projet survival and longitudinal data analysis

Amina Ghoul - Yamina Boubekeur - Daoued Karoui

22/10/2019

Introduction :

Le but de ce projet est de prévoir la probabilité de rechute du cancer du sein ("recurrent") à 24 mois. Pour cela, nous avons comparer les méthodes de l'analyse de survie (modèles de Cox, survival random forests) aux méthodes de classification (régression logistique, random forest).

Plan :

Le plan de notre travail se décompose de la manière suivante:

1. Traitement des données
2. Entrainement des diffèrents algorithmes de survie
3. Entrainement des diffèrents algorithmes de classification
4. Comparaison des algorithmes

- Packages:

```
library(KMsurv)
library(survival)
library(dplyr)
```

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

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

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

```
library(survminer)
```

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

```
## Loading required package: ggpubr
```

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

```
library(ggplot2)
library(ggfortify)
library(survival)
```

- Import des données:

Chaque ligne du dataset étudié représente les données de suivies pour un cas de cancer du sein.

```
DATA<-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wpbc.
glimpse(DATA)
```

```
## Observations: 198
## Variables: 35
## $ V1  <int> 119513, 8423, 842517, 843483, 843584, 843786, 844359, 8445...
```

```
## $ V2  <fct> N, N, N, N, R, R, N, R, N, N, N, N, N, R, N, R, N, R, N, N...
## $ V3  <int> 31, 61, 116, 123, 27, 77, 60, 77, 119, 76, 123, 125, 117, ...
## $ V4  <dbl> 18.02, 17.99, 21.37, 11.42, 20.29, 12.75, 18.98, 13.71, 13...
## $ V5  <dbl> 27.60, 10.38, 17.44, 20.38, 14.34, 15.29, 19.61, 20.83, 21...
## $ V6  <dbl> 117.50, 122.80, 137.50, 77.58, 135.10, 84.60, 124.40, 90.2...
## $ V7  <dbl> 1013.0, 1001.0, 1373.0, 386.1, 1297.0, 502.7, 1112.0, 577....
## $ V8  <dbl> 0.09489, 0.11840, 0.08836, 0.14250, 0.10030, 0.11890, 0.09...
## $ V9  <dbl> 0.10360, 0.27760, 0.11890, 0.28390, 0.13280, 0.15690, 0.12...
## $ V10 <dbl> 0.10860, 0.30010, 0.12550, 0.24140, 0.19800, 0.16640, 0.12...
## $ V11 <dbl> 0.07055, 0.14710, 0.08180, 0.10520, 0.10430, 0.07666, 0.08...
## $ V12 <dbl> 0.1865, 0.2419, 0.2333, 0.2597, 0.1809, 0.1995, 0.1727, 0....
## $ V13 <dbl> 0.06333, 0.07871, 0.06010, 0.09744, 0.05883, 0.07164, 0.05...
## $ V14 <dbl> 0.6249, 1.0950, 0.5854, 0.4956, 0.7572, 0.3877, 0.5285, 0....
## $ V15 <dbl> 1.8900, 0.9053, 0.6105, 1.1560, 0.7813, 0.7402, 0.8434, 1....
## $ V16 <dbl> 3.972, 8.589, 3.928, 3.445, 5.438, 2.999, 3.592, 3.856, 2....
## $ V17 <dbl> 71.55, 153.40, 82.15, 27.23, 94.44, 30.85, 61.21, 50.96, 2...
## $ V18 <dbl> 0.004433, 0.006399, 0.006167, 0.009110, 0.011490, 0.007775...
## $ V19 <dbl> 0.014210, 0.049040, 0.034490, 0.074580, 0.024610, 0.029870...
## $ V20 <dbl> 0.03233, 0.05373, 0.03300, 0.05661, 0.05688, 0.04561, 0.02...
## $ V21 <dbl> 0.009854, 0.015870, 0.018050, 0.018670, 0.018850, 0.013570...
## $ V22 <dbl> 0.01694, 0.03003, 0.03094, 0.05963, 0.01756, 0.01774, 0.01...
## $ V23 <dbl> 0.003495, 0.006193, 0.005039, 0.009208, 0.005115, 0.005114...
## $ V24 <dbl> 21.63, 25.38, 24.90, 14.91, 22.54, 15.51, 23.39, 17.06, 15...
## $ V25 <dbl> 37.08, 17.33, 20.98, 26.50, 16.67, 20.37, 25.45, 28.14, 30...
## $ V26 <dbl> 139.70, 184.60, 159.10, 98.87, 152.20, 107.30, 152.60, 110...
## $ V27 <dbl> 1436.0, 2019.0, 1949.0, 567.7, 1575.0, 733.2, 1593.0, 897....
## $ V28 <dbl> 0.1195, 0.1622, 0.1188, 0.2098, 0.1374, 0.1706, 0.1144, 0....
## $ V29 <dbl> 0.1926, 0.6656, 0.3449, 0.8663, 0.2050, 0.4196, 0.3371, 0....
## $ V30 <dbl> 0.3140, 0.7119, 0.3414, 0.6869, 0.4000, 0.5999, 0.2990, 0....
## $ V31 <dbl> 0.11700, 0.26540, 0.20320, 0.25750, 0.16250, 0.17090, 0.19...
## $ V32 <dbl> 0.2677, 0.4601, 0.4334, 0.6638, 0.2364, 0.3485, 0.2726, 0....
## $ V33 <dbl> 0.08113, 0.11890, 0.09067, 0.17300, 0.07678, 0.11790, 0.09...
## $ V34 <dbl> 5.0, 3.0, 2.5, 2.0, 3.5, 2.5, 1.5, 4.0, 2.0, 6.0, 2.0, 1.4...
## $ V35 <fct> 5, 2, 0, 0, 0, 0, ?, 10, 1, 20, 0, 0, 0, 6, 0, 1, 0, 1, 0,...
```

Ce jeu de données comprend 198 lignes et 35 variables.

On remarque que sur la colonne V35, il y a des valeurs manquantes notées "?", qu'on va remplacer par la suite par "NA"

```r
rm(DATA)
wpbc<-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wpbc.
wpbc[,35]
```

```
##   [1]  5  2  0  0  0  0 NA 10  1 20  0  0  0  6  0  1  0  1  0  1  0 13  0
##  [24]  0  2  1  0  0 NA 13 10  0  0  0  0  1  1  0  1  0 13  6  0  1  0  4
##  [47]  2  0  1  0  2  1  0  0  4  2  1 17  0 15 11  0  9  0  8  1  0  7  2
##  [70]  0  3  1  2  1  1  4  7  1  0  3  0  4  9  0  1 NA 14  0  0  0  1  0
##  [93]  2  0  0  1 27  5 24  1  0  1  0  7  0 15  0  0  3  1  1  0  6  2 11
## [116]  0  0 15  0 18  0 11  0  1  2  2  0  0  4 13  0  0  0  0  2  1  0  0
## [139]  0 13 16  3 13  0  1 27  0  4  0  0  7  0  7  0  0  9  0  2  0 20  0
## [162]  4  1  8  1  4  1  1  0  0  0  0  1  1  2  0  9  4  0  2  0  0  4  0
## [185]  0  0  0  1  7  2  0  1 21  2  0  0 NA  0
```

- On renomme les variables à l'aide de la documentation fournie :

```r
names_cov = paste0(rep(c('radius','texture','perimeter','area','smoothness','compactness',
                          'concavity','concave_points','symmetry','fractal_dimension'),3),
                   c(rep('_mean',10),rep('_SD',10),rep('_worst',10)))
names(wpbc) = c('id','recurrent','time',names_cov,c('Tumor_size','Lymph_node_status'))
```

Les variables sont : **id** : l'identifiant de la patiente, **recurrent** : R=rechute, N=non rechute, **time** : temps de rechute si R, temps sans maladie si N, 10 variables réelles obtenues pour chaque noyau cellulaire : **radius**, **texture**, **perimeter**,**area**, **smoothness**, **compactness**, **concavity**,**concave points**, **symmetry**, **fractal dimension**

Pour chacune de ces variables, on a leur moyenne **_mean** , leur écart-type **_SD** , et la moyenne des 3 plus grandes valeurs **_worst**.

