Untitled

```
#setwd('C:/Users/DanEscario/Desktop/MESIO/Statistical learning/Pr?ctiques/Tree based method')

library(readr)
soldat <- read_csv("soldat.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   y = col_integer()
## )
```

```
## See spec(...) for full column specifications.
```

```
#View(soldat)

if(!require("caret")) install.packages("caret")
```

```
## Loading required package: caret
```

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

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

```
library(caret)
if(!require("knitr")) install.packages("knitr")
```

```
## Loading required package: knitr
```

```
library(knitr)
if(!require("doMC")) install.packages("doMC")
```

```
## Loading required package: doMC
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(doMC)
if(!require("parallel")) install.packages("parallel")
library(parallel)
#doMC::registerDoMC(parallel::detectCores())
if(!require("dplyr")) install.packages("dplyr")
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:ggplot2':
##
##     vars
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(dplyr)
if(!require("magrittr")) install.packages("magrittr")
```

```
## Loading required package: magrittr
```

```
library(magrittr)
if(!require("ggplot2")) install.packages("ggplot2")
library(ggplot2)
if(!require("scales")) install.packages("scales")
```

```
## Loading required package: scales
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(scales)
if(!require("reshape2")) install.packages("reshape2")
```

```
## Loading required package: reshape2
```

```
library(reshape2)
if(!require("randomForest")) install.packages("randomForest")
```

```
## Loading required package: randomForest
```

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

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(randomForest)
if(!require("gbm")) install.packages("gbm")
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loaded gbm 2.1.3
```

```
library(gbm)
```

```
data<-soldat
colSums(is.na(data))
```

```
##  x1  x2  x3  x4  x5  x6  x7  x8  x9 x10 x11 x12 x13 x14 x15 x16 x17 x18
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## x19 x20 x21 x22 x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34 x35 x36
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## x37 x38 x39 x40 x41 x42 x43 x44 x45 x46 x47 x48 x49 x50 x51 x52 x53 x54
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## x55 x56 x57 x58 x59 x60 x61 x62 x63 x64 x65 x66 x67 x68 x69 x70 x71 x72
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 787   0
##   y
##   0
```

```
data<-data[,-71]
n<-nrow(data)
p<-ncol(data)
data$y<-as.factor(data$y)
levels(data$y)[levels(data$y)=="-1"] <- "Insoluble"
levels(data$y)[levels(data$y)=="1"] <- "Soluble"
table(data$y)
```

```
##
## Insoluble   Soluble
##      3493      2138
```

1.Do a short exploratory analysis in order to know some characteristics of each variable

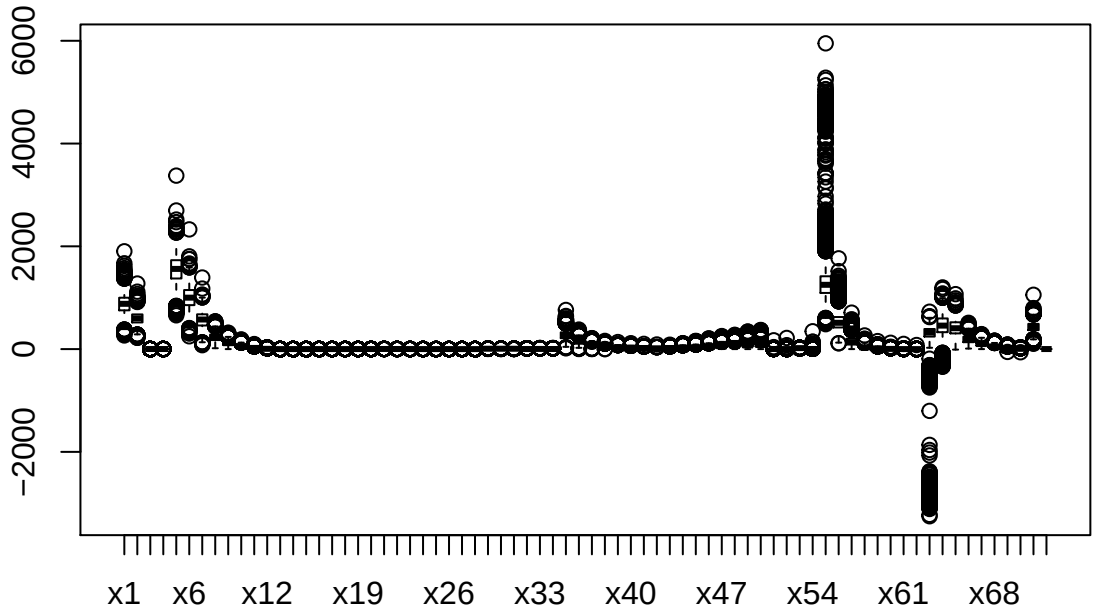
```
summary(data)
```

