

    # Read the file
    s01 <- read_csv(path, col_names = FALSE)

    # Identify the correct label
    if (activity == 9)
    {
      labee <- "walking"
    }
```

PART 1
PART 2
PART 3
References

```
    else if (activity == 13)
    {
      labee <- "stepper"
    }

    else if (activity == 14)
    {
      labee <- "crosstr"
    }

    else if (activity == 18)
    {
      labee <- "jumping"
    }

    # Add the label to the dataframe
    s01 <- cbind(s01,labee)

    # Add observations to the complete dataset
    new_person <- rbind(new_person, s01)
  }
}

# Switch the column positions such that the id is the first one
new_person <- new_person[,c(ncol(new_person),1:(ncol(new_person)-1))]

# Add variable names to the new dataframe
names(new_person) <- names(dailysport)


# Write a .txt file that will be upload to our Github
write.table(new_person,"/Users/andreamarcocchia/Desktop/new_person.tx
t",sep="\t",row.names=FALSE)
```

Ok, end of *non-running* section of our code!

In the next chunk we download the data from Github:

PART 1
PART 2
PART 3
References

Hide

```
# Load data from an URL
new_person <-read.csv(text=getURL("https://raw.githubusercontent.com/a
ndremarco/Naive-bayes-classifier/master/new_person.txt"), header=T, se
p = "\t")
```

Let's **merge the two dataframes**: we would like to have a big dataframe with 60000 observations that are related to two different persons.

Hide

```
# Modify the dataframe names
names(new_person) <- names(dailysport)

# Concatenate the two daframes
gigante <- rbind(dailysport,new_person)
```

Since now the two persons are combined as they are a single person who is observed for 60000 times in different activities. So now we create the train and test dataset.

Hide

```
# Select the number of elements in the test (30%)
n_test <- round(0.3*nrow(gigante))

idx_test <- sort(sample(1:nrow(gigante), n_test))

test_gigante <- gigante[idx_test,]
train_gigante <- gigante[setdiff(1:nrow(gigante),idx_test),]
```

And now…prediction!

Let's start using the *naiveBayes* implementation of *e1071* package.

Hide

PART 1
PART 2
PART 3
References

```
pred_new_person <- naiveBayes(id~., data = train_gigante)

prediz_np <- predict(pred_new_person, test_gigante[,-1])

confusionMatrix(prediz_np, test_gigante$id)
```

PART 1

PART 2

PART 3

References


```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    4470       0       1       0
##    jumping       4    4442      15       0
##    stepper       1       0    4595       0
##    walking       0       3       3    4466
##
## Overall Statistics
##
##                Accuracy : 0.9985
##                  95% CI : (0.9978, 0.999)
##     No Information Rate : 0.2563
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.998
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9989         0.9993         0.9959
## Specificity                  0.9999         0.9986         0.9999
## Pos Pred Value               0.9998         0.9957         0.9998
## Neg Pred Value               0.9996         0.9998         0.9986
## Prevalence                   0.2486         0.2469         0.2563
## Detection Rate               0.2483         0.2468         0.2553
## Detection Prevalence         0.2484         0.2478         0.2553
## Balanced Accuracy            0.9994         0.9990         0.9979
##                      Class: walking
## Sensitivity                  1.0000
## Specificity                  0.9996
## Pos Pred Value               0.9987
## Neg Pred Value               1.0000
## Prevalence                   0.2481
## Detection Rate               0.2481
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence         0.2484
## Balanced Accuracy            0.9998
```

Very good job also in this case, in fact the accuracy is near 100%.

Now it's important to understand the role of the data used to train the model: we have tried to train the *Naive Bayes* classifier on the data of one person, and test it on the data of the other one:

Hide