Tumor_size: diamètre de la tumeur, **Lymph_node_status**: nombre de ganglions lymphatiques positifs

Ensuite on transforme *id* en factor et *reccurent* en factor TRUE =N et FALSE = R

```r
wpbc = wpbc %>% mutate(id = factor(id)) %>%
                mutate( recurrent = recode_factor(recurrent , "N" = TRUE, 'R' = FALSE ))
glimpse(wpbc)
```

```
## Observations: 198
## Variables: 35
## $ id                      <fct> 119513, 8423, 842517, 843483, 843584, ...
## $ recurrent               <fct> TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, ...
## $ time                    <int> 31, 61, 116, 123, 27, 77, 60, 77, 119,...
## $ radius_mean             <dbl> 18.02, 17.99, 21.37, 11.42, 20.29, 12....
## $ texture_mean            <dbl> 27.60, 10.38, 17.44, 20.38, 14.34, 15....
## $ perimeter_mean          <dbl> 117.50, 122.80, 137.50, 77.58, 135.10,...
## $ area_mean               <dbl> 1013.0, 1001.0, 1373.0, 386.1, 1297.0,...
## $ smoothness_mean         <dbl> 0.09489, 0.11840, 0.08836, 0.14250, 0....
## $ compactness_mean        <dbl> 0.10360, 0.27760, 0.11890, 0.28390, 0....
## $ concavity_mean          <dbl> 0.10860, 0.30010, 0.12550, 0.24140, 0....
## $ concave_points_mean     <dbl> 0.07055, 0.14710, 0.08180, 0.10520, 0....
## $ symmetry_mean           <dbl> 0.1865, 0.2419, 0.2333, 0.2597, 0.1809...
## $ fractal_dimension_mean  <dbl> 0.06333, 0.07871, 0.06010, 0.09744, 0....
## $ radius_SD               <dbl> 0.6249, 1.0950, 0.5854, 0.4956, 0.7572...
## $ texture_SD              <dbl> 1.8900, 0.9053, 0.6105, 1.1560, 0.7813...
## $ perimeter_SD            <dbl> 3.972, 8.589, 3.928, 3.445, 5.438, 2.9...
## $ area_SD                 <dbl> 71.55, 153.40, 82.15, 27.23, 94.44, 30...
## $ smoothness_SD           <dbl> 0.004433, 0.006399, 0.006167, 0.009110...
## $ compactness_SD          <dbl> 0.014210, 0.049040, 0.034490, 0.074580...
## $ concavity_SD            <dbl> 0.03233, 0.05373, 0.03300, 0.05661, 0....
## $ concave_points_SD       <dbl> 0.009854, 0.015870, 0.018050, 0.018670...
## $ symmetry_SD             <dbl> 0.01694, 0.03003, 0.03094, 0.05963, 0....
## $ fractal_dimension_SD    <dbl> 0.003495, 0.006193, 0.005039, 0.009208...
## $ radius_worst            <dbl> 21.63, 25.38, 24.90, 14.91, 22.54, 15....
## $ texture_worst           <dbl> 37.08, 17.33, 20.98, 26.50, 16.67, 20....
## $ perimeter_worst         <dbl> 139.70, 184.60, 159.10, 98.87, 152.20,...
## $ area_worst              <dbl> 1436.0, 2019.0, 1949.0, 567.7, 1575.0,...
## $ smoothness_worst        <dbl> 0.1195, 0.1622, 0.1188, 0.2098, 0.1374...
## $ compactness_worst       <dbl> 0.1926, 0.6656, 0.3449, 0.8663, 0.2050...
## $ concavity_worst         <dbl> 0.3140, 0.7119, 0.3414, 0.6869, 0.4000...
## $ concave_points_worst    <dbl> 0.11700, 0.26540, 0.20320, 0.25750, 0....
## $ symmetry_worst          <dbl> 0.2677, 0.4601, 0.4334, 0.6638, 0.2364...
## $ fractal_dimension_worst <dbl> 0.08113, 0.11890, 0.09067, 0.17300, 0....
```

```
## $ Tumor_size              <dbl> 5.0, 3.0, 2.5, 2.0, 3.5, 2.5, 1.5, 4.0...
## $ Lymph_node_status       <int> 5, 2, 0, 0, 0, 0, NA, 10, 1, 20, 0, 0,...
```

- On transforme *time* en numérique

```
wpbc = dplyr::mutate(wpbc,time=as.numeric(time))
```

- Gérer les NA: On remplace les "NA" par la médiane

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:magrittr':
##
##     extract
```

```
DATA_NA<-wpbc %>% replace_na(list(`Lymph_node_status`=median(wpbc$`Lymph_node_status`,na.rm =T)))
sum(is.na(DATA_NA))
```

```
## [1] 0
```

Il n'y a plus de données manquantes dans notre jeu de données.

```
head(DATA_NA$perimeter_mean)
```

```
## [1] 117.50 122.80 137.50  77.58 135.10  84.60
```

```
head(DATA_NA$concavity_mean)
```

```
## [1] 0.1086 0.3001 0.1255 0.2414 0.1980 0.1664
```

On remarque que par exemple les valeurs des colonnes *perimeter_mean* et *concavity_mean* n'ont pas le même ordre de grandeur. Il faut alors, normaliser et centrer toutes les colonnes qui ont des valeurs numériques pour pouvoir les comparer entres elles.

```
scale <- function(x)(x- mean(x,na.rm=T))/sd(x,na.rm=T)
DATA_stan<- DATA_NA %>% mutate_at(names(DATA_NA)[-c(1,2,3)], scale)
print(head(DATA_stan$perimeter_mean))
```

```
## [1]  0.1236209  0.3714766  1.0589257 -1.7432477  0.9466892 -1.4149557
```

```
print(head(DATA_stan$concavity_mean))
```

```
## [1] -0.6750921  2.0384386 -0.4356213  1.2066670  0.5916945  0.1439266
```

On s'interresse à la probabilité de rechute à 24 mois.

On crée alors la variable de censure notée Z, il s'agit de la survenue ou non de l'évenement étudié, cette variable discréte Z codée:

- Z=1 si time <= 24 et recurrent = TRUE
- Z=0 si time >24 et recurrent = TRUE ou recurrent = FALSE (i.e la donnée est censurée)
- Z=NA si time <=24 et recurrent = FALSE

```
DATA_stan=DATA_stan %>%
  mutate(Z=ifelse((time<=24)&(recurrent==TRUE),1,
                ifelse((time>24)&(recurrent==TRUE),0,
                       ifelse((time>24)&(recurrent==FALSE),0,NA))))

sum(is.na(DATA_stan$Z))
```

```
## [1] 29
```

```r
dim(DATA_stan)
```

```
## [1] 198  36
```

Il y a 29 valeurs manquantes, on supprime les lignes contenant les NA.

```r
data_class = dplyr::filter(DATA_stan,is.na(Z)==FALSE)
data_class$Z = as.factor(data_class$Z)
dim(data_class)
```

```
## [1] 169  36
```

Il reste alors 169 lignes dans notre jeu de données.

- On sépare le train et le test:

L'échantillon d'entraînement est un sous-échantillon stratifié composé de 80% du dataset.

L'échantillon de test est un sous-échantillon stratifié composé de 20% du dataset.

Les échantilons train et test utilisés pour la classification contiennent la variable prédictive Z contrairement aux échantillons train et test utilisés pour les modèles de survie.