```
##        x1              x2              x3             x4
##  Min.   : 265.5   Min.   : 219.6   Min.   :1.208   Min.   :1.069
##  1st Qu.: 754.2   1st Qu.: 518.9   1st Qu.:1.447   1st Qu.:1.465
##  Median : 880.9   Median : 598.0   Median :1.469   Median :1.561
##  Mean   : 880.6   Mean   : 596.4   Mean   :1.472   Mean   :1.551
##  3rd Qu.:1000.1   3rd Qu.: 674.9   3rd Qu.:1.493   3rd Qu.:1.646
##  Max.   :1903.1   Max.   :1275.7   Max.   :1.703   Max.   :2.251
##        x5              x6              x7              x8
##  Min.   : 657.5   Min.   : 248.9   Min.   :  76.0   Min.   : 17.0
##  1st Qu.:1374.1   1st Qu.: 855.4   1st Qu.: 459.1   1st Qu.:175.8
##  Median :1560.4   Median :1000.9   Median : 575.8   Median :229.9
##  Mean   :1543.7   Mean   : 998.5   Mean   : 570.1   Mean   :230.9
##  3rd Qu.:1729.9   3rd Qu.:1149.1   3rd Qu.: 677.1   3rd Qu.:282.1
##  Max.   :3375.8   Max.   :2329.4   Max.   :1392.8   Max.   :541.8
##        x9             x10             x11             x12
##  Min.   :  0.00   Min.   :  0.00   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 84.12   1st Qu.: 41.50   1st Qu.:15.25   1st Qu.: 2.50
##  Median :115.00   Median : 58.75   Median :22.50   Median : 5.00
##  Mean   :116.37   Mean   : 59.70   Mean   :24.23   Mean   : 6.27
##  3rd Qu.:145.81   3rd Qu.: 75.38   3rd Qu.:30.56   3rd Qu.: 8.25
##  Max.   :318.38   Max.   :188.88   Max.   :98.00   Max.   :36.00
##       x13              x14             x15             x16
##  Min.   :0.00100   Min.   :0.0010   Min.   :0.0020   Min.   :0.0040
```

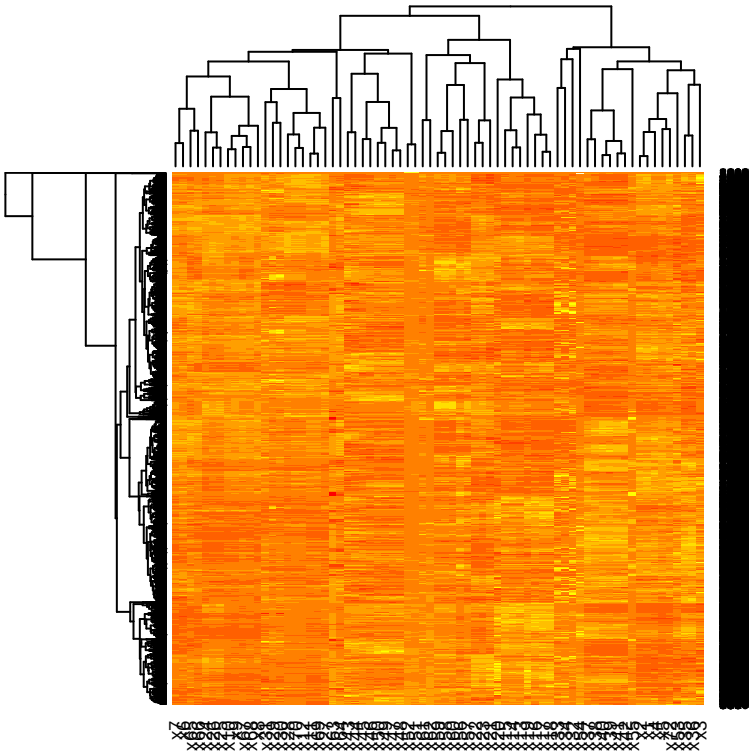
```
##  1st Qu.:0.03400   1st Qu.:0.0600   1st Qu.:0.0980   1st Qu.:0.1850
##  Median :0.05200   Median :0.0910   Median :0.1500   Median :0.2830
##  Mean   :0.06219   Mean   :0.1059   Mean   :0.1765   Mean   :0.3303
##  3rd Qu.:0.08100   3rd Qu.:0.1350   3rd Qu.:0.2250   3rd Qu.:0.4310
##  Max.   :0.28600   Max.   :0.4800   Max.   :0.8800   Max.   :1.4670
##       x17              x18              x19              x20
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2720   1st Qu.:0.3480   1st Qu.:0.4530   1st Qu.:0.5660
##  Median :0.4140   Median :0.5210   Median :0.6860   Median :0.8800
##  Mean   :0.4777   Mean   :0.5951   Mean   :0.7531   Mean   :0.9349
##  3rd Qu.:0.6325   3rd Qu.:0.7890   3rd Qu.:0.9850   3rd Qu.:1.2180
##  Max.   :1.9010   Max.   :2.2390   Max.   :2.5560   Max.   :3.2330
##       x21              x22              x23              x24
##  Min.   :1.860   Min.   :0.695   Min.   :0.1760   Min.   :0.0270
##  1st Qu.:2.430   1st Qu.:1.452   1st Qu.:0.7870   1st Qu.:0.3060
##  Median :2.610   Median :1.697   Median :0.9680   Median :0.3850
##  Mean   :2.611   Mean   :1.699   Mean   :0.9705   Mean   :0.3927
##  3rd Qu.:2.777   3rd Qu.:1.927   3rd Qu.:1.1435   3rd Qu.:0.4710
##  Max.   :4.060   Max.   :3.239   Max.   :2.1090   Max.   :1.1060
##       x25              x26              x27               x28
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.1470   1st Qu.:0.0730   1st Qu.:0.02800   1st Qu.:0.00500
##  Median :0.1910   Median :0.0980   Median :0.03700   Median :0.00800
##  Mean   :0.1977   Mean   :0.1011   Mean   :0.04068   Mean   :0.01036
##  3rd Qu.:0.2430   3rd Qu.:0.1240   3rd Qu.:0.05000   3rd Qu.:0.01300
##  Max.   :0.6660   Max.   :0.4010   Max.   :0.21100   Max.   :0.08500
##       x29              x30              x31              x32
##  Min.   : 2.678   Min.   : 1.931   Min.   : 1.543   Min.   : 0.707
##  1st Qu.: 6.819   1st Qu.: 6.521   1st Qu.: 6.246   1st Qu.: 0.866
##  Median : 7.219   Median : 6.882   Median : 6.569   Median : 1.500
##  Mean   : 7.527   Mean   : 7.052   Mean   : 6.650   Mean   : 3.210
##  3rd Qu.: 8.033   3rd Qu.: 7.387   3rd Qu.: 6.968   3rd Qu.: 4.717
##  Max.   :12.946   Max.   :11.980   Max.   :11.319   Max.   :17.292
##       x33              x34              x35              x36
##  Min.   : 0.707   Min.   : 0.707   Min.   : 15.62   Min.   :  1.875
##  1st Qu.: 1.500   1st Qu.: 1.118   1st Qu.:212.50   1st Qu.:115.375
##  Median : 3.162   Median : 2.121   Median :265.25   Median :144.000
##  Mean   : 4.548   Mean   : 3.855   Mean   :270.73   Mean   :146.687
##  3rd Qu.: 7.159   3rd Qu.: 6.164   3rd Qu.:326.25   3rd Qu.:176.250
##  Max.   :21.059   Max.   :21.651   Max.   :761.62   Max.   :379.500
##       x37               x38              x39               x40
##  Min.   :  1.125   Min.   :  0.375   Min.   :  0.125   Min.   :  0.00
##  1st Qu.: 64.500   1st Qu.: 40.625   1st Qu.: 25.750   1st Qu.: 19.38
##  Median : 83.250   Median : 54.000   Median : 36.250   Median : 28.12
##  Mean   : 83.455   Mean   : 54.082   Mean   : 36.732   Mean   : 28.59
##  3rd Qu.:101.125   3rd Qu.: 67.375   3rd Qu.: 47.125   3rd Qu.: 37.00
##  Max.   :212.875   Max.   :159.375   Max.   :136.625   Max.   :114.50
##       x41              x42              x43              x44
##  Min.   :  0.00   Min.   : 0.000   Min.   : 0.454   Min.   :  0.616
##  1st Qu.: 13.62   1st Qu.: 9.875   1st Qu.:13.755   1st Qu.: 20.369
##  Median : 20.88   Median :15.500   Median :21.045   Median : 29.984
##  Mean   : 20.96   Mean   :15.468   Mean   :22.580   Mean   : 32.402
##  3rd Qu.: 27.50   3rd Qu.:20.625   3rd Qu.:29.450   3rd Qu.: 41.964
##  Max.   :101.50   Max.   :89.625   Max.   :85.059   Max.   :115.804
```

```
##       x45               x46              x47               x48
##  Min.   :  0.543   Min.   :  0.91   Min.   :  0.751   Min.   :  0.00
##  1st Qu.: 25.117   1st Qu.: 30.68   1st Qu.: 37.080   1st Qu.: 42.18
##  Median : 35.880   Median : 44.48   Median : 54.345   Median : 61.22
##  Mean   : 39.327   Mean   : 49.33   Mean   : 60.700   Mean   : 68.64
##  3rd Qu.: 50.449   3rd Qu.: 64.01   3rd Qu.: 78.532   3rd Qu.: 88.62
##  Max.   :159.264   Max.   :210.14   Max.   :253.770   Max.   :275.60
##       x49              x50              x51               x52
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.0000   Min.   :  0.000
##  1st Qu.: 47.45   1st Qu.: 53.99   1st Qu.:  0.9605   1st Qu.:  0.737
##  Median : 70.16   Median : 79.66   Median :  1.4160   Median :  1.089
##  Mean   : 78.58   Mean   : 89.11   Mean   :  1.6947   Mean   :  1.416
##  3rd Qu.:101.12   3rd Qu.:114.34   3rd Qu.:  1.9990   3rd Qu.:  1.639
##  Max.   :341.64   Max.   :371.88   Max.   :171.1110   Max.   :216.500
##       x53              x54              x55              x56
##  Min.   :11.61   Min.   :  0.007   Min.   : 484.6   Min.   : 113.6
##  1st Qu.:17.37   1st Qu.:  1.139   1st Qu.:1092.6   1st Qu.: 420.5
##  Median :18.81   Median :  2.172   Median :1250.9   Median : 511.5
##  Mean   :18.84   Mean   :  3.372   Mean   :1358.5   Mean   : 541.4
##  3rd Qu.:20.25   3rd Qu.:  3.720   3rd Qu.:1417.2   3rd Qu.: 624.0
##  Max.   :27.25   Max.   :346.750   Max.   :5949.9   Max.   :1766.2
##       x57              x58              x59              x60
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.000   Min.   :  0.000
##  1st Qu.: 90.38   1st Qu.: 15.62   1st Qu.:  5.875   1st Qu.:  1.625
##  Median :135.00   Median : 31.00   Median : 11.500   Median :  3.250
##  Mean   :156.38   Mean   : 41.60   Mean   : 16.799   Mean   :  5.492
##  3rd Qu.:202.75   3rd Qu.: 59.12   3rd Qu.: 24.500   3rd Qu.:  8.000
##  Max.   :709.12   Max.   :265.50   Max.   :155.250   Max.   :122.750
##       x61               x62               x63               x64
##  Min.   :  0.0000   Min.   : 0.0000   Min.   :-3252.8   Min.   :-344.9
##  1st Qu.:  0.0000   1st Qu.: 0.0000   1st Qu.:  243.7   1st Qu.: 328.6
##  Median :  0.2500   Median : 0.0000   Median :  322.1   Median : 457.0
##  Mean   :  0.8435   Mean   : 0.1091   Mean   :  185.2   Mean   : 457.2
##  3rd Qu.:  1.1250   3rd Qu.: 0.0000   3rd Qu.:  391.1   3rd Qu.: 596.5
##  Max.   :100.8750   Max.   :79.3750   Max.   :  727.6   Max.   :1200.4
##       x65              x66              x67              x68
##  Min.   : -10.25   Min.   : 10.38   Min.   :  0.00   Min.   :-10.75
##  1st Qu.: 305.31   1st Qu.:134.00   1st Qu.: 67.75   1st Qu.: 36.50
##  Median : 406.00   Median :185.88   Median : 98.00   Median : 53.00
##  Mean   : 413.76   Mean   :189.30   Mean   : 99.57   Mean   : 54.21
##  3rd Qu.: 521.94   3rd Qu.:239.94   3rd Qu.:128.50   3rd Qu.: 69.62
##  Max.   :1069.50   Max.   :511.12   Max.   :291.62   Max.   :161.88
##       x69              x70              x72                y
##  Min.   :-55.38   Min.   :-61.375   Min.   : 103.1   Insoluble:3493
##  1st Qu.: 14.62   1st Qu.:  2.500   1st Qu.: 369.2   Soluble  :2138
##  Median : 21.75   Median :  4.875   Median : 431.5
##  Mean   : 23.39   Mean   :  6.161   Mean   : 427.6
##  3rd Qu.: 29.62   3rd Qu.:  8.125   3rd Qu.: 490.5
##  Max.   : 90.38   Max.   : 34.875   Max.   :1056.7
```