```
train_p1 <- naiveBayes(id~., data = dailysport)

pred_p2 <- predict(train_p1, new_person[,-1])

confusionMatrix(pred_p2, new_person$id)
```

PART 1
PART 2
PART 3
References

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    7500      37     874       0
##    jumping       0       0       0       0
##    stepper       0    4949      90       0
##    walking       0    2514    6536    7500
##
## Overall Statistics
##
##                Accuracy : 0.503
##                  95% CI : (0.4973, 0.5087)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3373
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  1.0000           0.00        0.01200
## Specificity                  0.9595           1.00        0.78004
## Pos Pred Value               0.8917            NaN        0.01786
## Neg Pred Value               1.0000           0.75        0.70314
## Prevalence                   0.2500           0.25        0.25000
## Detection Rate               0.2500           0.00        0.00300
## Detection Prevalence         0.2804           0.00        0.16797
## Balanced Accuracy            0.9798           0.50        0.39602
##                      Class: walking
## Sensitivity                  1.0000
## Specificity                  0.5978
## Pos Pred Value               0.4532
## Neg Pred Value               1.0000
## Prevalence                   0.2500
## Detection Rate               0.2500
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence          0.5517
## Balanced Accuracy             0.7989
```

As it’s possible to see the accuracy has decreased of about 50 percentage points.

It’s very important that the train and test datasets describe the features of the analyzed person. The train and test have to be balanced because the data through which the model is trained, will determin the quality of the prediction (https://eu.usatoday.com/story/tech/nation-now/2018/06/07/artificial-intelligence-fed-reddit-captions-became-psychopath/681888002/).

Decision Tree

Up to now we didn’t focus on the importance of the single original variables. So we would like to find out the most important variables, in terms of given informations. We do it through the **decision trees**, using the *rpart* package.

Hide

```
# Fit the model
daily_rpart = rpart(id ~ ., data = daily_train, control = rpart.contro
l(xval = 100))
daily_rpart
```

PART 1
PART 2
PART 3
References

```
## n= 21000
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 21000 15738 walking (0.2493333333 0.2499523810 0.250142857
1 0.2505714286)
##    2) LL-zMag< -0.28397 5249     0 jumping (0.0000000000 1.00000000
00 0.0000000000 0.0000000000) *
##    3) LL-zMag>=-0.28397 15751 10489 walking (0.3324233382 0.0000000
000 0.3335026348 0.3340740270)
##      6) RL-zMag>=-0.316885 10489  5246 walking (0.4991896272 0.0000
000000 0.0009533797 0.4998569930)
##       12) LA-xMag>=-0.20249 5239     3 crosstr (0.9994273716 0.0000
000000 0.0005726284 0.0000000000) *
##       13) LA-xMag< -0.20249 5250     7 walking (0.0000000000 0.0000
000000 0.0013333333 0.9986666667) *
##      7) RL-zMag< -0.316885 5262    19 stepper (0.0000000000 0.00000
00000 0.9963892056 0.0036107944) *
```