```r
library(caTools)
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

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

```r
set.seed(42)

data_class$recurrent=as.logical(data_class$recurrent)
data_class= data_class %>% mutate(id_1n=c(1:nrow(data_class)))
trainIndex = createDataPartition(data_class$recurrent, p=0.8, list=FALSE,times=1)

# échantillons train et test pour la classification
data_class_train <- data_class[trainIndex, ]
data_class_test <-  data_class[-trainIndex, ]

# échantillons train et test pour les modèles de survie
data_class_train_surv=data_class_train %>% dplyr::select(-Z)
data_class_test_surv=data_class_test %>% dplyr::select(-Z)
```

Entrainement des diffèrents algorithmes de survie

Kaplan Meier

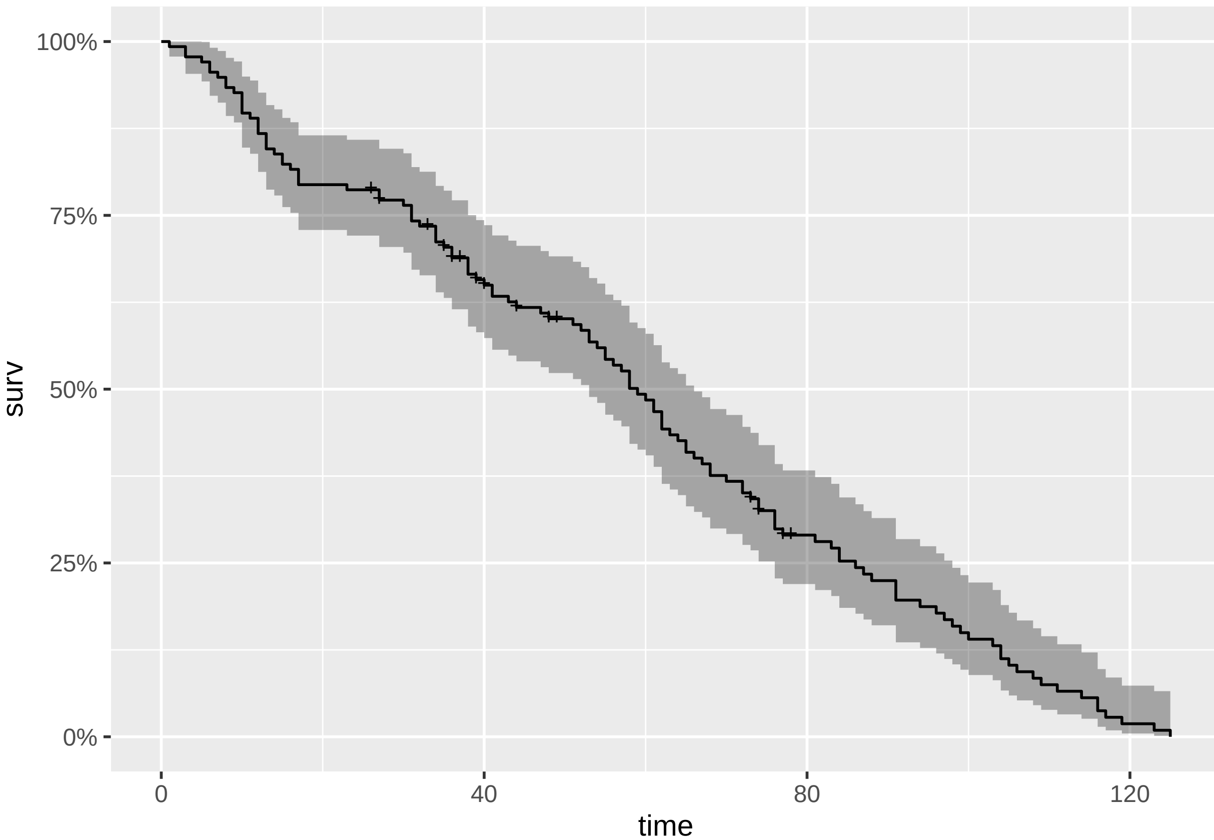
L'estimation de la fonction de survie de **Kaplan-Meier** s'obtient avec la fonction *survfit {survival}*.

```r
#estimation de la fonction de survie
km <- survfit(Surv(time, recurrent) ~ 1, data = data_class_train_surv)
summary(km)
```

```
## Call: survfit(formula = Surv(time, recurrent) ~ 1, data = data_class_train_surv)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     1    136       1  0.99265 0.00733      0.97839       1.0000
##     3    135       2  0.97794 0.01259      0.95357       1.0000
##     5    133       1  0.97059 0.01449      0.94260       0.9994
##     6    132       2  0.95588 0.01761      0.92198       0.9910
##     7    130       1  0.94853 0.01895      0.91211       0.9864
##     8    129       2  0.93382 0.02132      0.89296       0.9766
##     9    127       1  0.92647 0.02238      0.88363       0.9714
##    10    126       4  0.89706 0.02606      0.84741       0.9496
##    11    122       1  0.88971 0.02686      0.83859       0.9439
##    12    121       3  0.86765 0.02906      0.81252       0.9265
##    13    118       3  0.84559 0.03098      0.78699       0.9086
##    14    115       1  0.83824 0.03158      0.77858       0.9025
##    15    114       2  0.82353 0.03269      0.76189       0.8902
##    16    112       1  0.81618 0.03321      0.75361       0.8839
##    17    111       3  0.79412 0.03467      0.72899       0.8651
##    23    108       1  0.78676 0.03512      0.72085       0.8587
##    27    106       2  0.77192 0.03599      0.70450       0.8458
##    30    103       1  0.76443 0.03642      0.69628       0.8392
##    31    102       3  0.74194 0.03759      0.67181       0.8194
##    32     99       1  0.73445 0.03795      0.66371       0.8127
##    34     97       3  0.71173 0.03897      0.63930       0.7924
##    35     94       1  0.70416 0.03929      0.63122       0.7855
##    36     92       2  0.68885 0.03990      0.61493       0.7717
##    38     88       3  0.66537 0.04078      0.59006       0.7503
##    39     85       1  0.65754 0.04104      0.58183       0.7431
##    40     83       1  0.64962 0.04130      0.57351       0.7358
##    41     81       2  0.63358 0.04181      0.55671       0.7211
##    43     79       1  0.62556 0.04205      0.54835       0.7136
##    44     78       1  0.61754 0.04226      0.54002       0.7062
##    47     76       1  0.60941 0.04248      0.53159       0.6986
##    48     75       1  0.60129 0.04269      0.52319       0.6911
##    51     72       1  0.59294 0.04290      0.51454       0.6833
##    52     71       1  0.58459 0.04310      0.50593       0.6755
##    53     70       2  0.56788 0.04346      0.48879       0.6598
##    54     68       1  0.55953 0.04362      0.48026       0.6519
##    55     67       2  0.54283 0.04388      0.46329       0.6360
##    56     65       1  0.53448 0.04400      0.45485       0.6281
##    57     64       1  0.52613 0.04409      0.44643       0.6201
##    58     63       3  0.50107 0.04430      0.42135       0.5959
##    59     60       1  0.49272 0.04434      0.41304       0.5878
##    60     59       1  0.48437 0.04437      0.40476       0.5796
##    61     58       2  0.46767 0.04439      0.38829       0.5633
##    62     56       3  0.44262 0.04430      0.36377       0.5385
##    63     53       1  0.43426 0.04425      0.35565       0.5303
##    64     52       1  0.42591 0.04418      0.34756       0.5219
##    65     51       2  0.40921 0.04400      0.33146       0.5052
##    66     49       1  0.40086 0.04388      0.32345       0.4968
##    67     48       1  0.39251 0.04376      0.31547       0.4884
##    68     47       2  0.37581 0.04346      0.29959       0.4714
##    70     45       1  0.36745 0.04329      0.29169       0.4629
##    72     44       2  0.35075 0.04290      0.27599       0.4458
```

```
##    73      42       1  0.34240 0.04268       0.26818        0.4372
##    74      40       2  0.32528 0.04223       0.25220        0.4195
##    76      37       3  0.29891 0.04146       0.22775        0.3923
##    77      34       1  0.29012 0.04116       0.21968        0.3831
##    81      31       1  0.28076 0.04089       0.21104        0.3735
##    83      30       1  0.27140 0.04058       0.20246        0.3638
##    84      29       2  0.25268 0.03988       0.18545        0.3443
##    86      27       1  0.24332 0.03949       0.17703        0.3344
##    87      26       1  0.23396 0.03906       0.16867        0.3245
##    88      25       1  0.22461 0.03860       0.16037        0.3146
##    91      24       3  0.19653 0.03703       0.13585        0.2843
##    94      21       1  0.18717 0.03643       0.12782        0.2741
##    96      20       1  0.17781 0.03579       0.11985        0.2638
##    97      19       1  0.16845 0.03511       0.11197        0.2534
##    98      18       1  0.15910 0.03438       0.10416        0.2430
##    99      17       1  0.14974 0.03361       0.09645        0.2325
##   100      16       1  0.14038 0.03278       0.08882        0.2219
##   103      15       1  0.13102 0.03191       0.08129        0.2112
##   104      14       2  0.11230 0.02997       0.06657        0.1895
##   105      12       1  0.10294 0.02889       0.05939        0.1785
##   106      11       1  0.09359 0.02774       0.05235        0.1673
##   108      10       1  0.08423 0.02650       0.04546        0.1560
##   109       9       1  0.07487 0.02515       0.03875        0.1446
##   111       8       1  0.06551 0.02369       0.03225        0.1331
##   114       7       1  0.05615 0.02207       0.02599        0.1213
##   116       6       2  0.03743 0.01826       0.01439        0.0974
##   117       4       1  0.02808 0.01591       0.00925        0.0853
##   119       3       1  0.01872 0.01307       0.00476        0.0736
##   123       2       1  0.00936 0.00930       0.00133        0.0656
##   125       1       1  0.00000     NaN            NA            NA
```

```
autoplot(km)   #représentation de la courbe de survie
```