```
boxplot(data)
```




```
heatmap(as.matrix(scale(data[,!names(data)%in%c('y')])))
```

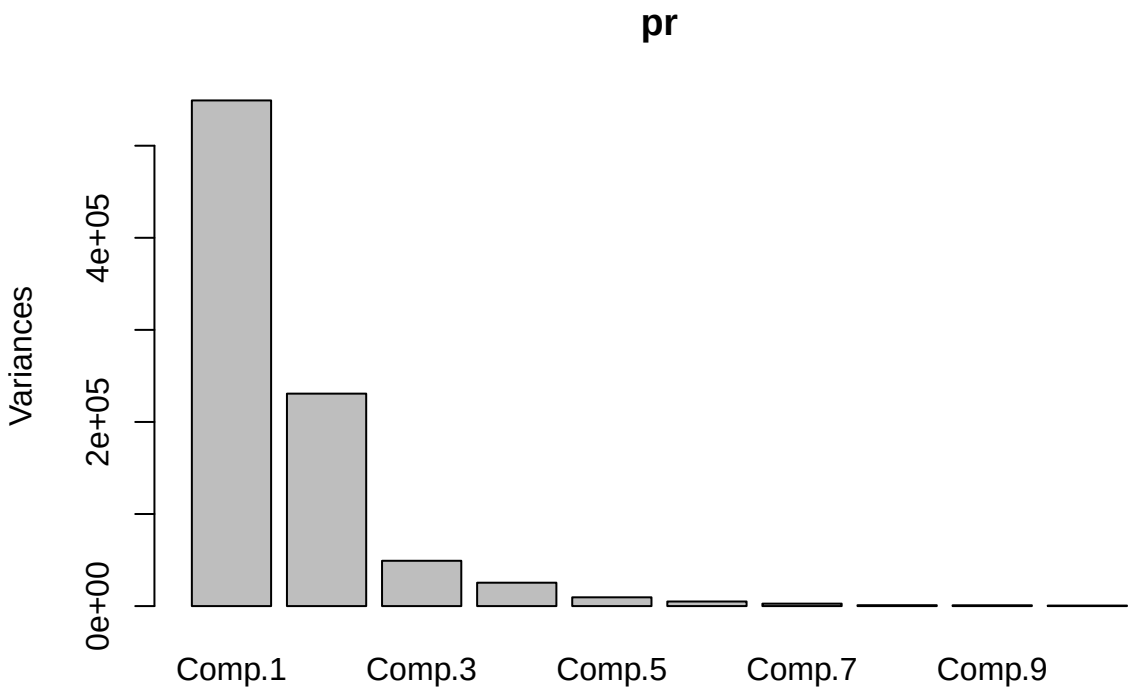


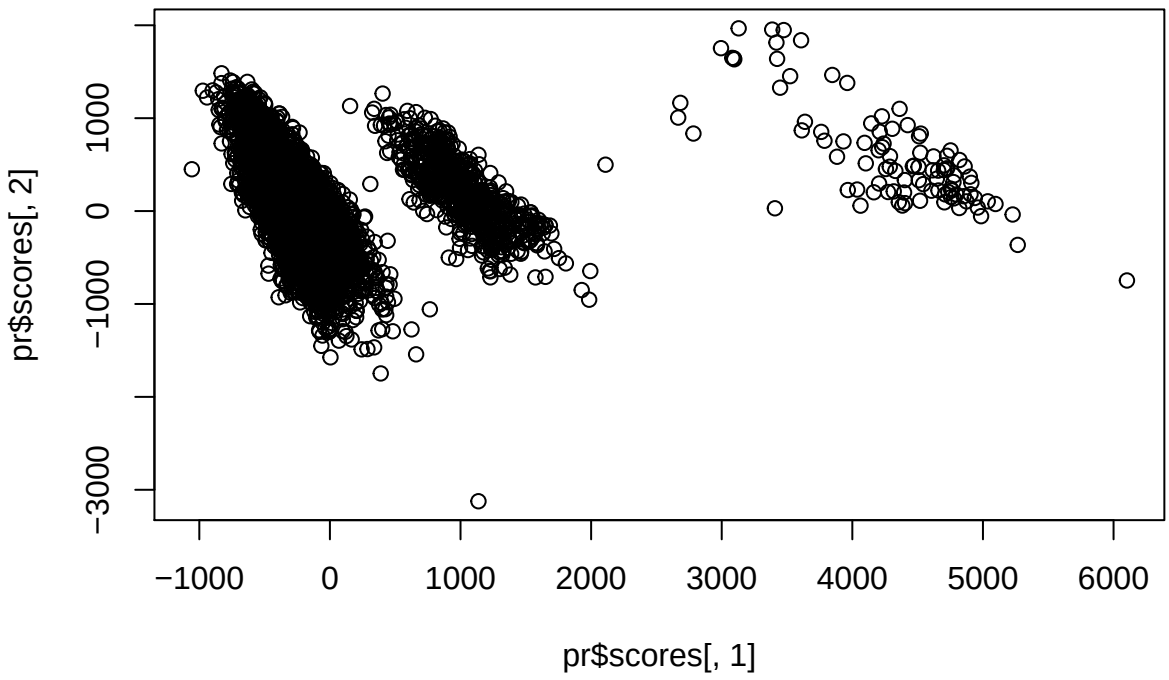
```
pr <- princomp(data[,!names(data)%in%c('y')])
summary(pr)
```

```
## Importance of components:
##                             Comp.1      Comp.2       Comp.3       Comp.4
## Standard deviation     741.061294 480.3445454 221.81446552 159.34860736
## Proportion of Variance   0.626934   0.2634021   0.05616856   0.02898746
## Cumulative Proportion    0.626934   0.8903361   0.94650467   0.97549213
##                             Comp.5       Comp.6       Comp.7       Comp.8
## Standard deviation     97.64611718 70.635126430 51.727860075 30.831780043
## Proportion of Variance  0.01088488  0.005695804  0.003054658  0.001085202
## Cumulative Proportion   0.98637700  0.992072808  0.995127466  0.996212668
##                              Comp.9      Comp.10      Comp.11      Comp.12
## Standard deviation     2.957349e+01 2.362351e+01 2.228370e+01 2.042334e+01
## Proportion of Variance 9.984324e-04 6.370922e-04 5.668762e-04 4.761756e-04
## Cumulative Proportion  9.972111e-01 9.978482e-01 9.984151e-01 9.988912e-01
##                             Comp.13      Comp.14      Comp.15      Comp.16
## Standard deviation     1.371173e+01 1.269381e+01 1.125975e+01 9.4281774203
## Proportion of Variance 2.146339e-04 1.839491e-04 1.447342e-04 0.0001014773
## Cumulative Proportion  9.991059e-01 9.992898e-01 9.994346e-01 0.9995360390
##                             Comp.17      Comp.18      Comp.19      Comp.20
## Standard deviation     8.275452e+00 7.955882e+00 7.251283e+00 6.273570e+00
## Proportion of Variance 7.818024e-05 7.225871e-05 6.002654e-05 4.493069e-05
## Cumulative Proportion  9.996142e-01 9.996865e-01 9.997465e-01 9.997914e-01
##                             Comp.21      Comp.22     Comp.23      Comp.24
## Standard deviation     5.5833442105 5.198196e+00 4.732140224 4.242132e+00
```