Hide

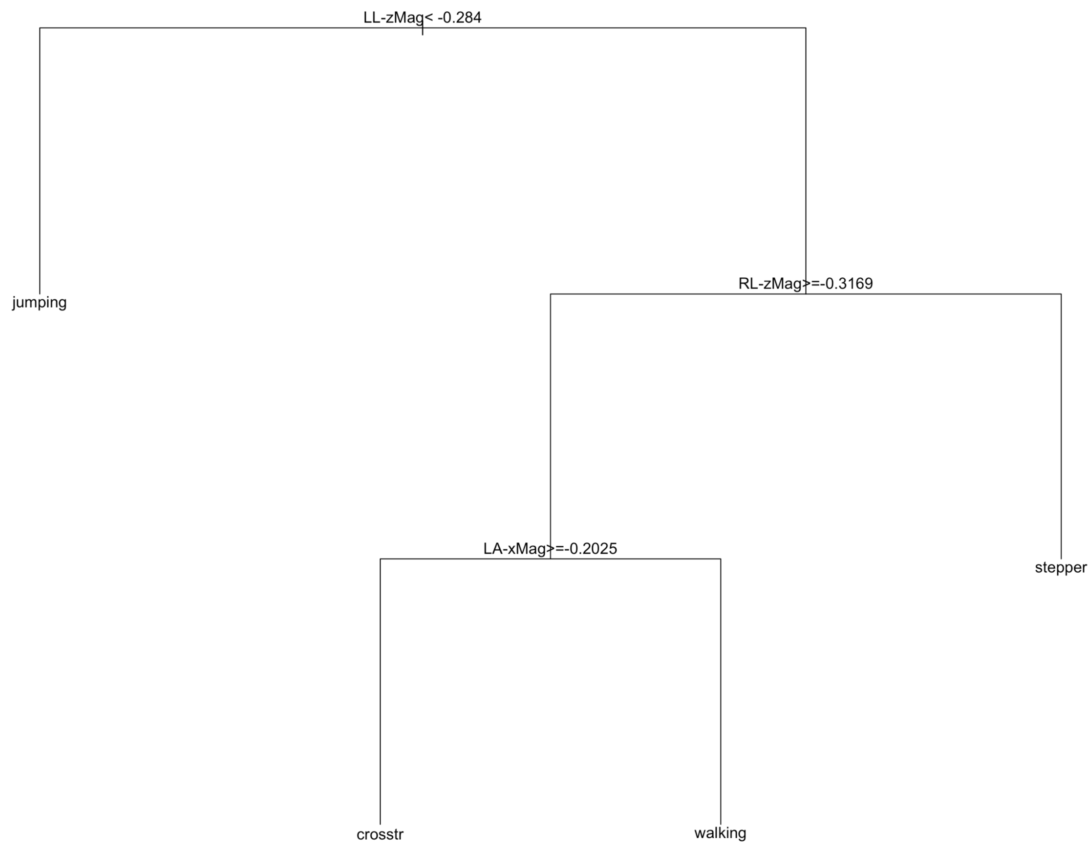
```
# Plot the decision tree
plot(daily_rpart)
text(daily_rpart)
```

PART 1

PART 2

PART 3

References



Ok, seems to have found the most important variables:

- **T-yMag**
- **RL-zMag**
- **LA-xMag**

It means that the magnetometer is the instrument that is more able to capture the diffences among the four activities.

PART 1
PART 2
PART 3
References

So we can classify using only them. Our hope is to obtain the same accuracy (or a bit smaller) as the one obtained using all the variables

Let's start working with three variables. In the next chunk we create a new dataframe:

Hide

```
test_ridotto <-  as.data.frame(daily_test[,c("id","T-yMag","RL-zMag",
"LA-xMag")])
train_ridotto <- as.data.frame(daily_train[,c("id","T-yMag","RL-zMag",
"LA-xMag")])
```

Let's check the results, using *Naive Bayes*:

Hide

```
t1 <- naiveBayes(id~., data = train_ridotto)

p1 <- predict(t1, test_ridotto[,-1])

confusionMatrix(p1, test_ridotto$id)
```

PART 1
PART 2
PART 3
References

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    2258       0       0       0
##    jumping       0    2251       0       0
##    stepper       0       0    2247       0
##    walking       6       0       0    2238
##
## Overall Statistics
##
##                Accuracy : 0.9993
##                  95% CI : (0.9985, 0.9998)
##     No Information Rate : 0.2516
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9991
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9973         1.0000         1.0000
## Specificity                  1.0000         1.0000         1.0000
## Pos Pred Value               1.0000         1.0000         1.0000
## Neg Pred Value               0.9991         1.0000         1.0000
## Prevalence                   0.2516         0.2501         0.2497
## Detection Rate               0.2509         0.2501         0.2497
## Detection Prevalence         0.2509         0.2501         0.2497
## Balanced Accuracy            0.9987         1.0000         1.0000
##                      Class: walking
## Sensitivity                  1.0000
## Specificity                  0.9991
## Pos Pred Value               0.9973
## Neg Pred Value               1.0000
## Prevalence                   0.2487
## Detection Rate               0.2487
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence         0.2493
## Balanced Accuracy            0.9996
```

Our hope has been respected: we reach 99.9% of the accuracy.

Now we want to reduce more the number of variables, so we pick up only the first two. We proceed as before:

Hide