Remarque: Un intervalle de confiance à 95% de type "log" calculé sur le log de la fonction de survie et qui donne une meilleure estimation de l'intervalle de confiance de la fonction de survie (représenté en gris). les traits verticaux sur la courbe représentent les individus censurés.

On remarque que par exemple, la probabilité de rechute du cancer du sein à 27 mois est de 0.77192 et l'intervalle de confiance (CI = 0.70450 ,0.8458)

Modèle de Cox:

Un modèle de Cox se calcule avec *coxph {survival}*

- Entrainement sur le *train* avec toutes les variables.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

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

```
cox_fit<-coxph(Surv(time, recurrent) ~. -id -id_1n, data=data_class_train_surv)
summary(cox_fit)
```

```
## Call:
## coxph(formula = Surv(time, recurrent) ~ . - id - id_1n, data = data_class_train_surv)
##
##   n= 136, number of events= 121
##
```

```
##                            coef exp(coef) se(coef)      z Pr(>|z|)
## radius_mean             0.90159   2.46351  3.54617  0.254 0.799308
## texture_mean            0.68253   1.97888  0.36044  1.894 0.058280 .
## perimeter_mean         -0.47847   0.61973  3.81352 -0.125 0.900154
## area_mean               0.55989   1.75048  1.88216  0.297 0.766106
## smoothness_mean        -0.36178   0.69643  0.34216 -1.057 0.290348
## compactness_mean       -0.14043   0.86899  0.67668 -0.208 0.835597
## concavity_mean          0.61301   1.84597  0.59740  1.026 0.304831
## concave_points_mean    -0.86239   0.42215  0.63946 -1.349 0.177458
## symmetry_mean          -0.62791   0.53371  0.24196 -2.595 0.009456 **
## fractal_dimension_mean  0.59757   1.81770  0.45316  1.319 0.187278
## radius_SD               1.14190   3.13271  1.02440  1.115 0.264977
## texture_SD              0.79080   2.20516  0.23342  3.388 0.000704 ***
## perimeter_SD           -2.22719   0.10783  0.91876 -2.424 0.015345 *
## area_SD                 1.32283   3.75403  0.83314  1.588 0.112339
## smoothness_SD          -1.04862   0.35042  0.26632 -3.937 8.24e-05 ***
## compactness_SD         -1.08098   0.33926  0.58633 -1.844 0.065238 .
## concavity_SD            1.30727   3.69607  0.51641  2.531 0.011359 *
## concave_points_SD       0.13189   1.14098  0.31320  0.421 0.673681
## symmetry_SD            -0.35780   0.69921  0.23636 -1.514 0.130075
## fractal_dimension_SD    0.51937   1.68096  0.45240  1.148 0.250959
## radius_worst           -2.92528   0.05365  1.97259 -1.483 0.138085
## texture_worst          -0.50795   0.60173  0.43769 -1.161 0.245843
## perimeter_worst         4.19080  66.07539  1.60334  2.614 0.008954 **
## area_worst             -2.09929   0.12254  1.42147 -1.477 0.139717
## smoothness_worst        0.61774   1.85473  0.32112  1.924 0.054391 .
## compactness_worst       0.32846   1.38883  0.65686  0.500 0.617045
## concavity_worst        -0.69472   0.49921  0.49766 -1.396 0.162716
## concave_points_worst    0.25773   1.29399  0.30114  0.856 0.392082
## symmetry_worst          0.56048   1.75151  0.35085  1.597 0.110157
## fractal_dimension_worst -0.69269  0.50023  0.56178 -1.233 0.217559
## Tumor_size              0.15426   1.16680  0.14842  1.039 0.298651
## Lymph_node_status      -0.19691   0.82127  0.14651 -1.344 0.178959
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                        exp(coef) exp(-coef) lower .95 upper .95
## radius_mean              2.46351    0.40592 0.0023608 2570.7301
## texture_mean             1.97888    0.50534 0.9763595    4.0108
## perimeter_mean           0.61973    1.61361 0.0003517 1092.1306
## area_mean                1.75048    0.57127 0.0437579   70.0258
## smoothness_mean          0.69643    1.43588 0.3561535    1.3618
## compactness_mean         0.86899    1.15077 0.2306883    3.2734
## concavity_mean           1.84597    0.54172 0.5724264    5.9529
## concave_points_mean      0.42215    2.36881 0.1205488    1.4784
## symmetry_mean            0.53371    1.87369 0.3321609    0.8575
## fractal_dimension_mean   1.81770    0.55015 0.7478096    4.4183
## radius_SD                3.13271    0.31921 0.4206813   23.3285
## texture_SD               2.20516    0.45348 1.3955716    3.4844
## perimeter_SD             0.10783    9.27373 0.0178114    0.6528
## area_SD                  3.75403    0.26638 0.7333810   19.2162
## smoothness_SD            0.35042    2.85370 0.2079214    0.5906
## compactness_SD           0.33926    2.94756 0.1075100    1.0706
## concavity_SD             3.69607    0.27056 1.3432956   10.1697
```

```
## concave_points_SD         1.14098    0.87644 0.6175681    2.1080
## symmetry_SD               0.69921    1.43018 0.4399708    1.1112
## fractal_dimension_SD      1.68096    0.59490 0.6925861    4.0798
## radius_worst              0.05365   18.63935 0.0011233    2.5624
## texture_worst             0.60173    1.66187 0.2551734    1.4190
## perimeter_worst          66.07539    0.01513 2.8528101 1530.4057
## area_worst                0.12254    8.16038 0.0075567    1.9872
## smoothness_worst          1.85473    0.53916 0.9884250    3.4803
## compactness_worst         1.38883    0.72003 0.3832883    5.0324
## concavity_worst           0.49921    2.00316 0.1882260    1.3240
## concave_points_worst      1.29399    0.77280 0.7171299    2.3349
## symmetry_worst            1.75151    0.57093 0.8805794    3.4838
## fractal_dimension_worst   0.50023    1.99909 0.1663345    1.5044
## Tumor_size                1.16680    0.85705 0.8722806    1.5608
## Lymph_node_status         0.82127    1.21763 0.6162722    1.0944
##
## Concordance= 0.742  (se = 0.024 )
## Rsquare= 0.475   (max possible= 0.999 )
## Likelihood ratio test= 87.7  on 32 df,   p=4e-07
## Wald test            = 87.93  on 32 df,   p=4e-07
## Score (logrank) test = 110.7  on 32 df,   p=1e-10
```

la colonne *Coeff* représente les coefficients de la regression et la colonne *exp(coeff)* représente le risque proportionnel (hazard ratio)

On remarque que de nombreuses variables ne sont pas significatives car $p > 0.05$ pour un grand nombre de variables prises individuellement.

