A study of Breast Cancer Wisconsin with Machine Learning & Survival analysis

Technical Report · November 2019

DOI: 10.13140/RG.2.2.13350.27203

3 authors, including:

Ayoub Abraich
Université Paris-Saclay
7 PUBLICATIONS **1** CITATION
SEE PROFILE

El Mehdi Bouchouat
Université Paris-Saclay
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

A study of Breast Cancer Wisconsin with Machine Learning & Survival analysis

Ayoub Abraich **El Mehdi Bouchouat** **Mohamed Tounsi**

Introduction

Breast cancer is the most common malignancy among women, accounting for nearly 1 in 3 cancers diagnosed among women in the United States, and it is the second leading cause of cancer death among women. Breast Cancer occurs as a result of abnormal growth of cells in the breast tissue, commonly referred to as a Tumor. A tumor does not mean cancer - tumors can be benign (not cancerous), pre-malignant (pre-cancerous), or malignant (cancerous). Tests such as MRI, mammogram, ultrasound and biopsy are commonly used to diagnose breast cancer performed.

The data set "Wisconsin Prognostic Breast Cancer (WPBC)" contains "reccurent" which has 2 levels "R = Reccurent" or "N=Non-recurrent" and 30 features discribing cell nuclei in the numerized picture. Ten real features are calculated for each nuclei : - radius : (mean of distances from center to points on the perimeter); - texture : (standard deviation of gray-scale values) ;
- perimeter; -area; - smoothness : (local variation in radius lengths) - compactness : (perimeter ^ 2 / area - 1,0); - concavity : (severity of concave portions of the contour) ; - concave points : (number of concave portions of the contour) ; - symmetry; - fractal dimension : ("coastline approximation" - 1).

The mean, the standard error(SE) and the worst or the biggest value (mean of the 3 values) of this features were calculated for each pictures, that is to say 30 features. We are gonna analyze the features to determine the prediction of diagnostic. This will allow us to predict the probability of realpse at 24 month. For that we have tried different methods (Survival Models, Classification with ML) by comparing the models.

Plan

- Survival Analysis :
  - Kaplan Meier Survival Curve
  - Cox model
  - Random survival forests
  - Results : comparison
- Classification with ML :
  - Objective
  - Data analysis : PCA & LDA
  - Application of ML methods
  - Results : comparison
- Conclusion : ML vs Survival models

Survival Analysis

Data preprocessing

Importation & transformation of data

To start we import the data, then we impute the missing data of the variable "Lymph.node.status" with its average. The data is then normalized by binary recoding the "recurrent".

```
get_data <- function(){
  wpbc = read_delim("wpbc.data.txt" , delim="," , col_names=F,  na = '?')

  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'))

  wpbc = wpbc %>% mutate(id = factor(id)) %>%
                  mutate( recurrent = recode_factor(recurrent , "N" = FALSE, 'R' = TRUE ))
  wpbc_sansNA = wpbc %>% replace_na(list(Lymph_node_status = median(wpbc$Lymph.node.status),
                                         na.RM = T))
  scale2 <- function(x)(x-mean(x,na.rm = T))/sd(x,na.rm = T)
  wpbc_stand = wpbc_sansNA%>% mutate_at(names(wpbc)[-c(1,2,3)] , scale2)
  wpbc_stand$time  <- as.numeric(wpbc_stand$time )
   return(na.omit(wpbc_stand))
}
```

We create 2 versions of data, the first for the survival model and the second for the machine learning

```
get_data_cox <- function(){
  data=get_data()
  data$recurrent  <- as.numeric(data$recurrent )
  return(data)
}
```

Data for survival analysis :

Kaplan Meier Survival Curve

- 'survfit' function creates survival curves from either a formula (e.g. the Kaplan-Meier), a previously fitted Cox model, or a previously fitted accelerated failure time model.
- 'Surv' function create a survival object, usually used as a response variable in a model formula.

```
km_fit <- survfit(Surv(time, recurrent) ~ 1, data=data_cox)
```

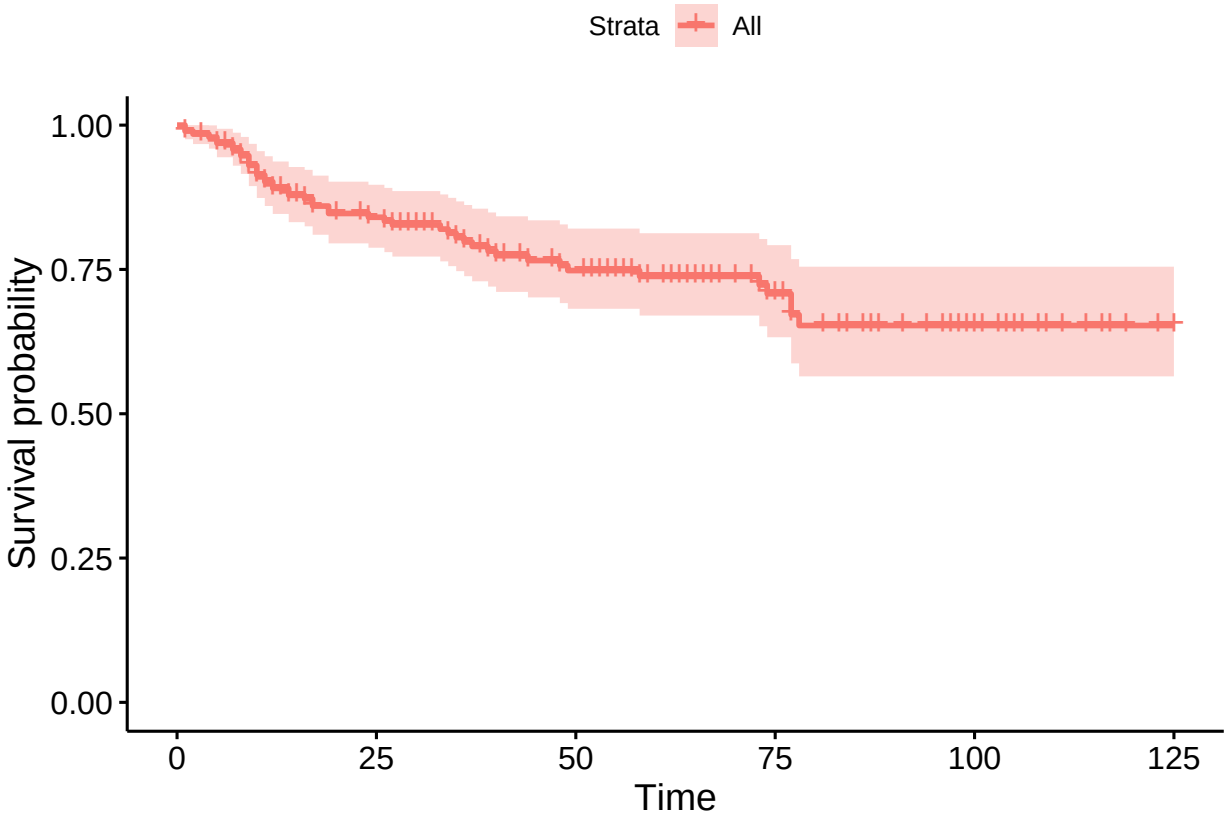
```
summary(km_fit,time=24)
```

```
## Call: survfit(formula = Surv(time, recurrent) ~ 1, data = data_cox)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##    24    129      28     0.84  0.0278        0.788        0.897
```

The probability of relapse at 24 months is 1-0.84 = 0.16 with a confidence interval [0.103; 0212].

```
ggsurvplot(km_fit, data = data_cox, pval = TRUE)
```

```
## Warning in .pvalue(fit, data = data, method = method, pval = pval, pval.coord = pval.coord, : There a
##  This is a null model.
```



Cox model

We code a function that gives us the formula: Surv (time, recurrent) ~ "sum of variables"

```
formulacox<-function(){
  x=names(data_cox)
  x=x[!x %in% c("id","recurrent","time" )]
  f=formula(paste("Surv(time,recurrent)~", paste0(x, collapse = " + ")))
  return(f)
}
formulacox()
```

```
## Surv(time, recurrent) ~ radius_mean + texture_mean + perimeter_mean +
##     area_mean + smoothness_mean + compactness_mean + concavity_mean +
##     concave.points_mean + symmetry_mean + fractal.dimension_mean +
##     radius_SD + texture_SD + perimeter_SD + area_SD + smoothness_SD +
##     compactness_SD + concavity_SD + concave.points_SD + symmetry_SD +
##     fractal.dimension_SD + radius_worst + texture_worst + perimeter_worst +
```

```
##     area_worst + smoothness_worst + compactness_worst + concavity_worst +
##     concave.points_worst + symmetry_worst + fractal.dimension_worst +
##     Tumor.size + Lymph.node.status
## <environment: 0x56438bd1f6e8>
```

We train the Cox model with all the variables.