```
## Proportion of Variance 0.0000355879 3.084742e-05 0.000025564 2.054385e-05
## Cumulative Proportion  0.9998270231 9.998579e-01 0.999883434 9.999040e-01
##                            Comp.25      Comp.26      Comp.27      Comp.28
## Standard deviation     3.868152e+00 3.331671e+00 3.211863e+00 2.754085e+00
## Proportion of Variance 1.708129e-05 1.267179e-05 1.177681e-05 8.659009e-06
## Cumulative Proportion  9.999211e-01 9.999337e-01 9.999455e-01 9.999542e-01
##                            Comp.29      Comp.30      Comp.31      Comp.32
## Standard deviation     2.635823e+00 2.502377e+00 2.351962e+00 2.262760e+00
## Proportion of Variance 7.931329e-06 7.148569e-06 6.315009e-06 5.845079e-06
## Cumulative Proportion  9.999621e-01 9.999692e-01 9.999756e-01 9.999814e-01
##                            Comp.33      Comp.34      Comp.35      Comp.36
## Standard deviation     2.039101e+00 1.729637e+00 1.468154e+00 1.359197e+00
## Proportion of Variance 4.746693e-06 3.415257e-06 2.460688e-06 2.109010e-06
## Cumulative Proportion  9.999862e-01 9.999896e-01 9.999920e-01 9.999941e-01
##                            Comp.37      Comp.38      Comp.39      Comp.40
## Standard deviation     1.307280e+00 9.261104e-01 8.300574e-01 7.168806e-01
## Proportion of Variance 1.950970e-06 9.791272e-07 7.865563e-07 5.866879e-07
## Cumulative Proportion  9.999961e-01 9.999971e-01 9.999979e-01 9.999984e-01
##                            Comp.41      Comp.42      Comp.43      Comp.44
## Standard deviation     6.613629e-01 4.851539e-01 4.445672e-01 4.070666e-01
## Proportion of Variance 4.993364e-07 2.687031e-07 2.256256e-07 1.891667e-07
## Cumulative Proportion  9.999989e-01 9.999992e-01 9.999994e-01 9.999996e-01
##                            Comp.45      Comp.46      Comp.47      Comp.48
## Standard deviation     3.909313e-01 2.796269e-01 2.646176e-01 1.116993e-01
## Proportion of Variance 1.744675e-07 8.926302e-08 7.993756e-08 1.424344e-08
## Cumulative Proportion  9.999998e-01 9.999999e-01 1.000000e+00 1.000000e+00
##                            Comp.49      Comp.50      Comp.51      Comp.52
## Standard deviation     9.760332e-02 5.818657e-02 2.854722e-02 2.253795e-02
## Proportion of Variance 1.087534e-08 3.865085e-09 9.303390e-10 5.798858e-10
## Cumulative Proportion  1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##                            Comp.53      Comp.54      Comp.55      Comp.56
## Standard deviation     1.835594e-02 1.095329e-02 9.108041e-03 8.215978e-03
## Proportion of Variance 3.846511e-10 1.369629e-10 9.470294e-11 7.706054e-11
## Cumulative Proportion  1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##                            Comp.57      Comp.58      Comp.59      Comp.60
## Standard deviation     6.571494e-03 5.047057e-03 2.949725e-03 2.507330e-03
## Proportion of Variance 4.929942e-11 2.907971e-11 9.932910e-12 7.176893e-12
## Cumulative Proportion  1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##                            Comp.61      Comp.62      Comp.63      Comp.64
## Standard deviation     1.324594e-03 6.551175e-04 5.002873e-04 5.277454e-06
## Proportion of Variance 2.002990e-12 4.899502e-13 2.857278e-13 3.179526e-17
## Cumulative Proportion  1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
##                            Comp.65      Comp.66 Comp.67 Comp.68 Comp.69
## Standard deviation     2.418746e-06 2.291078e-06       0       0       0
## Proportion of Variance 6.678735e-18 5.992295e-18       0       0       0
## Cumulative Proportion  1.000000e+00 1.000000e+00       1       1       1
##                        Comp.70 Comp.71
## Standard deviation           0       0
## Proportion of Variance       0       0
## Cumulative Proportion        1       1
```

```
plot(pr);plot(pr$scores[,1], pr$scores[,2])
```


pr
Variances
0e+00
2e+05
4e+05
Comp.1
Comp.3
Comp.5
Comp.7
Comp.9



The exploratory analysis includes a summary of the main numerical characteristics of each variable, a graphic representation of this attributes through a boxplot, a heat map of the matrix of variables, and a Principal Components Analysis of the data. The PCA shows that the first two principal components only catch 8.9% of the total variance. The first four principal components catch up to the 9.75% of the total variance.

2. Separate the data into 2 balanced partitions: a training set (2,815 compounds) and a test set (2,816 compounds). Use this same partition in the training phase (and validation phase if necessary) and the test phase of each of the sections that are presented below. Use the value 1234 as random seed to do the partition.

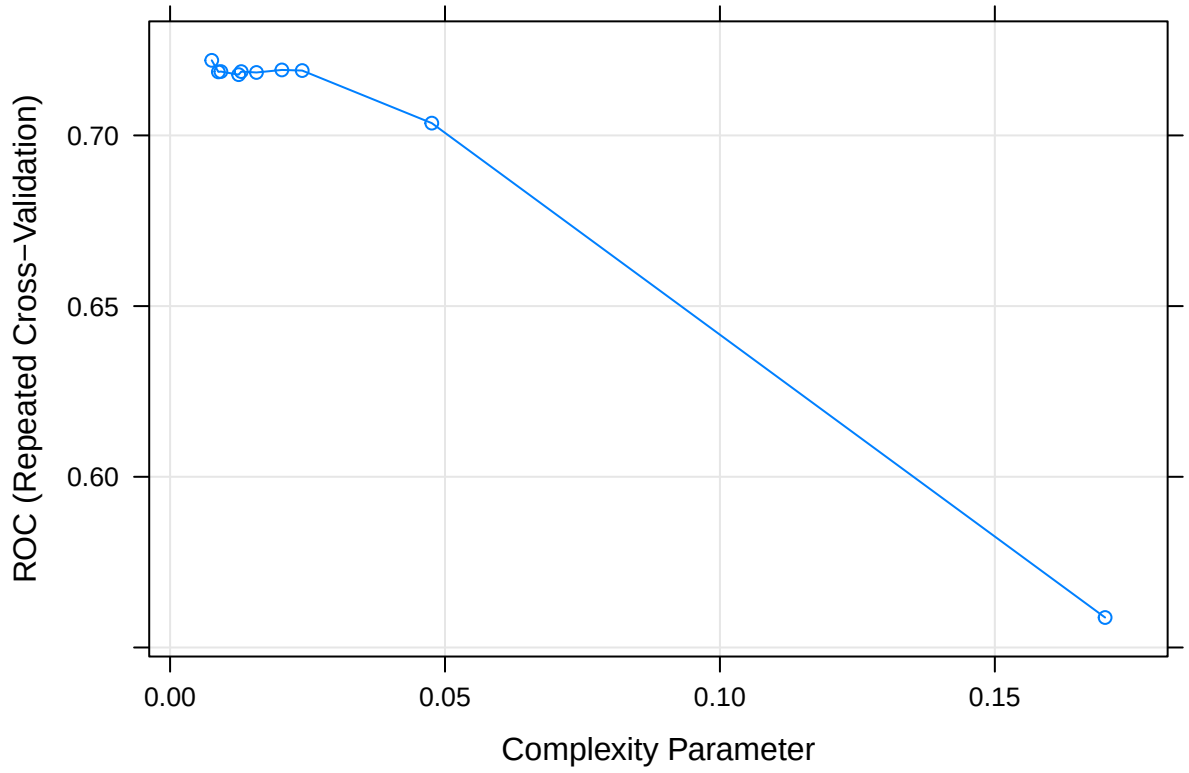
```
set.seed(1234)
#inTest <- createDataPartition(y=data$y, p=.5, list=FALSE)
inTest <- sample(nrow(data),
                 2816)
training<-data[-inTest,]
testing<-data[inTest,]
```

3. Fit a pruned single tree classifier to predict the aqueous solubility. Assess the performance of the tree by using suitable metrics.