Voyons si nous pouvons, avec la fonction *stepAIC {MASS}* , améliorer notre modèle par minimisation de *l'AIC*

- Amélioration du modèle :

```r
cox_final = stepAIC(cox_fit,trace = F,direction = "backward")
summary(cox_final)
```

```
## Call:
## coxph(formula = Surv(time, recurrent) ~ texture_mean + symmetry_mean +
##     texture_SD + perimeter_SD + area_SD + smoothness_SD + compactness_SD +
##     concavity_SD + perimeter_worst + area_worst, data = data_class_train_surv)
##
##   n= 136, number of events= 121
##
##                     coef exp(coef) se(coef)      z Pr(>|z|)
## texture_mean     0.31944   1.37635  0.11488  2.781 0.005426 **
## symmetry_mean   -0.35777   0.69923  0.11477 -3.117 0.001824 **
## texture_SD       0.44545   1.56120  0.12457  3.576 0.000349 ***
## perimeter_SD    -1.04705   0.35097  0.41042 -2.551 0.010736 *
## area_SD          1.54802   4.70217  0.52873  2.928 0.003413 **
## smoothness_SD   -0.48669   0.61466  0.14216 -3.423 0.000618 ***
## compactness_SD  -0.50627   0.60274  0.16577 -3.054 0.002257 **
## concavity_SD     0.72643   2.06768  0.20539  3.537 0.000405 ***
## perimeter_worst  2.41737  11.21634  0.68960  3.505 0.000456 ***
## area_worst      -2.82603   0.05925  0.78912 -3.581 0.000342 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
##                 exp(coef) exp(-coef) lower .95 upper .95
## texture_mean      1.37635    0.72656   1.09886    1.7239
## symmetry_mean     0.69923    1.43014   0.55838    0.8756
## texture_SD        1.56120    0.64053   1.22299    1.9929
## perimeter_SD      0.35097    2.84924   0.15701    0.7845
## area_SD           4.70217    0.21267   1.66818   13.2542
## smoothness_SD     0.61466    1.62691   0.46519    0.8122
## compactness_SD    0.60274    1.65910   0.43554    0.8341
## concavity_SD      2.06768    0.48363   1.38247    3.0925
## perimeter_worst  11.21634    0.08916   2.90309   43.3353
## area_worst        0.05925   16.87824   0.01262    0.2782
##
## Concordance= 0.71  (se = 0.028 )
## Rsquare= 0.398   (max possible= 0.999 )
## Likelihood ratio test= 68.97  on 10 df,   p=7e-11
## Wald test            = 71.47  on 10 df,   p=2e-11
## Score (logrank) test = 74.17  on 10 df,   p=7e-12
```

```
cox_final
```

```
## Call:
## coxph(formula = Surv(time, recurrent) ~ texture_mean + symmetry_mean +
##     texture_SD + perimeter_SD + area_SD + smoothness_SD + compactness_SD +
##     concavity_SD + perimeter_worst + area_worst, data = data_class_train_surv)
##
##                     coef exp(coef) se(coef)      z        p
## texture_mean     0.31944   1.37635  0.11488  2.781 0.005426
## symmetry_mean   -0.35777   0.69923  0.11477 -3.117 0.001824
## texture_SD       0.44545   1.56120  0.12457  3.576 0.000349
## perimeter_SD    -1.04705   0.35097  0.41042 -2.551 0.010736
## area_SD          1.54802   4.70217  0.52873  2.928 0.003413
## smoothness_SD   -0.48669   0.61466  0.14216 -3.423 0.000618
## compactness_SD  -0.50627   0.60274  0.16577 -3.054 0.002257
## concavity_SD     0.72643   2.06768  0.20539  3.537 0.000405
## perimeter_worst  2.41737  11.21634  0.68960  3.505 0.000456
## area_worst      -2.82603   0.05925  0.78912 -3.581 0.000342
##
## Likelihood ratio test=68.97  on 10 df, p=7.015e-11
## n= 136, number of events= 121
```

En utilisant la méthode de sélection de variables backward, les variables explicatives qui expliquent le mieux notre modèle sont : texture_mean, symmetry_mean, texture_SD, perimeter_SD, area_SD, smoothness_SD, compactness_SD, concavity_SD , perimeter_worst, area_worst.

On remarque bien que y a une nette amélioration au niveau des valeurs significatives et la p-value est passé de $p = 4e - 07$ à $7.015e - 11$ pour le test du ratio du maximum de vraissamblace.

- Représentation des rapports de risque

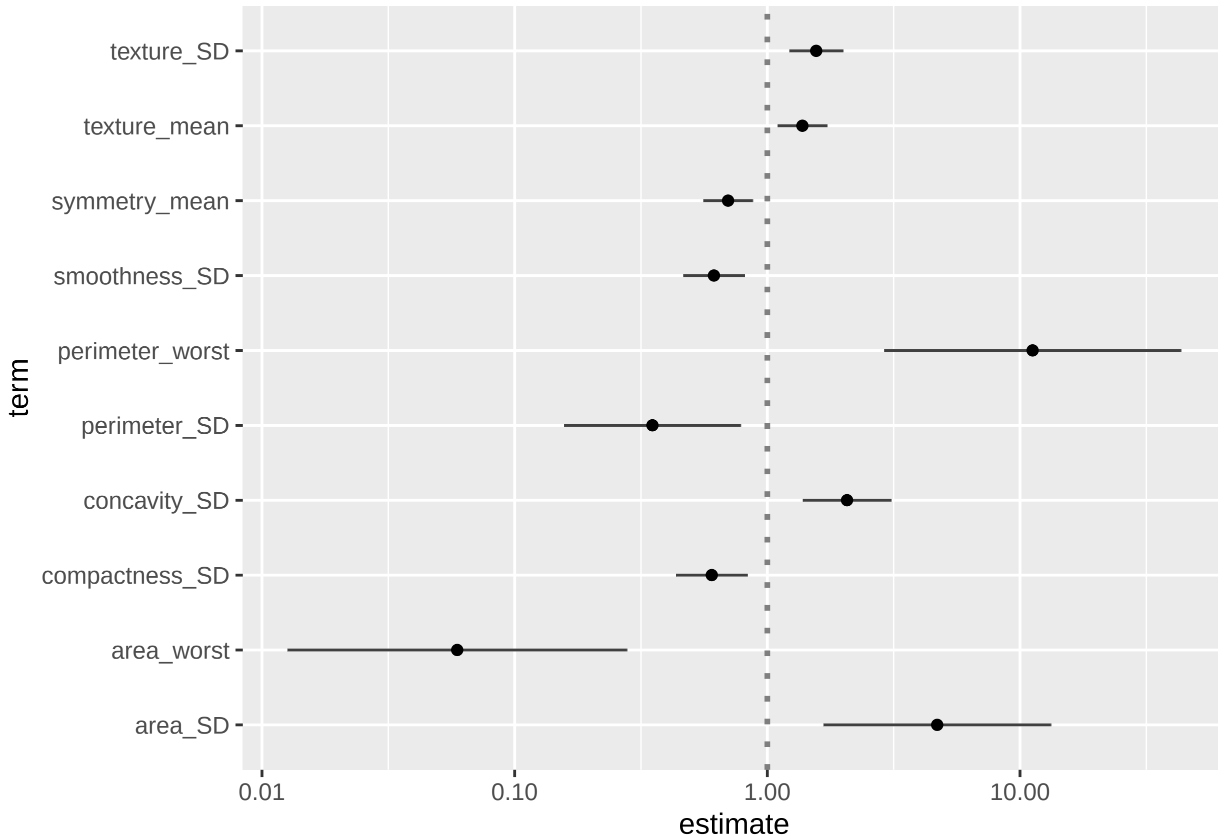
```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
##
## Attaching package: 'GGally'
```

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

```
ggcoef(cox_final, exponentiate = TRUE)
```