We choose then the most relevant model, i.e the one that minimizes the AIC.

```
#stepAIC(cox_model_all)
```

```
model_cox <- coxph(formula = Surv(time, recurrent) ~ radius_mean + perimeter_mean +
    area_mean + smoothness_mean + concavity_mean + fractal.dimension_mean +
    compactness_SD + concavity_SD + radius_worst + area_worst +
    compactness_worst + concavity_worst + Lymph.node.status,
    data = data_cox)
cox_fit <-survfit(model_cox)
```

```
summary(model_cox)
```

```
## Call:
## coxph(formula = Surv(time, recurrent) ~ radius_mean + perimeter_mean +
##     area_mean + smoothness_mean + concavity_mean + fractal.dimension_mean +
##     compactness_SD + concavity_SD + radius_worst + area_worst +
##     compactness_worst + concavity_worst + Lymph.node.status,
##     data = data_cox)
##
##   n= 194, number of events= 46
##
##                              coef  exp(coef)   se(coef)      z Pr(>|z|)
## radius_mean            -1.452e+01  4.922e-07  4.199e+00 -3.459 0.000543
## perimeter_mean          1.042e+01  3.356e+04  4.225e+00  2.466 0.013651
## area_mean               3.826e+00  4.586e+01  1.960e+00  1.952 0.050995
## smoothness_mean         9.031e-01  2.467e+00  3.968e-01  2.276 0.022869
## concavity_mean         -1.402e+00  2.460e-01  8.274e-01 -1.695 0.090102
## fractal.dimension_mean -1.284e+00  2.770e-01  4.210e-01 -3.049 0.002292
## compactness_SD          1.425e+00  4.159e+00  4.577e-01  3.114 0.001846
## concavity_SD           -1.016e+00  3.619e-01  4.703e-01 -2.161 0.030682
## radius_worst            4.116e+00  6.130e+01  1.968e+00  2.091 0.036512
## area_worst             -3.184e+00  4.144e-02  1.803e+00 -1.766 0.077385
## compactness_worst      -1.528e+00  2.170e-01  5.441e-01 -2.808 0.004990
## concavity_worst         1.652e+00  5.218e+00  5.688e-01  2.904 0.003679
## Lymph.node.status       4.132e-01  1.512e+00  1.317e-01  3.138 0.001701
##
## radius_mean            ***
## perimeter_mean         *
## area_mean              .
## smoothness_mean        *
## concavity_mean         .
## fractal.dimension_mean **
## compactness_SD         **
## concavity_SD           *
## radius_worst           *
```

```
## area_worst             .
## compactness_worst      **
## concavity_worst        **
## Lymph.node.status      **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                         exp(coef) exp(-coef) lower .95 upper .95
## radius_mean             4.922e-07  2.032e+06 1.311e-10 1.847e-03
## perimeter_mean          3.356e+04  2.980e-05 8.496e+00 1.326e+08
## area_mean               4.586e+01  2.180e-02 9.836e-01 2.139e+03
## smoothness_mean         2.467e+00  4.053e-01 1.133e+00 5.370e+00
## concavity_mean          2.460e-01  4.065e+00 4.860e-02 1.245e+00
## fractal.dimension_mean 2.770e-01  3.610e+00 1.214e-01 6.321e-01
## compactness_SD          4.159e+00  2.404e-01 1.696e+00 1.020e+01
## concavity_SD            3.619e-01  2.763e+00 1.440e-01 9.097e-01
## radius_worst            6.130e+01  1.631e-02 1.295e+00 2.902e+03
## area_worst              4.144e-02  2.413e+01 1.211e-03 1.418e+00
## compactness_worst       2.170e-01  4.607e+00 7.471e-02 6.305e-01
## concavity_worst         5.218e+00  1.916e-01 1.711e+00 1.591e+01
## Lymph.node.status       1.512e+00  6.615e-01 1.168e+00 1.957e+00
##
## Concordance= 0.764  (se = 0.037 )
## Likelihood ratio test= 48.94  on 13 df,   p=5e-06
## Wald test            = 47.61  on 13 df,   p=8e-06
## Score (logrank) test = 54.02  on 13 df,   p=6e-07
```

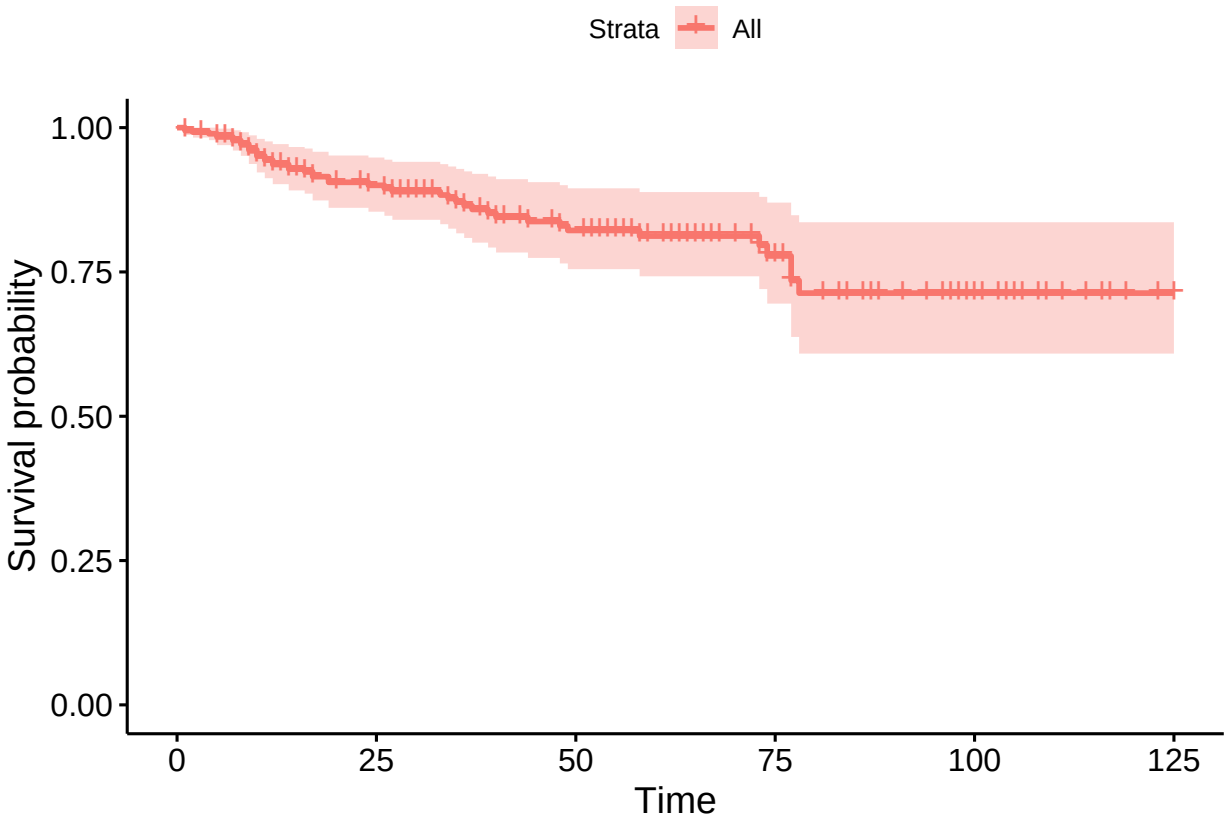
```
summary(cox_fit,time=24)
```

```
## Call: survfit(formula = model_cox)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##    24    129      28      0.9  0.0239        0.854        0.948
```