```
ctrl <- trainControl(method = "repeatedcv", repeats=3, classProbs=TRUE,summaryFunction=twoClassSummary)
CART1Model <- train (y ~ .,
                     data=training,
                     method="rpart",
                     trControl=ctrl,
                     metric='ROC',
                     tuneLength=10,
                     preProc=c("center","scale"))
CART1Model
```

```
## CART
##
## 2815 samples
##   71 predictor
##    2 classes: 'Insoluble', 'Soluble'
##
## Pre-processing: centered (71), scaled (71)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 2534, 2533, 2533, 2534, 2534, 2534, ...
## Resampling results across tuning parameters:
##
##   cp           ROC        Sens       Spec
##   0.007578558  0.7219869  0.8328594  0.5470580
##   0.008780037  0.7186049  0.8365358  0.5338628
##   0.009242144  0.7187003  0.8346156  0.5329369
##   0.012476895  0.7177984  0.8267070  0.5301563
##   0.012939002  0.7186960  0.8274744  0.5270925
##   0.015711645  0.7184383  0.8284488  0.5129290
##   0.020332717  0.7191800  0.8236352  0.5138719
##   0.024029575  0.7190014  0.8192069  0.5157549
##   0.047597043  0.7035912  0.7428178  0.5933769
##   0.170055453  0.5587832  0.8638739  0.2536924
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.007578558.
```

```
plot(CART1Model)
```




```
CART2Probs <- predict(CART1Model, newdata = testing, type = "prob")
CART2Classes <- predict(CART1Model, newdata = testing, type = "raw")
conf_3<-confusionMatrix(data=CART2Classes,testing$y)
conf_3
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Insoluble Soluble
##   Insoluble      1464     467
##   Soluble         296     589
##
##                Accuracy : 0.729
##                  95% CI : (0.7122, 0.7454)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4026
##  Mcnemar's Test P-Value : 7.536e-10
##
##             Sensitivity : 0.8318
##             Specificity : 0.5578
##          Pos Pred Value : 0.7582
##          Neg Pred Value : 0.6655
##              Prevalence : 0.6250
##          Detection Rate : 0.5199
```

```
##    Detection Prevalence : 0.6857
##       Balanced Accuracy : 0.6948
##
##        'Positive' Class : Insoluble
##
```

4. Fit a Random Forest (RF) classifier to predict the aqueous solubility. Tune the parameters: number of trees and number of variables in each tree, by implementing a grid search procedure. Assess the performance of RF using suitable metrics. Determine which variables are the most relevant in the solubility prediction.

```
customRF <- list(type = "Classification", library = "randomForest", loop = NULL)
customRF$parameters <- data.frame(parameter = c("mtry", "ntree"), class = rep("numeric", 2), label = c(
customRF$grid <- function(x, y, len = NULL, search = "grid") {}
customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
}
customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata)
customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata, type = "prob")
customRF$sort <- function(x) x[order(x[,1]),]
customRF$levels <- function(x) x$classes


# train model
control <- trainControl(method="repeatedcv",
                        number=5,
                        repeats=1,
                        verbose=TRUE,
                        classProbs=TRUE,
                        search = "grid",
                        summaryFunction=twoClassSummary)

tunegrid <- expand.grid(.mtry=c(5,10,15), .ntree=c(1000, 1500))
set.seed(3001)
custom <- train(y ~ .,
                data = training,
                method=customRF,
                metric='ROC',
                tuneGrid=tunegrid,
                trControl=control)
```

```
## + Fold1.Rep1: mtry= 5, ntree=1000
## - Fold1.Rep1: mtry= 5, ntree=1000
## + Fold1.Rep1: mtry=10, ntree=1000
## - Fold1.Rep1: mtry=10, ntree=1000
## + Fold1.Rep1: mtry=15, ntree=1000
## - Fold1.Rep1: mtry=15, ntree=1000
## + Fold1.Rep1: mtry= 5, ntree=1500
## - Fold1.Rep1: mtry= 5, ntree=1500
## + Fold1.Rep1: mtry=10, ntree=1500
## - Fold1.Rep1: mtry=10, ntree=1500
## + Fold1.Rep1: mtry=15, ntree=1500
## - Fold1.Rep1: mtry=15, ntree=1500
```

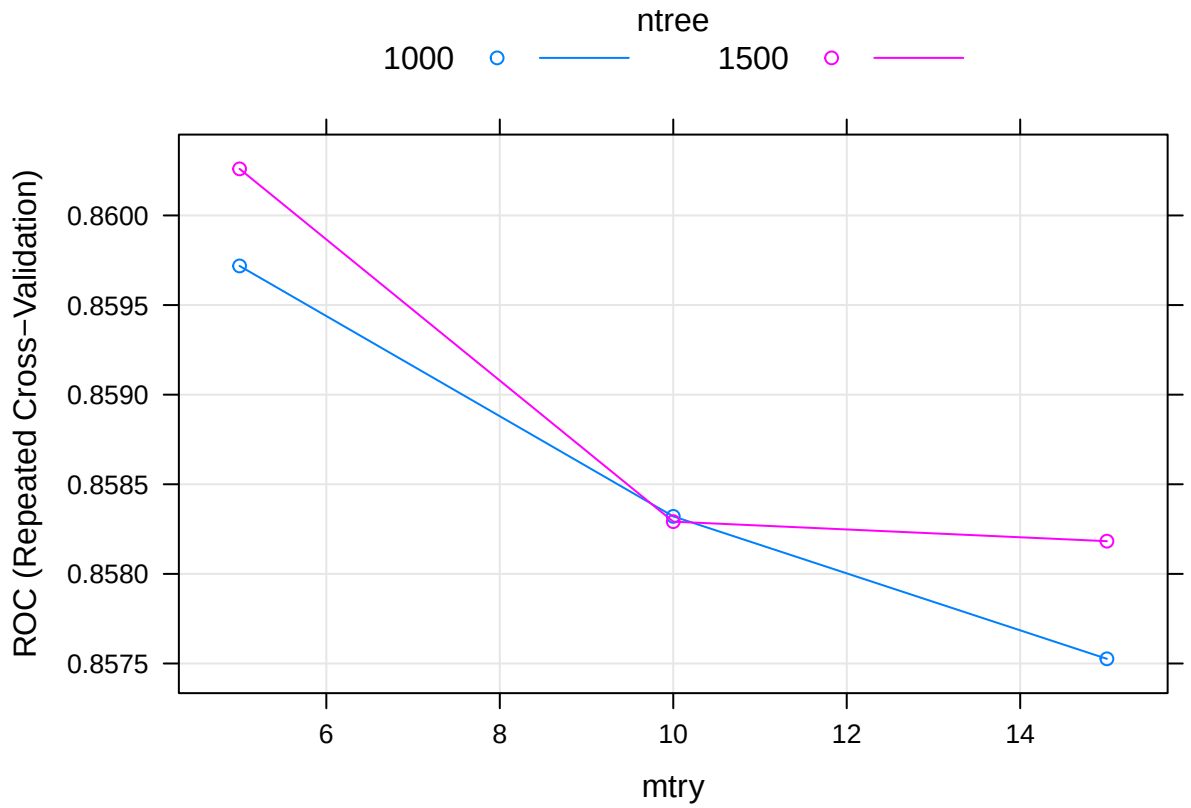
```
## + Fold2.Rep1: mtry= 5, ntree=1000
## - Fold2.Rep1: mtry= 5, ntree=1000
## + Fold2.Rep1: mtry=10, ntree=1000
## - Fold2.Rep1: mtry=10, ntree=1000
## + Fold2.Rep1: mtry=15, ntree=1000
## - Fold2.Rep1: mtry=15, ntree=1000
## + Fold2.Rep1: mtry= 5, ntree=1500
## - Fold2.Rep1: mtry= 5, ntree=1500
## + Fold2.Rep1: mtry=10, ntree=1500
## - Fold2.Rep1: mtry=10, ntree=1500
## + Fold2.Rep1: mtry=15, ntree=1500
## - Fold2.Rep1: mtry=15, ntree=1500
## + Fold3.Rep1: mtry= 5, ntree=1000
## - Fold3.Rep1: mtry= 5, ntree=1000
## + Fold3.Rep1: mtry=10, ntree=1000
## - Fold3.Rep1: mtry=10, ntree=1000
## + Fold3.Rep1: mtry=15, ntree=1000
## - Fold3.Rep1: mtry=15, ntree=1000
## + Fold3.Rep1: mtry= 5, ntree=1500
## - Fold3.Rep1: mtry= 5, ntree=1500
## + Fold3.Rep1: mtry=10, ntree=1500
## - Fold3.Rep1: mtry=10, ntree=1500
## + Fold3.Rep1: mtry=15, ntree=1500
## - Fold3.Rep1: mtry=15, ntree=1500
## + Fold4.Rep1: mtry= 5, ntree=1000
## - Fold4.Rep1: mtry= 5, ntree=1000
## + Fold4.Rep1: mtry=10, ntree=1000
## - Fold4.Rep1: mtry=10, ntree=1000
## + Fold4.Rep1: mtry=15, ntree=1000
## - Fold4.Rep1: mtry=15, ntree=1000
## + Fold4.Rep1: mtry= 5, ntree=1500
## - Fold4.Rep1: mtry= 5, ntree=1500
## + Fold4.Rep1: mtry=10, ntree=1500
## - Fold4.Rep1: mtry=10, ntree=1500
## + Fold4.Rep1: mtry=15, ntree=1500
## - Fold4.Rep1: mtry=15, ntree=1500
## + Fold5.Rep1: mtry= 5, ntree=1000
## - Fold5.Rep1: mtry= 5, ntree=1000
## + Fold5.Rep1: mtry=10, ntree=1000
## - Fold5.Rep1: mtry=10, ntree=1000
## + Fold5.Rep1: mtry=15, ntree=1000
## - Fold5.Rep1: mtry=15, ntree=1000
## + Fold5.Rep1: mtry= 5, ntree=1500
## - Fold5.Rep1: mtry= 5, ntree=1500
## + Fold5.Rep1: mtry=10, ntree=1500
## - Fold5.Rep1: mtry=10, ntree=1500
## + Fold5.Rep1: mtry=15, ntree=1500
## - Fold5.Rep1: mtry=15, ntree=1500
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 5, ntree = 1500 on full training set
```