```
test_ridotto2 <-  as.data.frame(daily_test[,c("id","T-yMag","RL-zMag"
)])
train_ridotto2 <- as.data.frame(daily_train[,c("id","T-yMag","RL-zMag"
)])
```

Now that we are working with two dimensions, it’s easier to see how the variables are distributed:

Hide

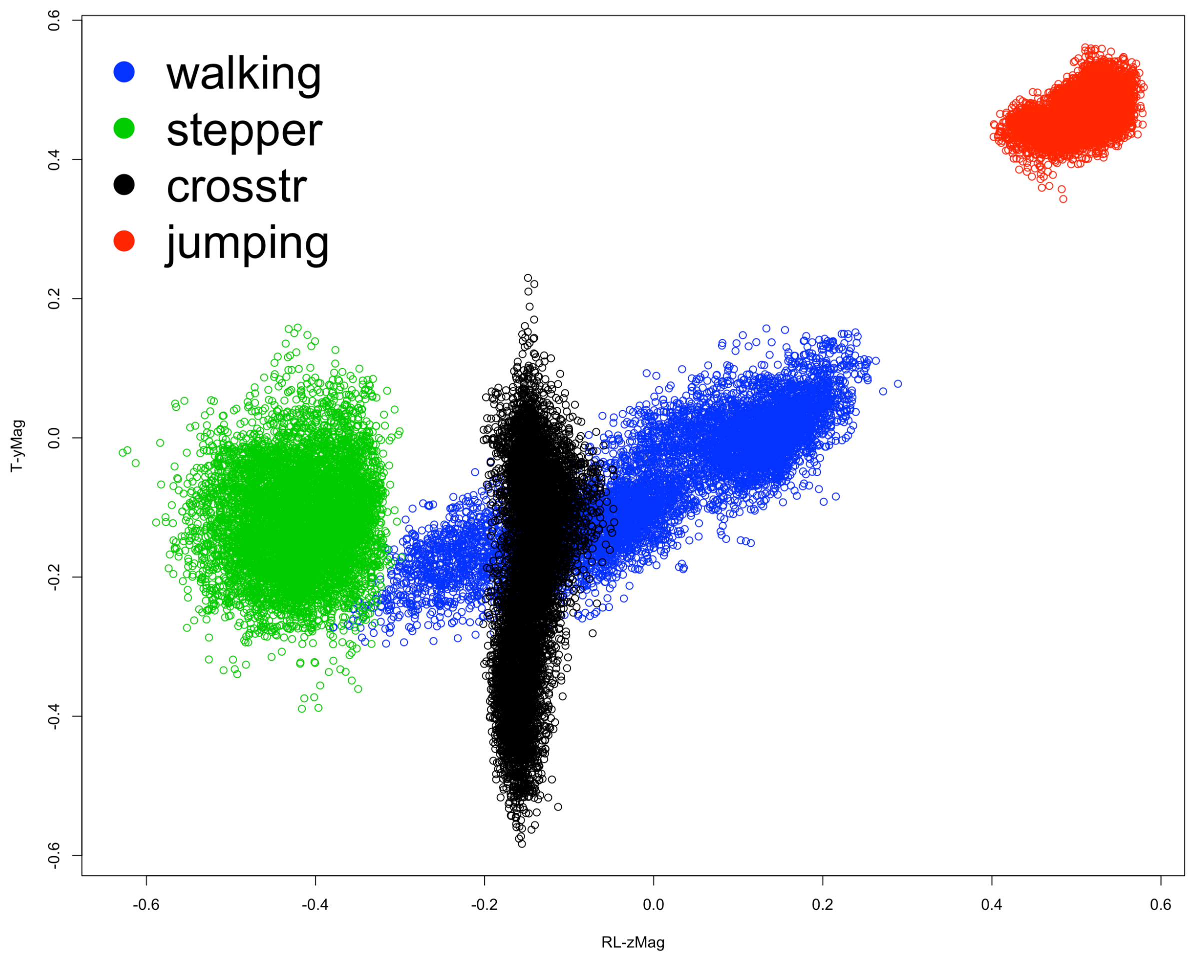
```
plot(dailysport$`RL-zMag`,dailysport$`T-yMag`, col=dailysport$id, xlab
 ="RL-zMag", ylab = "T-yMag")
legend(x="topleft", legend = unique(dailysport$id), col=unique(dailysp
ort$id), pch=16, cex=3, bty = 'n')
```

PART 1

PART 2

PART 3

References



Visualizing the plot we can expect that some errors can occur in the walking and cross training classification.

Let's see…and do!

Hide

PART 1
PART 2
PART 3
References