Par exemple pour les variables *perimeter_worst* et *area_SD* plus leurs valeurs augmentes plus le risque de rechuter augmente , contrairement à, *perimeter_SD* et *area_worst* qui plus leurs valeurs diminues plus le risque de rechuter diminue

- Prédiction sur le *test*

```
pred_cox = survfit(cox_final)
pred_cox$surv[24]
```

```
## [1] 0.7540666
```

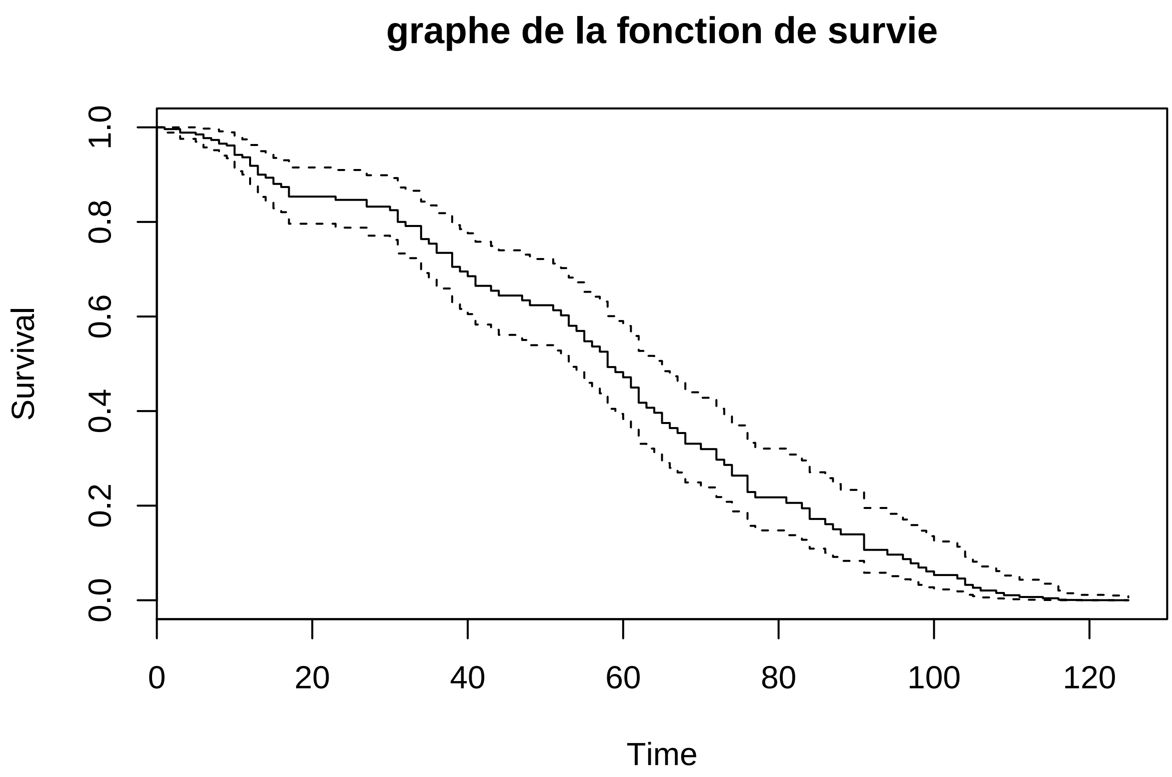
La fonction *survfit* appliquée au modèle de Cox renvoie: les valeurs de la fonction de survie conditionnelle estimée aux différents temps d'observation *survival*, la valeur de chaque covariable étant par défaut égale à la valeur moyenne de la covariable *std.err* et les intervalles de confiances à 95% *lower 95% CI* et *upper 95% CI*

Par exemple pour le temps time = 24mois la probabilité de rechute est égale à 0.754

- Représentation du graphe de la fonction de survie

On peut lire le résultat précédent sur le graphe suivant.

```
plot(pred_cox, xlab = "Time", ylab="Survival", ylim = c(0,1), main="graphe de la fonction de survie" )
```



- Mesure de performance

```
pred_total =predict(cox_final)        #on effectue une prediction sur le data au complet
pred_test=predict(cox_final,data_class_test_surv,type='risk')     #on effectue une prediction que sur le
Surv.rsp <- Surv(data_class_train_surv$time, data_class_train_surv$recurrent)
Surv.rsp.new <- Surv(data_class_train_surv$time, data_class_train_surv$recurrent)
```

```
accuracies <- rep(0,5)  #liste qui contient les accuracy des algorithmes
algorithmes <- rep('algo',5)  #liste qui contient les noms des algorithmes
```

```
library(survAUC)
times <- seq(1,130,1)
AUC_CD <- AUC.cd(Surv.rsp, Surv.rsp.new,pred_total, pred_test, times)
accuracies[1]<-AUC_CD$iauc * 100
algorithmes[1]<-'Cox'
AUC_CD$iauc
```

```
## [1] 0.7798892
```

On a une accuracy de 0.7798892.

Survival Random forest :

Le modèle de survival random forest se calcule avec *ranger {ranger}*

- Entrainement sur le _train:

```
library(ranger)

ranger_model <- ranger(Surv(data_class_train_surv$time,data_class_train_surv$recurrent) ~.-id-id_1n,data
# affiche les coefficients
sort(ranger_model$variable.importance)
```

```
##                 area_SD          smoothness_SD      compactness_worst
##           -0.0015285909          -0.0011411710          -0.0011105872
##               radius_SD             Tumor_size           radius_worst
##           -0.0008288645          -0.0008089384          -0.0006808865
##              area_worst           perimeter_SD        perimeter_worst
##           -0.0006657882          -0.0002534700           0.0001887592
##          compactness_SD        smoothness_mean       compactness_mean
##            0.0003152333           0.0003315968           0.0003781597
##     concave_points_mean  fractal_dimension_mean      Lymph_node_status
##            0.0004525664           0.0005186406           0.0006152788
## fractal_dimension_worst      concave_points_SD            radius_mean
##            0.0006887979           0.0006994666           0.0007038067
##             symmetry_SD   concave_points_worst          symmetry_mean
##            0.0007233903           0.0010305070           0.0010430871
##    fractal_dimension_SD         perimeter_mean       smoothness_worst
##            0.0013425524           0.0015217586           0.0016165103
##               area_mean           concavity_SD        concavity_worst
##            0.0018438870           0.0021319584           0.0021596164
##          symmetry_worst             texture_SD         concavity_mean
##            0.0026180549           0.0033505028           0.0034139112
##           texture_worst           texture_mean
##            0.0076295530           0.0104087258
```

On observe que la variable la plus importante est concavity_mean et area_SD est la moins importante dans notre modèle.

```r
sapply(data.frame(ranger_model$survival),mean)[24]
```

```
##      X24
## 0.664807
```

La probabilité de rechute à 24 mois est égale à 0.6648

- Prédiction sur le test :

```r
pred_rf=predict(ranger_model,data_class_test_surv)
```

- AUC

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
roc(response=((data_class_test_surv$recurrent)), predictor=1 - pred_rf$survival[,24])
```

```
## Setting levels: control = FALSE, case = TRUE

## Setting direction: controls < cases

##
## Call:
## roc.default(response = ((data_class_test_surv$recurrent)), predictor = 1 -     pred_rf$survival[, 24]
##
```

```
## Data: 1 - pred_rf$survival[, 24] in 3 controls (((data_class_test_surv$recurrent)) FALSE) < 30 cases
## Area under the curve: 0.7889
```

L'AUC sur le test à 24 mois est de 0.7889

```
accuracies[2]<-0.7889 *100
algorithmes[2]<-'randf_surv'
```

Classification :

Regression logistique:

```
library(caret)
control <- trainControl(method="repeatedcv", number=5, repeats=3)
fit.glm <- train(Z~.-id-id_1n, data=data_class_train, method="glm",trControl=control, metric="Accuracy")
```

- Prédiction sur le **test** :

```
pred_glm = predict(fit.glm, newdata = data_class_test)
```

- Matrice de confusion et accuracy **Régression Logistique**:

```
MC_glm <- table(`Predicted Class`=pred_glm,`Actual Class`=data_class_test$Z)
print(MC_glm)
```

```
##                  Actual Class
## Predicted Class  0  1
##                0 19  0
##                1  3 11
```

```
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
acc_glm = accuracy(MC_glm)
acc_glm
```

```
## [1] 90.90909
```

On remarque une accuracy sur le *test* de 91%

```
accuracies[3]<-acc_glm
algorithmes[3]<-'reglog'
```

- ROC :

```
library(pROC)
pROC_obj <- roc(data_class_test$Z,as.numeric(pred_glm),
            smoothed = TRUE,
            # arguments for ci
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            # arguments for plot
            plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
            print.auc=TRUE, show.thres=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

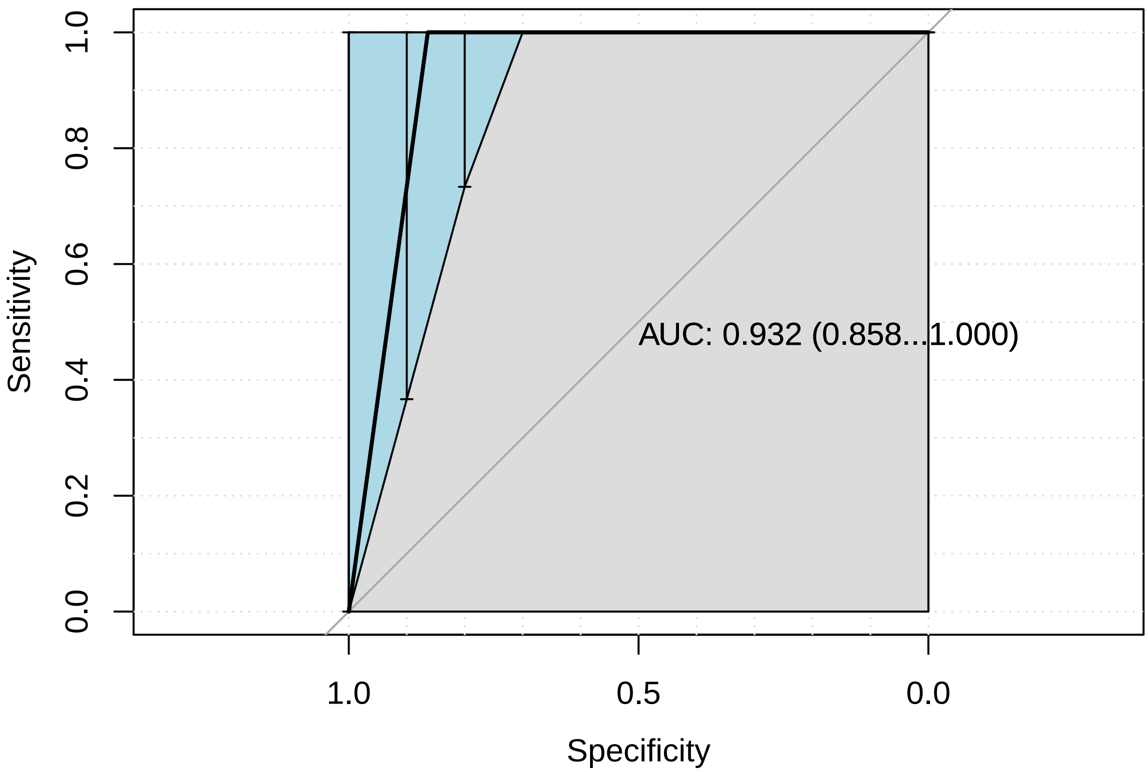
```
## Setting direction: controls < cases
```

```
sens.ci <- ci.se(pROC_obj)
plot(sens.ci, type="shape", col="lightblue")
```

```
## Warning in plot.ci.se(sens.ci, type = "shape", col = "lightblue"): Low
```

```
## definition shape.
```

```r
plot(sens.ci, type="bars")
```



Random Forest :

```r
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ranger':
##
##     importance

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

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

```r
set.seed(15)
data_class_train$Z=as.factor(data_class_train$Z)
rf.fit <- randomForest(Z~.-id-id_1n, data=data_class_train)
```

- Prédiction sur le **test**:

```r
pred_rf = predict(rf.fit, newdata = data_class_test)
```

- Matrice de confusion et accuracy **Random Forest**:

```r
MC_rf <- table(`Predicted Class`=pred_rf,`Actual Class`=data_class_test$Z)
```

```r
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
acc_rf=accuracy(MC_rf)
acc_rf
```

```
## [1] 96.9697
```

On a une accuracy sur le **test** de 97%

```r
accuracies[4]<-acc_rf
algorithmes[4]<-'Randfor_class'
```

- ROC:

```r
pROC_obj <- roc(data_class_test$Z,as.numeric(pred_rf),
            smoothed = TRUE,
            # arguments for ci
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            # arguments for plot
            plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
            print.auc=TRUE, show.thres=TRUE)
```

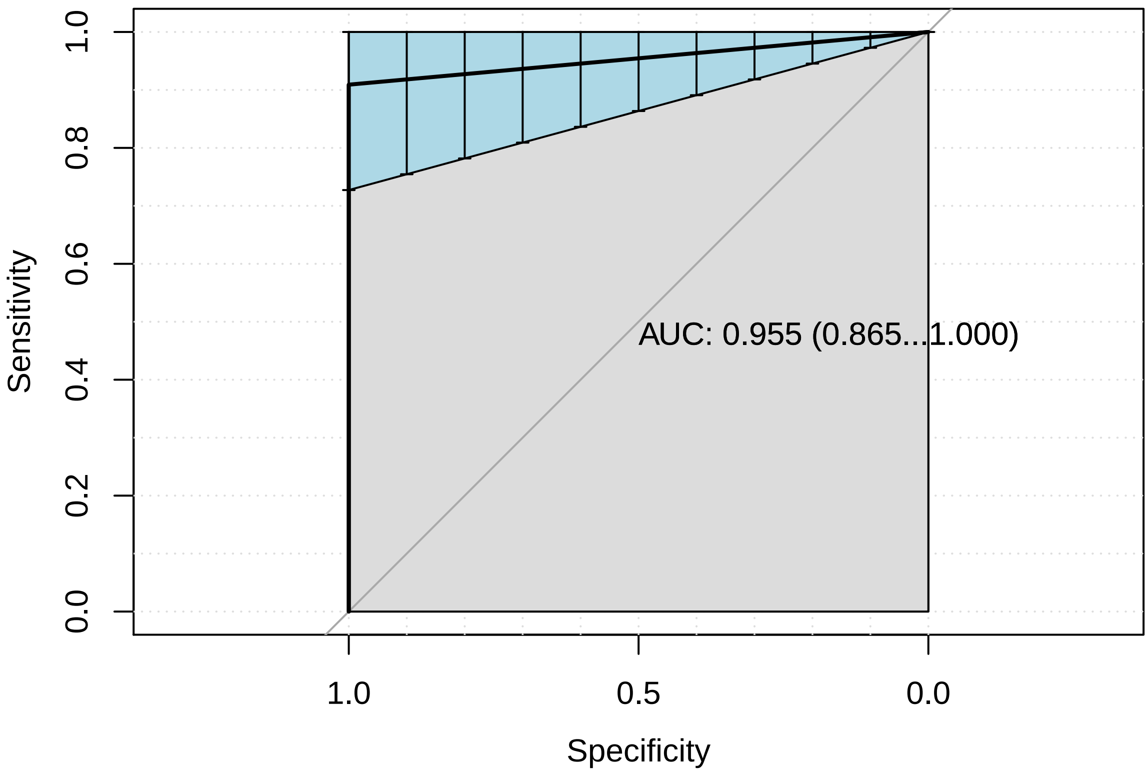
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
sens.ci <- ci.se(pROC_obj)
plot(sens.ci, type="shape", col="lightblue")
```

```
## Warning in plot.ci.se(sens.ci, type = "shape", col = "lightblue"): Low
## definition shape.
```

```r
plot(sens.ci, type="bars")
```