The probability of relapse at 24 months is 1-0.9 = 0.1 with a confidence interval [0.052; 0146]

```
ggsurvplot(cox_fit, data = data_cox, pval = TRUE)
```

```
## Warning in .pvalue(fit, data = data, method = method, pval = pval, pval.coord = pval.coord, : There a
##  This is a null model.
```



Random survival forests

We train the ranger model:

```
r_fit <- ranger(formulacox(),
                      data = data_cox,
                      mtry = 4,
                      importance = "permutation",
                      splitrule = "extratrees",
                      verbose = TRUE)
```

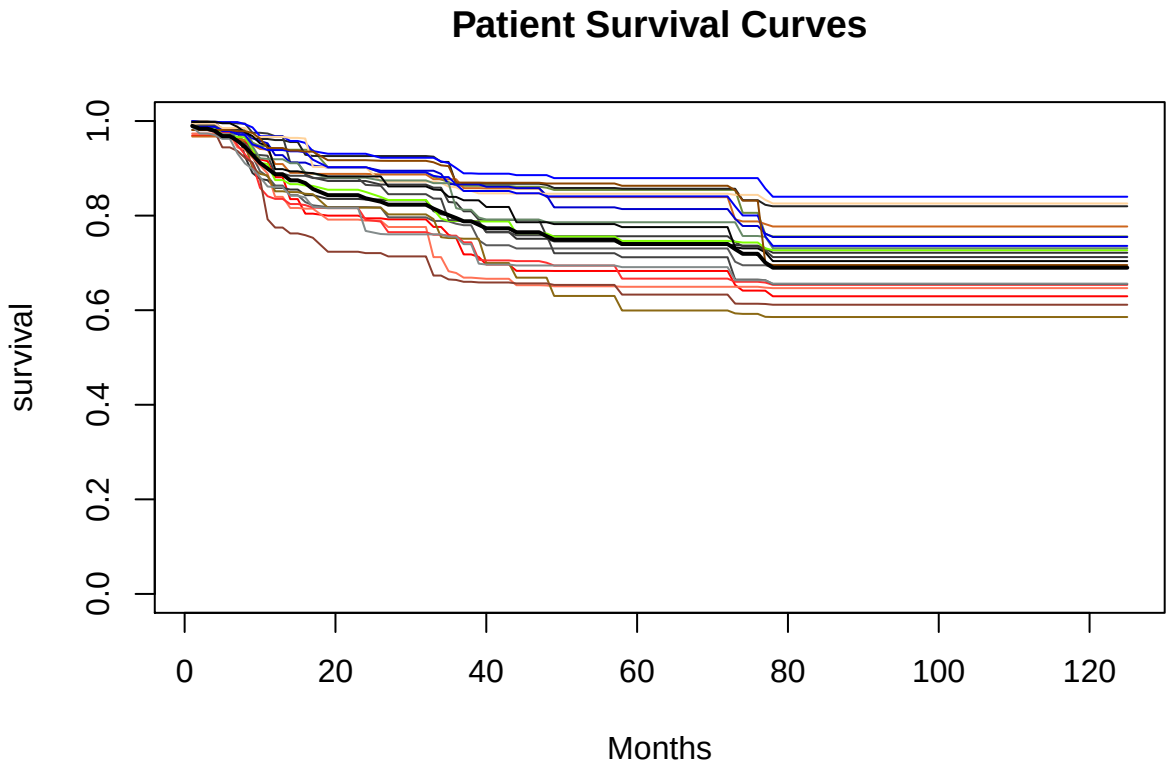
Average the survival models

```
death_times <- r_fit$unique.death.times
surv_prob <- data.frame(r_fit$survival)
avg_prob <- sapply(surv_prob,mean)
```

Plot the survival models for each patient:

```
N=20 ## nombre of patients


plot(r_fit$unique.death.times,r_fit$survival[1,],
     type = "l",
     ylim = c(0,1),
     col = "red",
```

```
     xlab = "Months",
     ylab = "survival",
     main = "Patient Survival Curves")



#
cols <- colors()

for (n in sample(c(2:dim(data_cox)[1]), N)){
  lines(r_fit$unique.death.times, r_fit$survival[n,], type = "l", col = cols[n])
}
lines(death_times, avg_prob, lwd = 2)
legend(500, 0.7, legend = c('Average = black'))
```



Probability of relapse for this model at 24 months of a patient n:

```
n=20
1-(r_fit$survival[n,][24])
```

```
## [1] 0.06879598
```

Average relapse probability for this model at 24 months:

```
 pr_mean<- 1-mean(sapply(1:dim(data_cox)[1], function(n) r_fit$survival[n,][24]))
 pr_mean
```

```
## [1] 0.1768696
```

The next block of code illustrates how ranger() ranks variable importance.

```
vi <- data.frame(sort(round(r_fit$variable.importance, 4), decreasing = TRUE))
names(vi) <- "importance"
head(vi)
```

```
##                   importance
## radius_worst          0.0064
## Lymph.node.status     0.0062
## perimeter_worst       0.0052
## area_SD               0.0045
## symmetry_mean         0.0037
## Tumor.size            0.0036
```

Notice that ranger() flags 'Lymph.node.status','radius_worst','Tumor.size', 'area_mean','area_worst' and 'perimeter_worst' as the most important; the same variables with the smallest p-values in the Cox model. ranger() does compute Harrell's c-index (See [8] p. 370 for the definition), which is similar to the Concordance statistic described above. This is a generalization of the ROC curve, which reduces to the Wilcoxon-Mann-Whitney statistic for binary variables, which in turn, is equivalent to computing the area under the ROC curve.

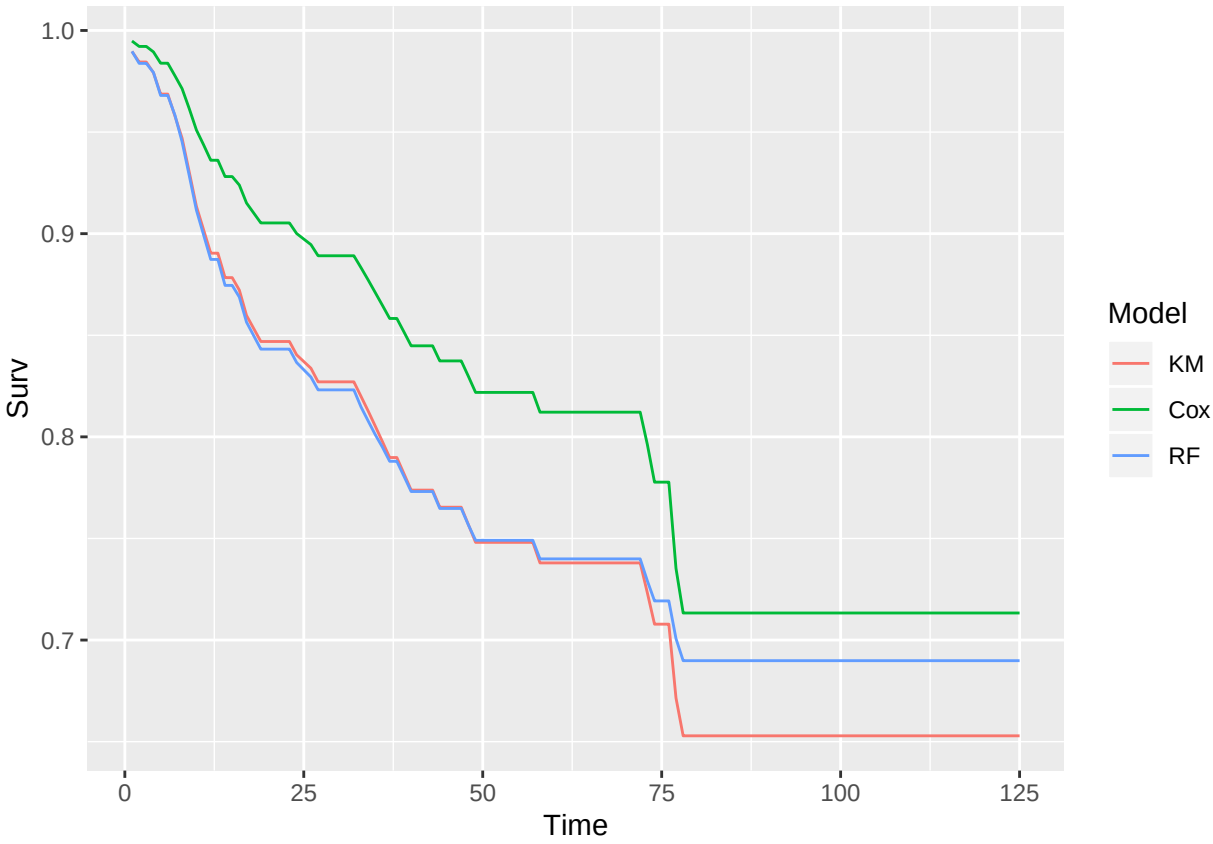
```
cat("Prediction Error = 1 - Harrell's c-index = ", r_fit$prediction.error)
```

```
## Prediction Error = 1 - Harrell's c-index =  0.3775947
```