```
summary(custom)
```

```
##                 Length Class      Mode
```

```
## call              5   -none-     call
## type              1   -none-     character
## predicted      2815   factor     numeric
## err.rate       4500   -none-     numeric
## confusion         6   -none-     numeric
## votes          5630   matrix     numeric
## oob.times      2815   -none-     numeric
## classes           2   -none-     character
## importance       71   -none-     numeric
## importanceSD      0   -none-     NULL
## localImportance   0   -none-     NULL
## proximity         0   -none-     NULL
## ntree             1   -none-     numeric
## mtry              1   -none-     numeric
## forest           14   -none-     list
## y              2815   factor     numeric
## test              0   -none-     NULL
## inbag             0   -none-     NULL
## xNames           71   -none-     character
## problemType       1   -none-     character
## tuneValue         2   data.frame list
## obsLevels         2   -none-     character
## param             0   -none-     list
```

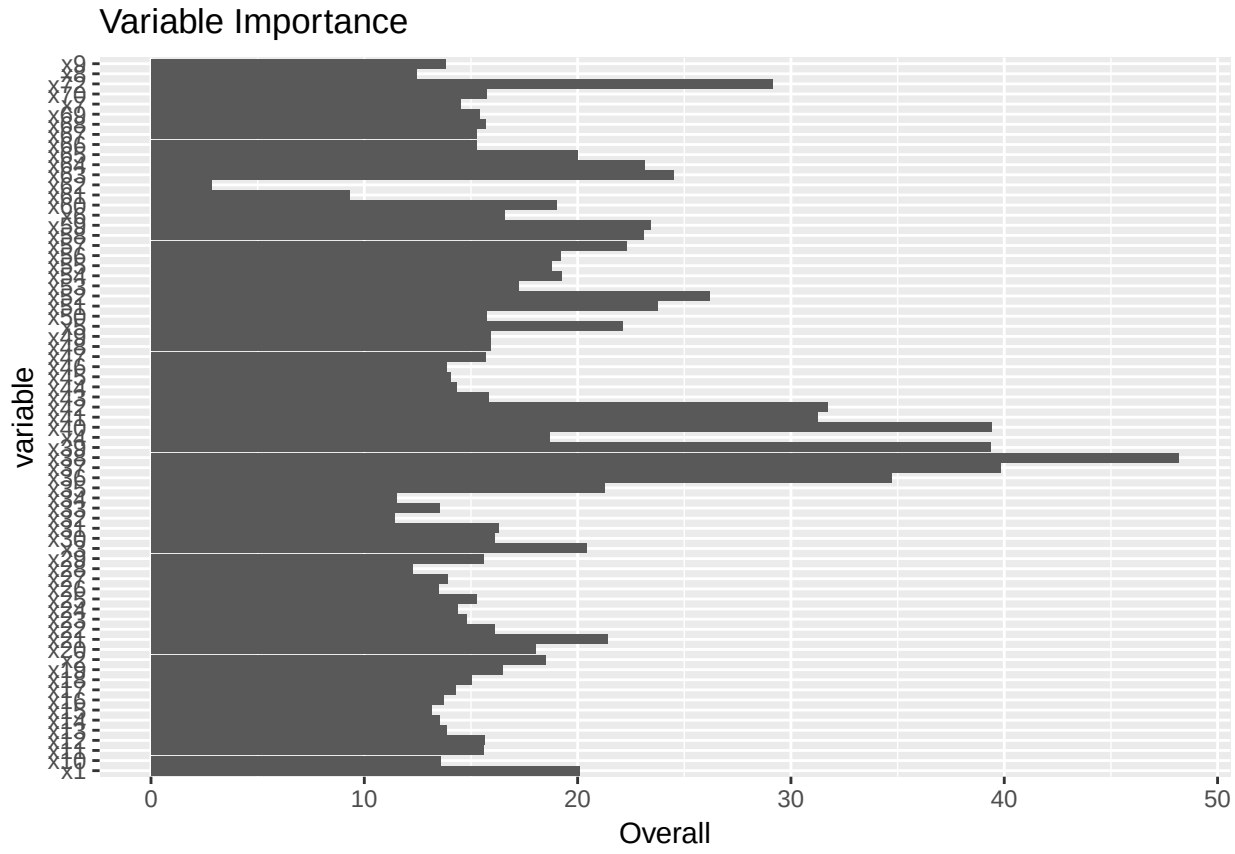
```
plot(custom)
```



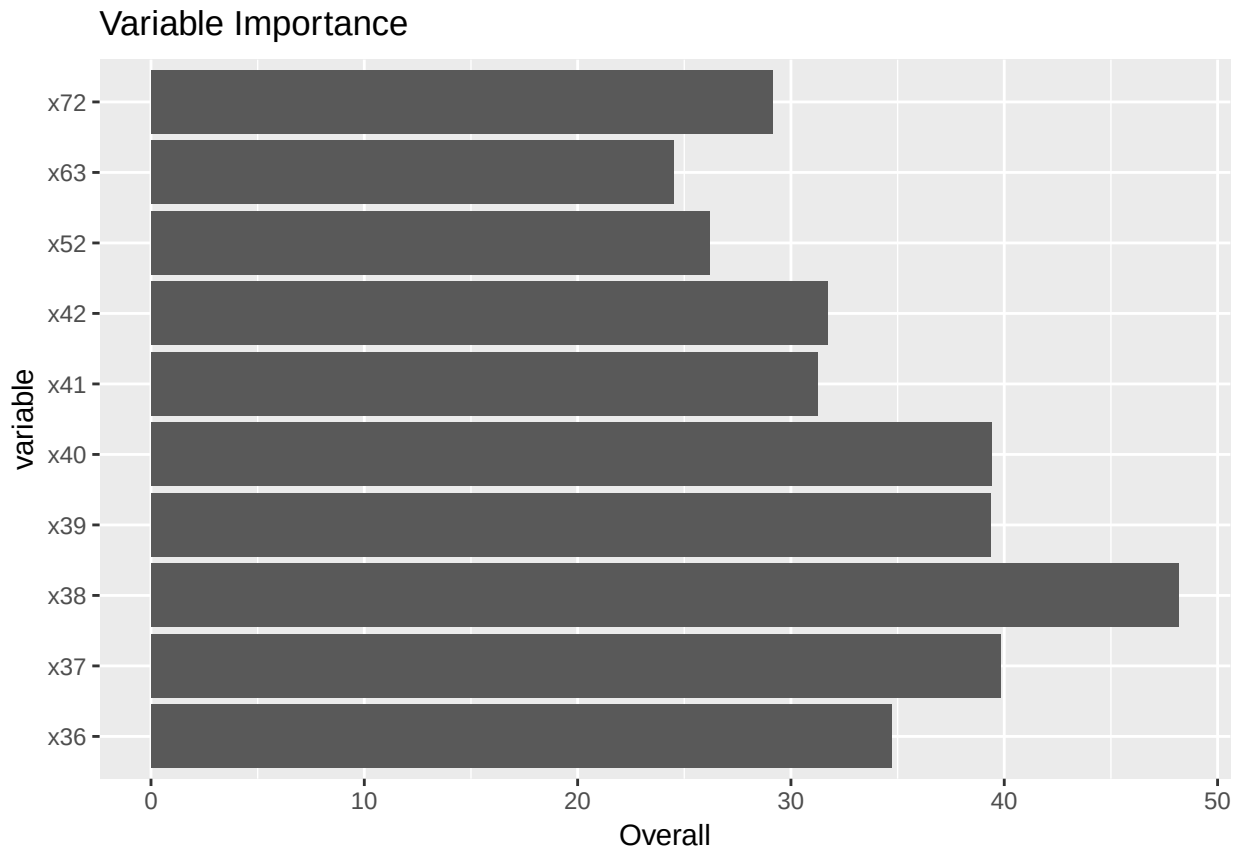
```r
conf_4<-confusionMatrix(data=predict(custom, newdata = testing, type = "raw"),testing$y)