Naive Bayes:

- Entrainement sur le *train*:

```
library(e1071)
set.seed(15)
fit.NB <- naiveBayes(Z~.-id-id_1n , data = data_class_train,metric="accuracy")
```

- Prédiction sur le **test**:

```
pred_NB = predict(fit.NB, newdata = data_class_test)
```

- Accuracy **Naïve Bayes** :

```
MC_NB <- table(`Predicted Class`=pred_NB,`Actual Class`=data_class_test$Z)
acc_BN <-accuracy(MC_NB)
print(acc_BN)
```

```
## [1] 75.75758
```

On a une accuracy sur le **test** d'environ 76%

```
accuracies[5]<-acc_BN
algorithmes[5]<-'NaiveBayes'
```

```
pROC_obj <- roc(data_class_test$Z,as.numeric(pred_NB),
            smoothed = TRUE,
            # arguments for ci
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            # arguments for plot
            plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
            print.auc=TRUE, show.thres=TRUE)
```

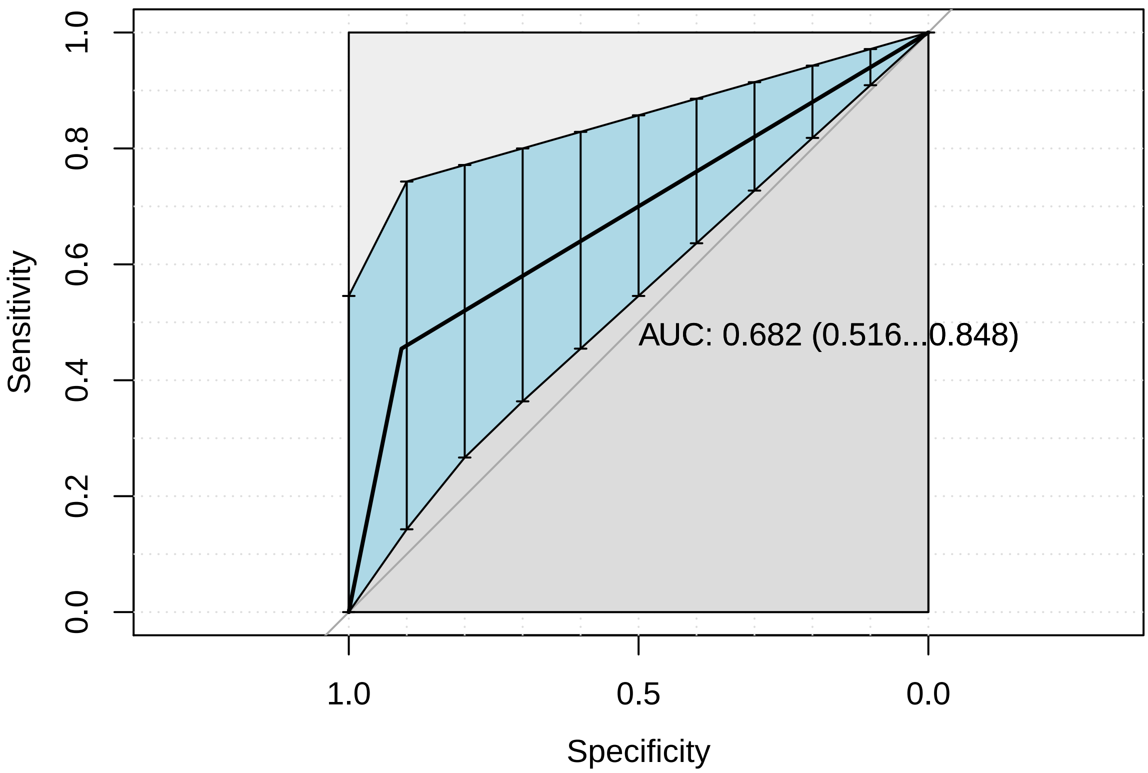
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
sens.ci <- ci.se(pROC_obj)
plot(sens.ci, type="shape", col="lightblue")
```

```
## Warning in plot.ci.se(sens.ci, type = "shape", col = "lightblue"): Low
## definition shape.
```

```r
plot(sens.ci, type="bars")
```

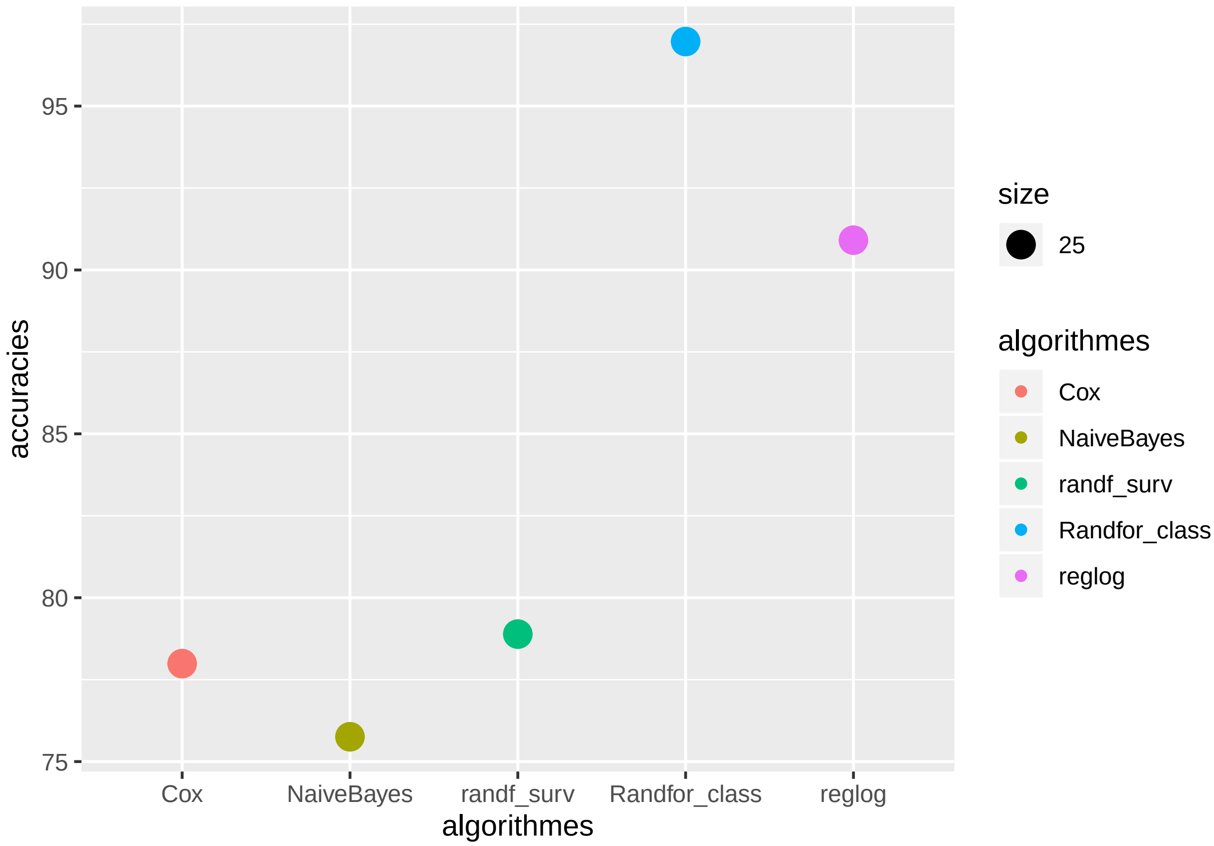


Comparaison des algorithmes:

- Graphe de toutes les accuracy:

On représente les accuracy de chaque algorithme en pourcentage.

```r
df_algo <- data.frame(accuracies,algorithmes)
ggplot(df_algo, aes(x=algorithmes, y=accuracies, group=algorithmes)) +
  geom_point(aes( color=algorithmes,size=25))
```



Conclusion :

En comparant les accuracies des différents modèles, on remarque que le meilleur modèle pour la prédiction est le modèle de random forest de classification.