Comparaison of the three survival models :

Finally, to provide an "eyeball comparison" of the three survival curves, we plot them on the same graph.The following code pulls out the survival data from the three model objects and puts them into a data frame for ggplot().

```
kmi <- rep("KM",length(km_fit$time))
km_df <- data.frame(km_fit$time,km_fit$surv,kmi)
names(km_df) <- c("Time","Surv","Model")

coxi <- rep("Cox",length(cox_fit$time))
cox_df <- data.frame(cox_fit$time,cox_fit$surv,coxi)
names(cox_df) <- c("Time","Surv","Model")

rfi <- rep("RF",length(r_fit$unique.death.times))
rf_df <- data.frame(r_fit$unique.death.times,avg_prob,rfi)
names(rf_df) <- c("Time","Surv","Model")

plot_df <- rbind(km_df,cox_df,rf_df)

p <- ggplot(plot_df, aes(x = Time, y = Surv, color = Model))
p + geom_line()
```



We compare also the average relapse probability for 3 models at 24 months: :

```
prob_rechute_list_surv <- list(KM=1-km_fit$surv[24],COX=1-cox_fit$surv[24]
            ,RSF= 1-mean(sapply(1:dim(data_cox)[1], function(n) r_fit$survival[n,][24])))

surv_prob_report <- data.frame(prob_rechute_mean=prob_rechute_list_surv)

surv_prob_report
```

```
##   prob_rechute_mean.KM prob_rechute_mean.COX prob_rechute_mean.RSF
## 1            0.1729175             0.1108799             0.1768696
```

Classification with ML :

Objective

In machine learning binary classification we try to fit a function f that predict the probability $P(Y = 1 \mid X = x) := f(x)$ given that X are features and Y are labels . In our problem , we have $Y = 1_{\text{Recurrent=True , Time } \leq 24} := 1_A$. So : $\{Y = 1\} = A$. Therfore : $p_{A,X} := P(A \mid X) = P(Y = 1 \mid X)$. Our goal is predicting $p_{A,X}$. We note in the following code the class $\{Y = 1\}$ by M , $\{Y = 0\}$ by B and Y by outcome_classif.

Data Importation

We import the data for the machine learning, and we code a variable outcome_classif in B and M.

```
get_data_ml <- function(){
  data=get_data()
  temp = data %>% dplyr::select(time,recurrent)%>%
           mutate(outcome_classif=
           ifelse((time <= 24)&(recurrent==TRUE),1,
           ifelse((time > 24)&(recurrent=TRUE),0,
           ifelse((time>24)&(recurrent=FALSE),0,NA))))
  data$recurrent<-NULL
  data$id<-NULL

  data$outcome_classif<-as.factor(temp$outcome_classif)

  return(na.omit(data))
}



dataml<-get_data_ml()
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   X2 = col_character()
## )

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

```
levels(dataml$outcome_classif) <- c("B", "M")
```

```
formulaml<-function(){
  x=names(dataml)
  x=x[!x %in% c("outcome_classif" )]
  f=formula(paste("outcome_classif~", paste0(x, collapse = " + ")))
  return(f)
}
formulaml()
```

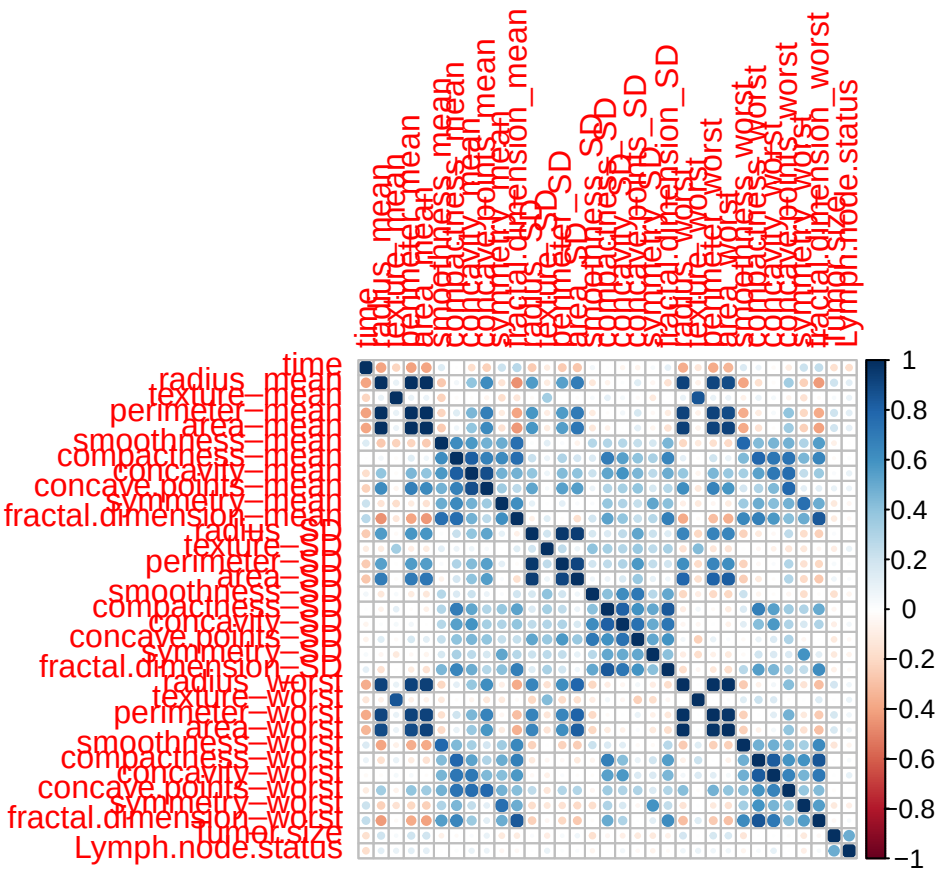
```
## outcome_classif ~ time + radius_mean + texture_mean + perimeter_mean +
##     area_mean + smoothness_mean + compactness_mean + concavity_mean +
##     concave.points_mean + symmetry_mean + fractal.dimension_mean +
##     radius_SD + texture_SD + perimeter_SD + area_SD + smoothness_SD +
##     compactness_SD + concavity_SD + concave.points_SD + symmetry_SD +
##     fractal.dimension_SD + radius_worst + texture_worst + perimeter_worst +
##     area_worst + smoothness_worst + compactness_worst + concavity_worst +
##     concave.points_worst + symmetry_worst + fractal.dimension_worst +
##     Tumor.size + Lymph.node.status
## <environment: 0x564395617bd0>
```

```
table(dataml$outcome_classif)
```

```
##
##   B   M
## 127  28
```

We notice that the data are unbalanced. This problem will be dealt with later.