#summary
kable(custom$results,digits=2)
```

mtry	ntree	ROC	Sens	Spec	ROCSD	SensSD	SpecSD
5	1000	0.86	0.88	0.63	0.04	0.03	0.06
5	1500	0.86	0.89	0.62	0.04	0.03	0.06
10	1000	0.86	0.88	0.63	0.04	0.03	0.06
10	1500	0.86	0.88	0.63	0.04	0.03	0.06
15	1000	0.86	0.88	0.63	0.04	0.03	0.07
15	1500	0.86	0.88	0.63	0.04	0.03	0.07

```r
#variable importance
var.imp <- custom$finalModel %>%
   varImp() %>%
  as.data.frame()
var.imp$variable <- rownames(var.imp)

ggplot(var.imp) +
  geom_bar(aes(x=variable,
               y=Overall),
           stat='identity') +
  coord_flip() +
  labs(title="Variable Importance")
```




```
#Top 10 Variables
var.imp %>%
  arrange(desc(Overall)) %>%
  head(10) %>%
  ggplot() +
  geom_bar(aes(x=variable,
               y=Overall),
           stat='identity') +
  coord_flip() +
  labs(title="Variable Importance")
```

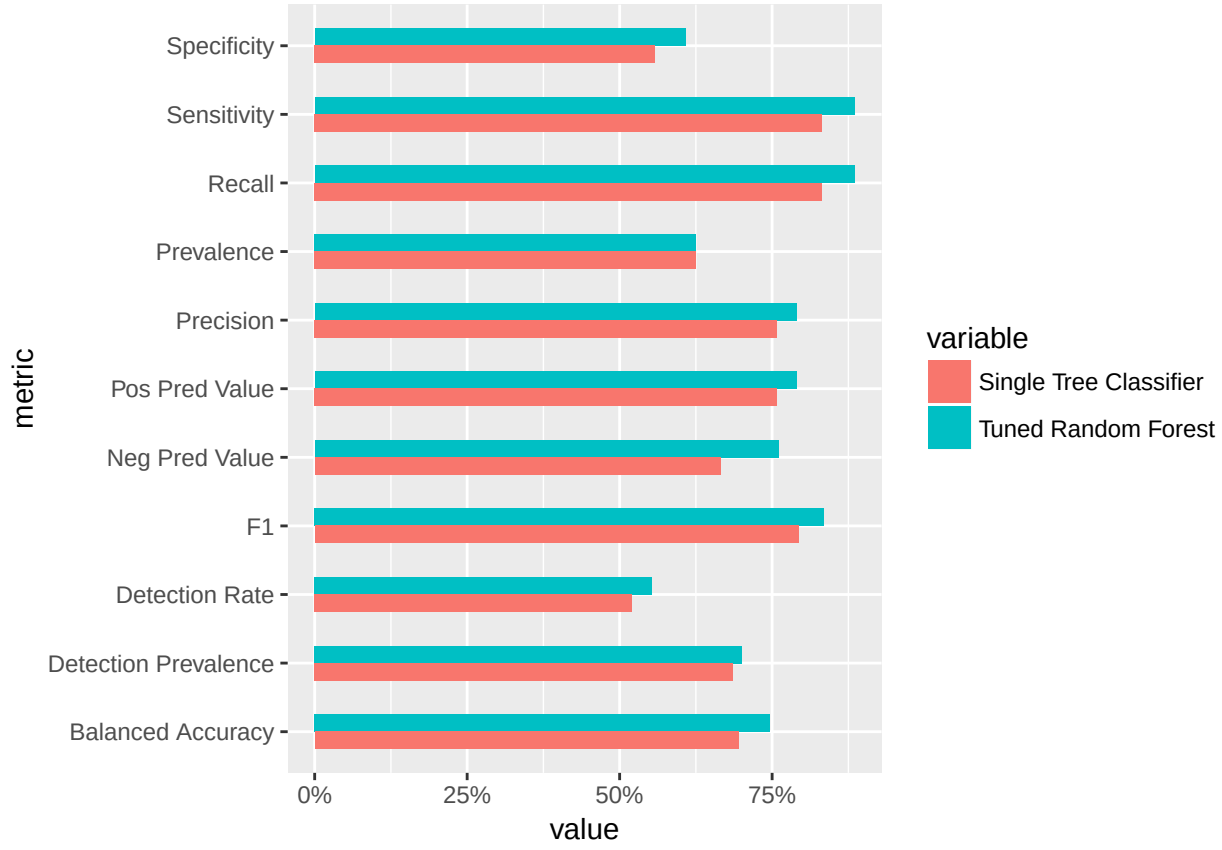



```
cat(paste0("Best model has mtry=",
           custom$bestTune$mtry,
           " and ntree=",
           custom$bestTune$ntree))
```

```
## Best model has mtry=5 and ntree=1500
```

5. In view of the above metrics, compare the classifiers in 3) and 4).

```
obj_plot_comparison<-cbind(conf_3$byClass,conf_4$byClass) %>%
  as.data.frame() %>%
  set_colnames(c("Single Tree Classifier", "Tuned Random Forest"))
obj_plot_comparison$metric<-as.vector(rownames(obj_plot_comparison))

p1 <- melt(obj_plot_comparison,id.var="metric") %>%
  ggplot(aes(x=metric,
             y=value,
             fill=variable)) +
  geom_bar(stat='identity',
           position="dodge",
           width=.5) +
  coord_flip() +
  scale_y_continuous(labels=percent)

p1
```



The tuned Random Forest is superior in ALL of the metrics when being compared to the single tree classifier.

6. Apply the discrete AdaBoost algorithm (with an exponential loss function).

```
training<-data[-inTest,]
training$y <- as.numeric(training$y=="Insoluble")
adaModel <- gbm(y ~ .,
                data = training,
                distribution="bernoulli", #since it is a classification problem
                n.trees=5000,
                interaction.depth=4)



yhat.boost = predict(adaModel,newdata=testing[,!names(testing)%in%c('y')],n.trees=5000)

confusionMatrix(data=as.factor(ifelse(yhat.boost>=0.5,"Insoluble","Soluble")),testing$y)
```

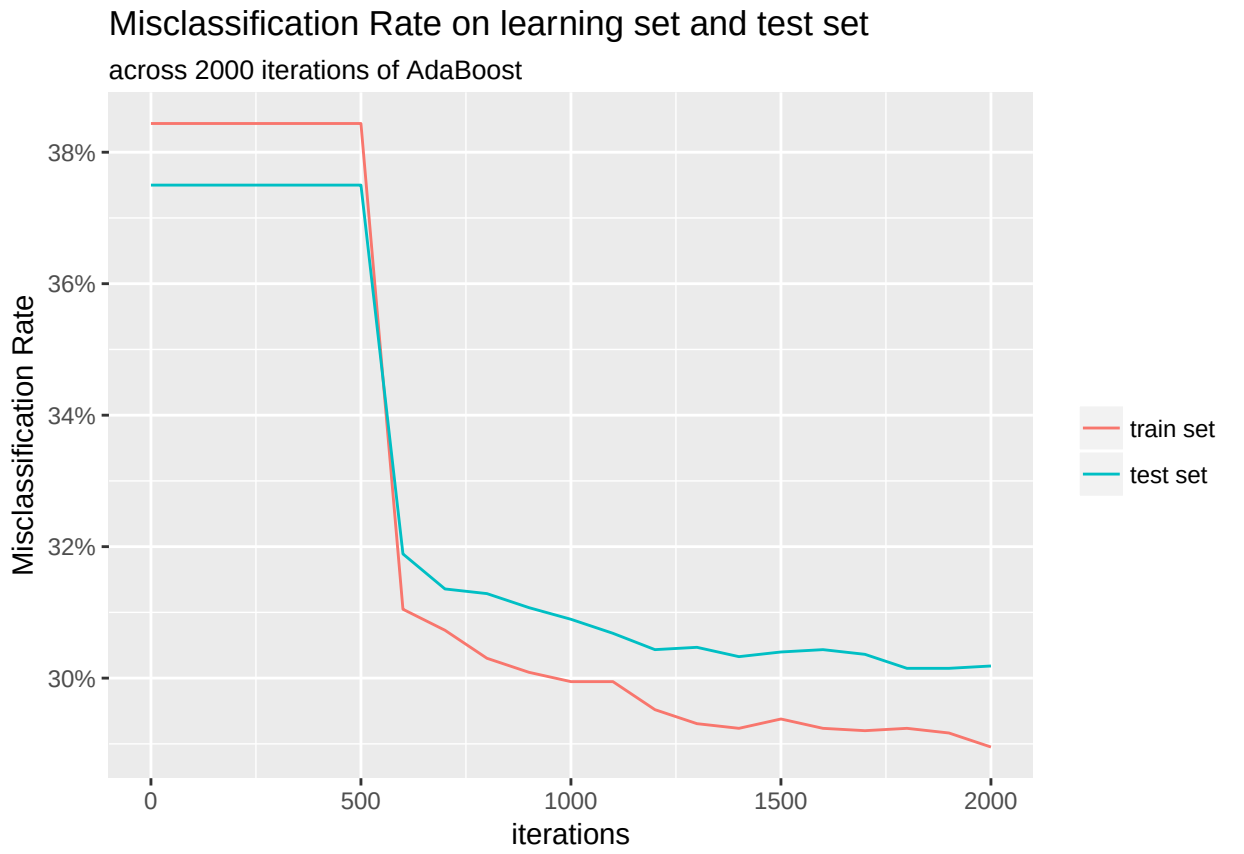
```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Insoluble Soluble
##   Insoluble      1355     294
##   Soluble         405     762
##
##                Accuracy : 0.7518
##                  95% CI : (0.7354, 0.7676)
##     No Information Rate : 0.625
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4814
##  Mcnemar's Test P-Value : 3.174e-05
##
##             Sensitivity : 0.7699
##             Specificity : 0.7216
##          Pos Pred Value : 0.8217
##          Neg Pred Value : 0.6530
##              Prevalence : 0.6250
##          Detection Rate : 0.4812
##    Detection Prevalence : 0.5856
##       Balanced Accuracy : 0.7457
##
##        'Positive' Class : Insoluble
##
```

6.1. Using "stumps" as classification trees compute the misclassification rates of both the learning set and the test set across 2,000 iterations of AdaBoost. Represent these error as a function of the number of boosting iterations.