```
t2 <- naiveBayes(id~., data = train_ridotto2)

p2 <- predict(t2, test_ridotto2[,-1])

confusionMatrix(p2, test_ridotto2$id)
```

PART 1

PART 2

PART 3

References


```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction crosstr jumping stepper walking
##    crosstr    2141       0       0     296
##    jumping       0    2251       0       0
##    stepper       0       0    2245      68
##    walking     123       0       2    1874
##
## Overall Statistics
##
##                Accuracy : 0.9457
##                  95% CI : (0.9408, 0.9503)
##     No Information Rate : 0.2516
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9275
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: crosstr Class: jumping Class: stepper
## Sensitivity                  0.9457         1.0000         0.9991
## Specificity                  0.9561         1.0000         0.9899
## Pos Pred Value               0.8785         1.0000         0.9706
## Neg Pred Value               0.9813         1.0000         0.9997
## Prevalence                   0.2516         0.2501         0.2497
## Detection Rate               0.2379         0.2501         0.2494
## Detection Prevalence         0.2708         0.2501         0.2570
## Balanced Accuracy            0.9509         1.0000         0.9945
##                      Class: walking
## Sensitivity                  0.8374
## Specificity                  0.9815
## Pos Pred Value               0.9375
## Neg Pred Value               0.9480
## Prevalence                   0.2487
## Detection Rate               0.2082
```

PART 1
PART 2
PART 3
References