```
corr_mat <- cor(dataml[, -ncol(dataml)])
corrplot(corr_mat,method = "circle")
```



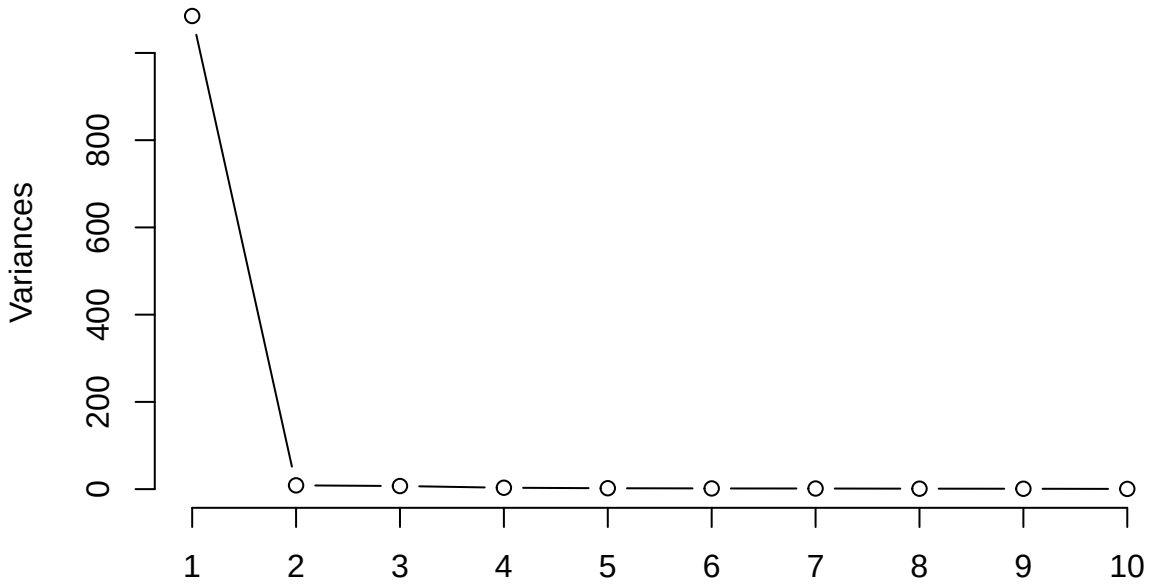
We note that "time" is almost uncorrelated with the other variables.

There is a great correlation between some variables.

Data preprocessing

Because there are so much correlation some machine learning models can fail. We are going to create a LDA version of the data ### PCA :

```
pca_res <- prcomp(dataml[,-ncol(dataml)])
plot(pca_res, type="l")
```

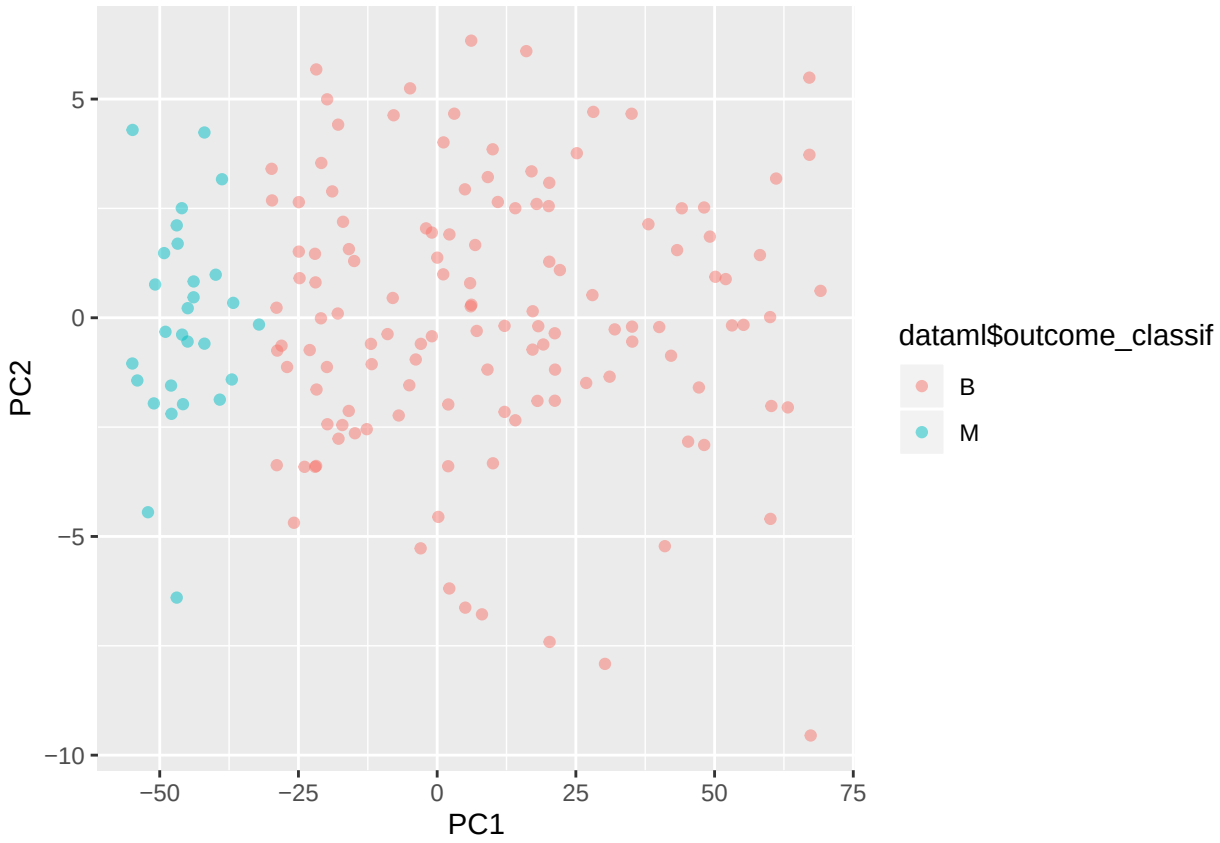