```
#a stump is 1-node tree.
#https://stats.stackexchange.com/questions/16501/what-does-interaction-depth-mean-in-gbm

get_classification_rate_by_iterations <- function(x) {
adaModel = gbm(y ~ .,
               data = training,
               distribution="bernoulli", #since it is a classification problem
               n.trees=x,
               interaction.depth=1)

adaModel_pred_train <- predict(adaModel, training, n.trees = x, type = "response")

misclassification_rate_train <- mean(as.character(ifelse(adaModel_pred_train>=0.5,"Insoluble","Soluble"))

adaModel_pred_test <- predict(adaModel, testing, n.trees = x, type = "response")

misclassification_rate_test <- mean(as.character(ifelse(adaModel_pred_test>=0.5,"Insoluble","Soluble"))

return(c(misclassification_rate_train,misclassification_rate_test))
}

misclassification_rates_stump_train <- sapply(seq(0,2000,by=100),function(x) get_classification_rate_by_

cbind(t(misclassification_rates_stump_train),
      seq(0,2000,by=100)) %>%
  as.data.frame() %>%
  set_colnames(c("train set",
                 "test set",
                 "iterations")) %>%
  melt(id="iterations") %>%
  ggplot(aes(x=iterations,
             y=value,
             col=variable)) +
  geom_line() +
```

```
  scale_y_continuous(labels= percent) +
  labs(title="Misclassification Rate on learning set and test set",
       subtitle="across 2000 iterations of AdaBoost",
       y="Misclassification Rate",
       col="")
```



When the number of iterations are higher than 500, missclassification rate drops at a high scale. Before that, it was just constant. As expected, missclassification rate is higheron the test set than at the test set, and it decreases as iterations increase.

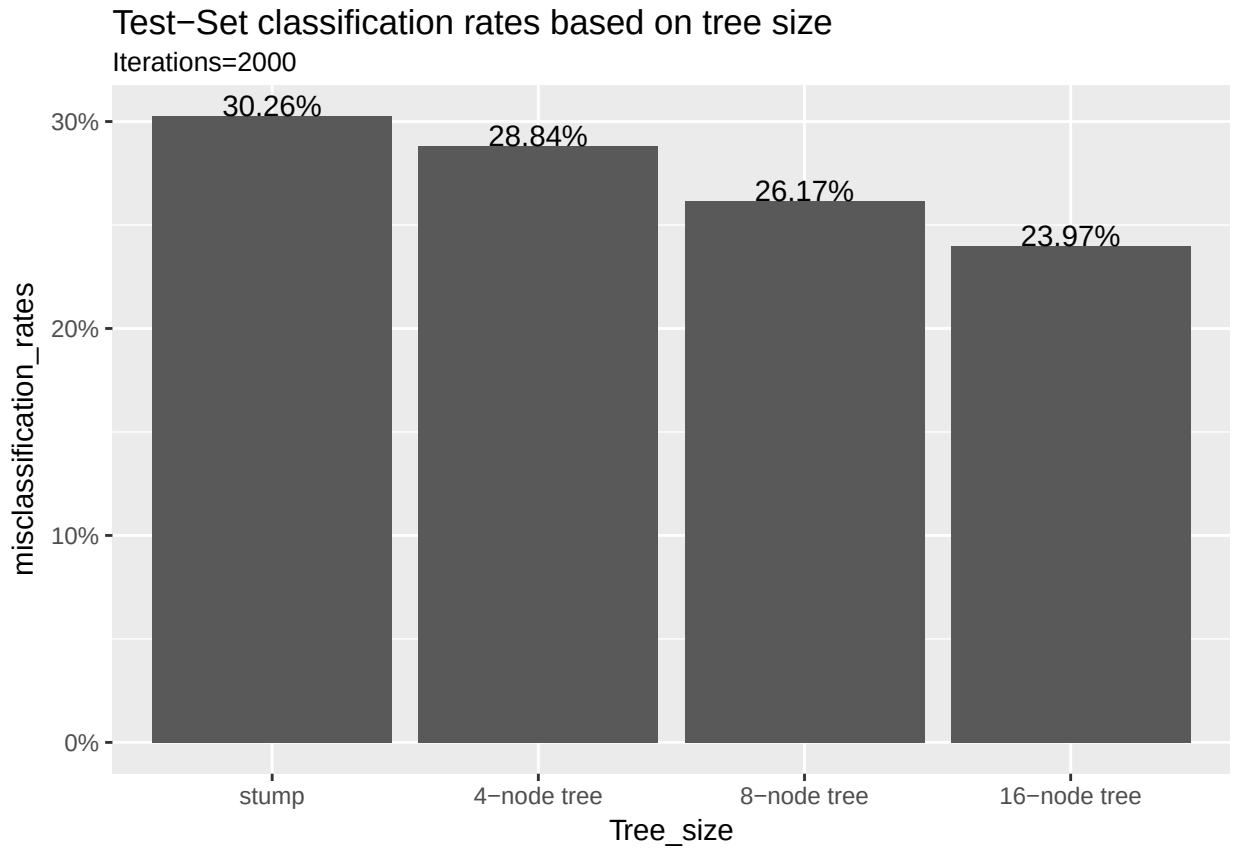
6.2. Compare the test-set misclassification rates attained by different ensemble classifiers based on trees of sizes: stumps, 4-node trees, 8-node trees, and 16-node trees.

```
get_test_classification_rate_by_tree_size <- function(x) {
adaModel = gbm(y ~ .,
               data = training,
               distribution="bernoulli", #since it is a classification problem
               n.trees=2000,
               interaction.depth=x)

adaModel_test_pred <- predict(adaModel, testing, n.trees = 2000, type = "response")

misclassification_rate <- mean(as.factor(ifelse(adaModel_test_pred>=0.5,"Insoluble","Soluble"))!=testing

return(misclassification_rate)
}
```

```
misclassification_rates <- sapply(c(1,2,4,16),function(x) get_test_classification_rate_by_tree_size(x))

t(misclassification_rates)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.3025568 0.2883523 0.2617188 0.2397017
```

```
misclassification_rates_plot <- cbind(misclassification_rates,c("stump","4-node tree","8-node tree","16
  as.data.frame() %>%
  set_colnames(c("misclassification_rates","Tree_size")) %>%
  mutate(misclassification_rates=as.numeric(as.character(misclassification_rates)),
         Tree_size=factor(Tree_size, levels = c("stump","4-node tree","8-node tree","16-node tree")))

ggplot(misclassification_rates_plot,
         aes(x=Tree_size,
             y=misclassification_rates,
             label=percent(misclassification_rates))) +
geom_bar(stat='identity') +
geom_text(vjust=0) +
scale_y_continuous(labels = percent) +
  labs(title="Test-Set classification rates based on tree size",
       subtitle="Iterations=2000")
```



Missclassification rate decreases as tree size increases.