```
## Detection Prevalence          0.2221
## Balanced Accuracy             0.9094
```

The accuracy is a bit smaller than before, in fact we reached 94.5%.

Analyzing the confusion matrix results, it's possible to see that the errors, in accordance to the decision tree plot, occur during the classification between *crosstr* and *walking* classes.

Hide

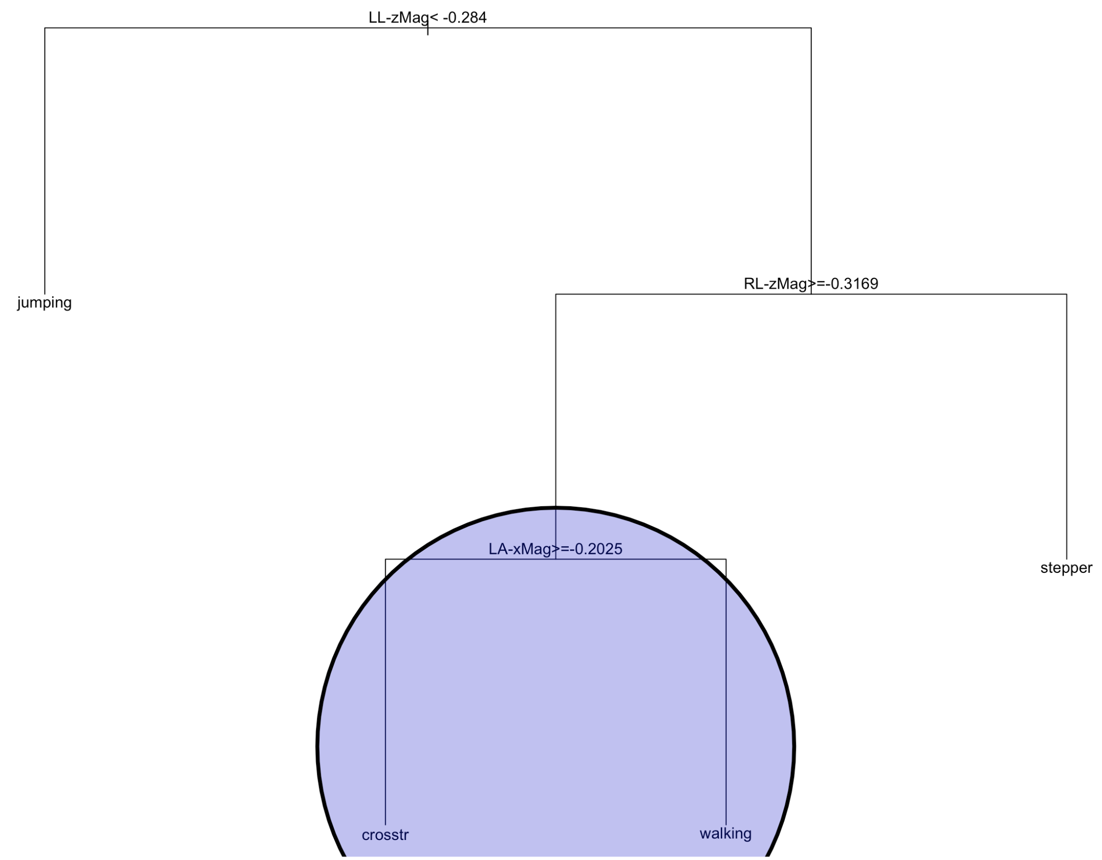
```
plot( daily_rpart )                    # Basic plot
text(daily_rpart)
draw.circle(2.5,0.1,0.7,nv=100,border=NULL,col=rgb(0,0,0.8,0.3),lty=1,
density=NULL,angle=45,lwd=4)
```

PART 1

PART 2

PART 3

References



Summary

Let’s make a final recap!

At the end of this project it’s really clear the importance of three things in **classification**:

- Assumptions about the variables’ distribution
- Variable selection
- Construction of test and train dataset

PART 1

PART 2

PART 3

References

About the **first point**, we have seen that the behaviour of our **non-parametric** *Naive Bayes* classifies better than the one built under gaussian assumption. So, generally, we can say that if you don't have certainty about the distribution of your data, the better choice is to handle the problem via a *non-parametric* approach.

Regarding to the **second point**, we have seen that is not necessary to use all the variables to obtain a great classification. It's always recommended to extract the meaningfull variables or to reduce the features' space (via PCA or LDA) for avoiding the curse of dimensionality.

And finally, for the **third point** (the last but not the least), we have seen that building both a non-balanced train and test bring us into a mess. It's important to train the model with data related to the subject that has to be predicted.

References

- S.B. Kotsiantis, *Supervised Machine Learning: A review of classification techniques*
- A.Ashari,I.Paryudi, A.M.Tjoa, *Performance Comparison between Naive Bayes, Decision Tree and the K-Nearest Neighbor in Searching Alternative Design in an Energy Simulation Tool*
- H.Zhang, *The Optimality of Naive Bayes*
- P. Domingos, M. Pazzani, *On the Optimality of the Simple Bayesian Classifier under Zero-One Loss*
- G. H. John, P. Langley, *Estimating Continuous Distributions in Bayesian Classifiers*