```
summary(pca_res)
```

```
## Importance of components:
##                             PC1     PC2    PC3     PC4     PC5     PC6
## Standard deviation     32.9336 2.94027 2.6706 1.79493 1.44571 1.26851
## Proportion of Variance  0.9738 0.00776 0.0064 0.00289 0.00188 0.00144
## Cumulative Proportion   0.9738 0.98160 0.9880 0.99090 0.99277 0.99422
##                             PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     1.19036 1.03906 0.93796 0.72466 0.69591 0.64601
## Proportion of Variance 0.00127 0.00097 0.00079 0.00047 0.00043 0.00037
## Cumulative Proportion  0.99549 0.99646 0.99725 0.99772 0.99815 0.99853
##                            PC13    PC14   PC15    PC16    PC17    PC18
## Standard deviation     0.59293 0.52763 0.4720 0.43934 0.35894 0.29936
## Proportion of Variance 0.00032 0.00025 0.0002 0.00017 0.00012 0.00008
## Cumulative Proportion  0.99884 0.99909 0.9993 0.99947 0.99958 0.99966
##                            PC19    PC20    PC21    PC22    PC23    PC24
## Standard deviation     0.29236 0.24933 0.21331 0.18727 0.18141 0.16772
## Proportion of Variance 0.00008 0.00006 0.00004 0.00003 0.00003 0.00003
## Cumulative Proportion  0.99974 0.99980 0.99984 0.99987 0.99990 0.99992
##                            PC25    PC26    PC27    PC28    PC29   PC30
## Standard deviation     0.14362 0.13210 0.12134 0.11595 0.09926 0.0721
## Proportion of Variance 0.00002 0.00002 0.00001 0.00001 0.00001 0.0000
## Cumulative Proportion  0.99994 0.99996 0.99997 0.99998 0.99999 1.0000
##                           PC31    PC32    PC33
## Standard deviation     0.0481 0.03221 0.01634
## Proportion of Variance 0.0000 0.00000 0.00000
## Cumulative Proportion  1.0000 1.00000 1.00000
```

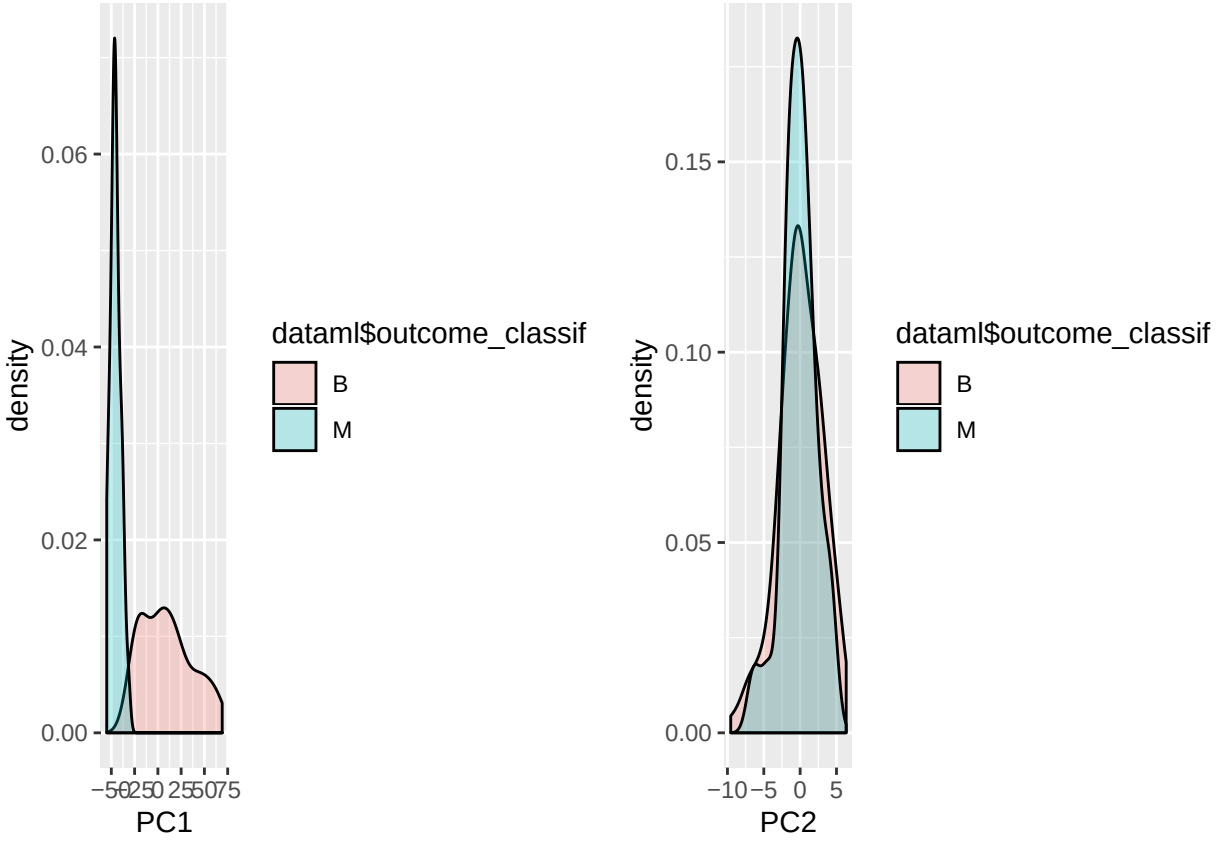
The 2 first components explains the 0.98160 of the variance.

```
pca_df <- as.data.frame(pca_res$x)
ggplot(pca_df, aes(x=PC1, y=PC2, col=dataml$outcome_classif)) + geom_point(alpha=0.5)
```



The data can be easly separated.

```
g_pc1 <- ggplot(pca_df, aes(x=PC1, fill=dataml$outcome_classif)) + geom_density(alpha=0.25)
g_pc2 <- ggplot(pca_df, aes(x=PC2, fill=dataml$outcome_classif)) + geom_density(alpha=0.25)
grid.arrange(g_pc1, g_pc2, ncol=2)
```



Here we observe the distribution of each component

LDA

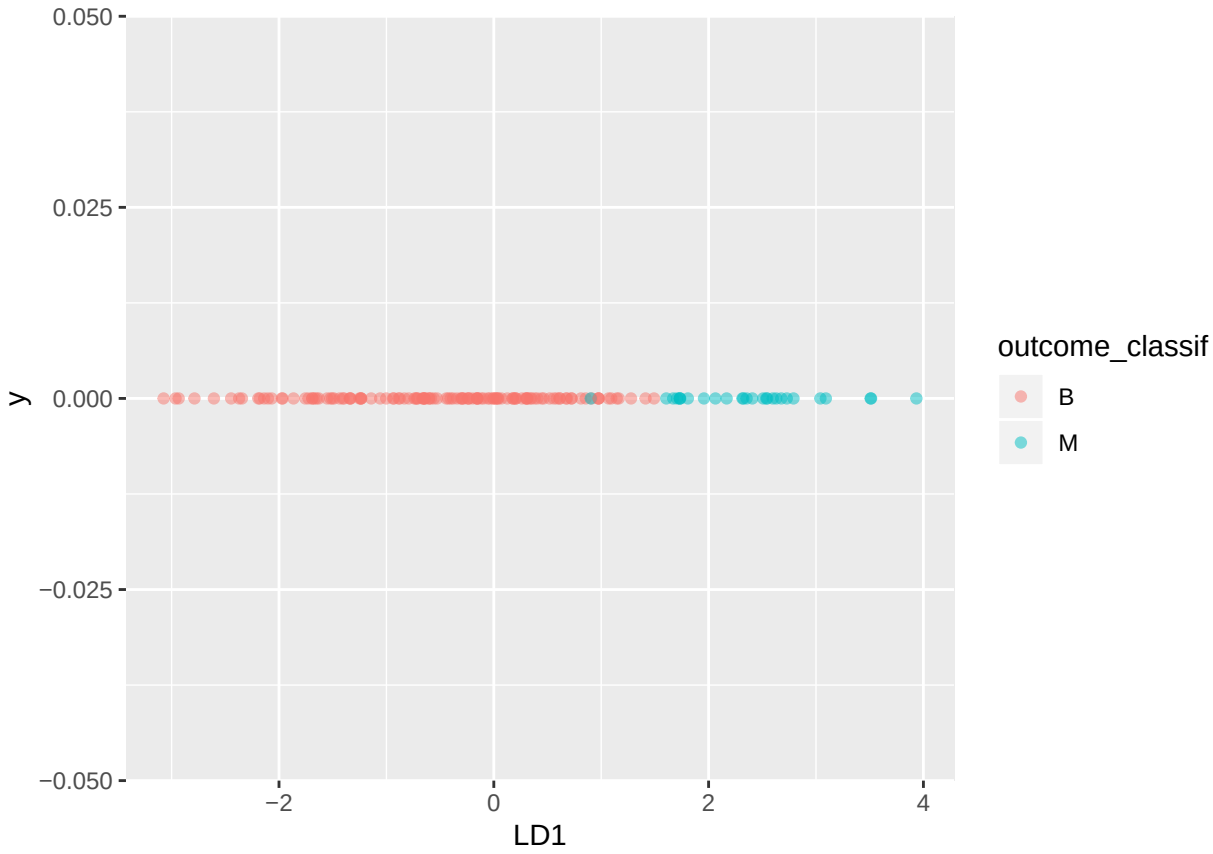
Let's try LDA instead of PCA. LDA take in consideration the different classes and could get better results

```
lda_res <- lda(formulaml(),dataml)
lda_df <- predict(lda_res,dataml)$x %>% as.data.frame() %>% cbind(outcome_classif=dataml$outcome_classi
lda_res
```

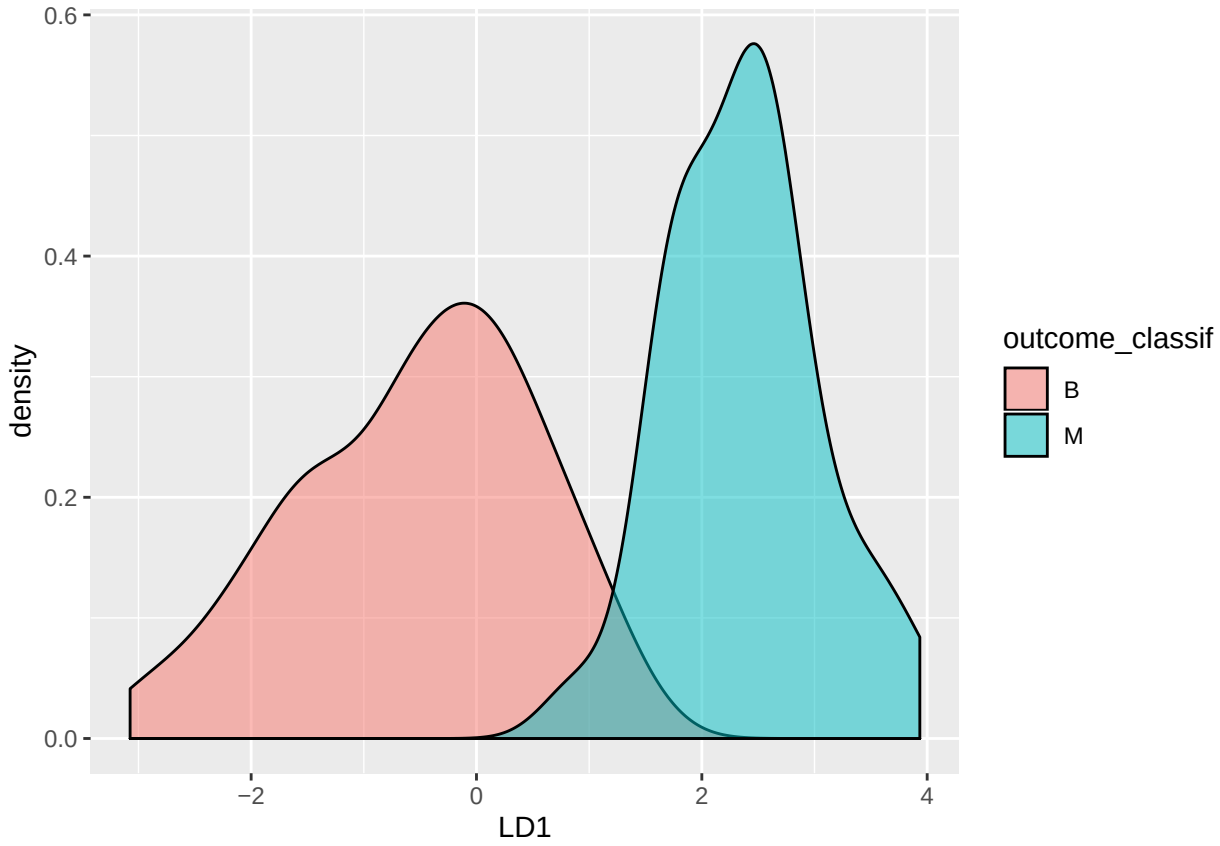
```
## Call:
## lda(formulaml(), data = dataml)
##
## Prior probabilities of groups:
##         B         M
## 0.8193548 0.1806452
##
## Group means:
##       time radius_mean texture_mean perimeter_mean  area_mean
## B 65.93701  -0.2058737   -0.1041309     -0.2063834 -0.2118710
## M 10.39286   0.6403132   -0.0422783      0.6425313  0.6788675
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## B      0.10089007      0.004324166    -0.07567636          -0.1006389
## M     -0.06864709      0.065986726     0.24716926           0.4359653
##   symmetry_mean fractal.dimension_mean  radius_SD texture_SD perimeter_SD
## B    0.08323665              0.1649092 -0.1135276 -0.1274691   -0.1268599
## M   -0.29966580             -0.4064695  0.3576584 -0.2160019    0.3417071
```

```
##      area_SD smoothness_SD compactness_SD concavity_SD concave.points_SD
## B -0.1550228    0.02731384     0.01655763  -0.04438862       -0.06580874
## M  0.4697535   -0.15917691    -0.04645573  -0.08853708       -0.11765593
##    symmetry_SD fractal.dimension_SD radius_worst texture_worst
## B  0.002780218            0.0773860   -0.1622700   -0.06463074
## M -0.084205130           -0.2383412    0.7822055   -0.02204507
##   perimeter_worst area_worst smoothness_worst compactness_worst
## B      -0.1697202 -0.1640616        0.1287648        0.07545488
## M       0.7646163  0.7951063        0.0193452       -0.01410382
##   concavity_worst concave.points_worst symmetry_worst
## B      0.01326722          -0.02089705      0.1085008
## M      0.11885489           0.34973142     -0.1270670
##   fractal.dimension_worst  Tumor.size Lymph.node.status
## B               0.1674132 -0.07650408       -0.08311951
## M              -0.2335090  0.37909276        0.43073000
##
## Coefficients of linear discriminants:
##                                LD1
## time                    -0.03761145
## radius_mean             -5.06894720
## texture_mean            -0.44137133
## perimeter_mean           2.29387231
## area_mean                2.17748355
## smoothness_mean          0.30190377
## compactness_mean         1.04468315
## concavity_mean          -1.28271534
## concave.points_mean      0.63088424
## symmetry_mean           -0.14739152
## fractal.dimension_mean  -0.85816216
## radius_SD               -0.16026409
## texture_SD              -0.48425350
## perimeter_SD             0.12975104
## area_SD                  0.32277559
## smoothness_SD            0.83276440
## compactness_SD           1.60487997
## concavity_SD            -0.93814516
## concave.points_SD       -0.59574719
## symmetry_SD              0.07513737
## fractal.dimension_SD    -0.52138800
## radius_worst             2.24265901
## texture_worst            0.37289175
## perimeter_worst         -0.79445107
## area_worst              -1.06896729
## smoothness_worst        -0.29203722
## compactness_worst       -2.00741248
## concavity_worst          1.12627500
## concave.points_worst     0.12130749
## symmetry_worst           0.29883438
## fractal.dimension_worst  0.69909162
## Tumor.size              -0.01709189
## Lymph.node.status        0.33297688
```

```
ggplot(lda_df, aes(x=LD1, y=0, col=outcome_classif)) + geom_point(alpha=0.5)
```




```
ggplot(lda_df, aes(x=LD1, fill=outcome_classif)) + geom_density(alpha=0.5)
```



We are going to create a training and test set of these data :

```
set.seed(1234)
data_index <- createDataPartition(dataml$outcome_classif, p=0.7, list = FALSE)
train_data_lda <- lda_df[data_index,]
test_data_lda <- lda_df[-data_index,]
```

Modeling

We are going to get a training and a testing set to use when building some models:

```
train_data <- dataml[data_index, 1:ncol(dataml)]
test_data <- dataml[-data_index, 1:ncol(dataml)]
```

The dataset is a bit unbalanced:

We will use two methods to work around this problem :

Under-sampling method :

```
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10,
                     verboseIter = FALSE,
```

```
                     sampling = "down")

set.seed(42)
model_rf_under <- caret::train(formulaml(),
                           data = train_data,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = ctrl)
```

We return the confusion matrix of this model:

```
pred_rf_under <- predict(model_rf_under, test_data)
cm_rf_under <- confusionMatrix(pred_rf_under, test_data$outcome_classif, positive = "M")
cm_rf_under
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 38  1
##          M  0  7
##
##                Accuracy : 0.9783
##                  95% CI : (0.8847, 0.9994)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.001629
##
##                   Kappa : 0.9204
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8750
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9744
##              Prevalence : 0.1739
##          Detection Rate : 0.1522
##    Detection Prevalence : 0.1522
##       Balanced Accuracy : 0.9375
##
##        'Positive' Class : M
##
```

Rose method :

```
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10,
                     verboseIter = FALSE,
                     sampling = "rose")
```

```r
set.seed(42)
model_rf_rose <- caret::train(formulaml(),
                          data = train_data,
                          method = "rf",
                          preProcess = c("scale", "center"),
                          trControl = ctrl)
```

```
## Loaded ROSE 0.0-3
```

Similarly, we return the confusion matrix :

```r
pred_rf_rose <- predict(model_rf_rose, test_data)
cm_rf_rose<- confusionMatrix(pred_rf_rose, test_data$outcome_classif, positive = "M")
cm_rf_rose
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 38  1
##          M  0  7
##
##                Accuracy : 0.9783
##                  95% CI : (0.8847, 0.9994)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.001629
##
##                   Kappa : 0.9204
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8750
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9744
##              Prevalence : 0.1739
##          Detection Rate : 0.1522
##    Detection Prevalence : 0.1522
##       Balanced Accuracy : 0.9375
##
##        'Positive' Class : M
##
```

Gradient boosting model

```r
fitControl <- trainControl(method="repeatedcv",
                           classProbs = TRUE,
                          number = 10,
                          ## repeated ten times
                          repeats = 10)
```

We train le gradient boosting model :

```
set.seed(825)
model_gbm <- train(formulaml(), data = train_data,
                   method = "gbm",
                   trControl = fitControl,
                   ## This last option is actually one
                   ## for gbm() that passes through
                   verbose = FALSE)
model_gbm
```

```
## Stochastic Gradient Boosting
##
## 109 samples
##  33 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 98, 98, 98, 98, 98, 98, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      1.0000000  1.0000000
##   1                  100      1.0000000  1.0000000
##   1                  150      1.0000000  1.0000000
##   2                   50      1.0000000  1.0000000
##   2                  100      1.0000000  1.0000000
##   2                  150      1.0000000  1.0000000
##   3                   50      0.9990909  0.9974419
##   3                  100      1.0000000  1.0000000
##   3                  150      1.0000000  1.0000000
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth
##  = 1, shrinkage = 0.1 and n.minobsinnode = 10.
```

We return the confusion matrix for this model :

```
pred_gbm <- predict(model_gbm, test_data)
pred_prob_gbm <- predict(model_gbm, dataml, type="prob")

cm_gbm <- confusionMatrix(pred_gbm, test_data$outcome_classif, positive = "M")
```

Random forest model :

```
model_rf <- train(formulaml(),
                  train_data,
```

```
                  method="ranger",
                  metric="ROC",
                  trControl=fitControl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "ROC" was not
## in the result set. Accuracy will be used instead.
```

```
pred_rf <- predict(model_rf, test_data)
cm_rf <- confusionMatrix(pred_rf, test_data$outcome_classif, positive = "M")
cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 38  1
##          M  0  7
##
##                Accuracy : 0.9783
##                  95% CI : (0.8847, 0.9994)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.001629
##
##                   Kappa : 0.9204
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8750
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9744
##              Prevalence : 0.1739
##          Detection Rate : 0.1522
##    Detection Prevalence : 0.1522
##       Balanced Accuracy : 0.9375
##
##        'Positive' Class : M
##
```

KNN model :

```
model_knn <- train(formulaml(),
                   train_data,
                   method="knn",
                   metric="ROC",
                   tuneLength=10,
                   trControl=fitControl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "ROC" was not
## in the result set. Accuracy will be used instead.
```

```r
pred_knn <- predict(model_knn, test_data)
cm_knn <- confusionMatrix(pred_knn, test_data$outcome_classif, positive = "M")
cm_knn
```

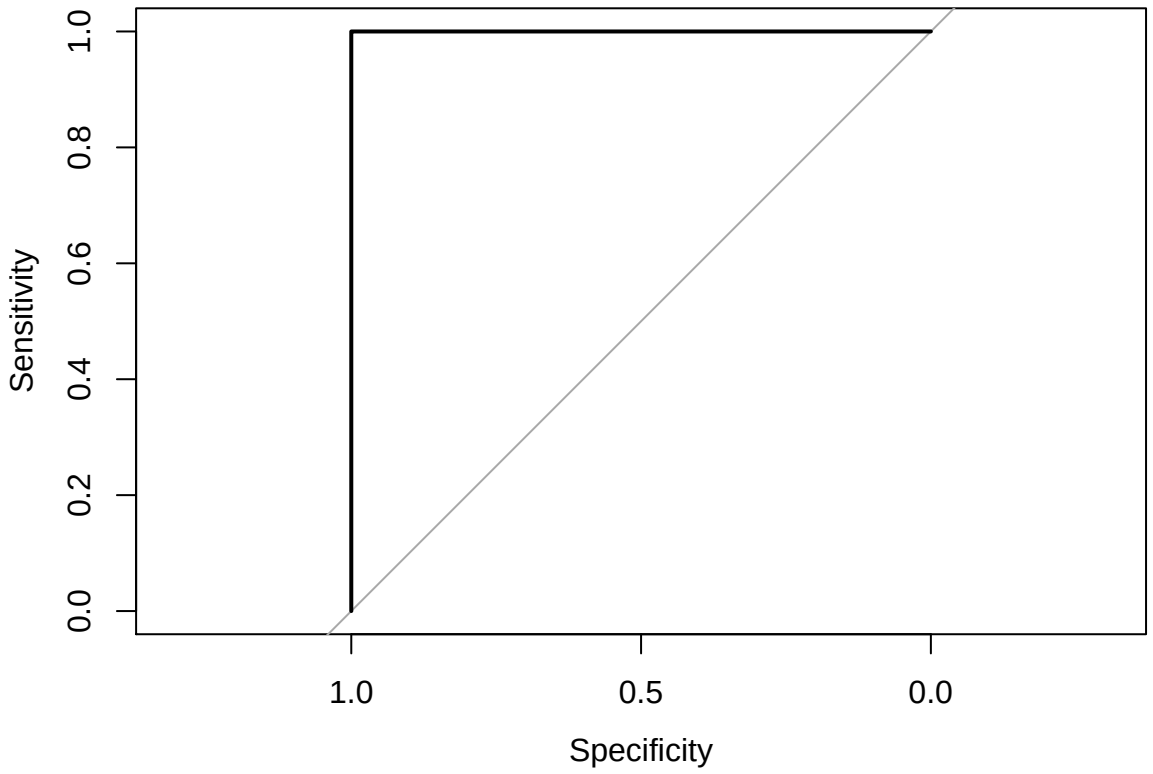
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 38  1
##          M  0  7
##
##                Accuracy : 0.9783
##                  95% CI : (0.8847, 0.9994)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.001629
##
##                   Kappa : 0.9204
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8750
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9744
##              Prevalence : 0.1739
##          Detection Rate : 0.1522
##    Detection Prevalence : 0.1522
##       Balanced Accuracy : 0.9375
##
##        'Positive' Class : M
##
```

```r
pred_prob_knn <- predict(model_knn, test_data, type="prob")
roc_knn <- roc(test_data$outcome_classif, pred_prob_knn$M)
```

```
## Setting levels: control = B, case = M
```

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

```r
plot(roc_knn)
```




```
pred_prob_knn
```

```
##           B         M
## 1  1.0000000 0.00000000
## 2  1.0000000 0.00000000
## 3  0.1304348 0.86956522
## 4  1.0000000 0.00000000
## 5  1.0000000 0.00000000
## 6  0.9565217 0.04347826
## 7  1.0000000 0.00000000
## 8  1.0000000 0.00000000
## 9  1.0000000 0.00000000
## 10 1.0000000 0.00000000
## 11 1.0000000 0.00000000
## 12 1.0000000 0.00000000
## 13 0.1304348 0.86956522
## 14 1.0000000 0.00000000
## 15 1.0000000 0.00000000
## 16 0.1304348 0.86956522
## 17 1.0000000 0.00000000
## 18 1.0000000 0.00000000
## 19 1.0000000 0.00000000
## 20 1.0000000 0.00000000
## 21 1.0000000 0.00000000
## 22 1.0000000 0.00000000
## 23 1.0000000 0.00000000
## 24 1.0000000 0.00000000
## 25 1.0000000 0.00000000
## 26 1.0000000 0.00000000
```

```
## 27 0.1304348 0.86956522
## 28 1.0000000 0.00000000
## 29 1.0000000 0.00000000
## 30 1.0000000 0.00000000
## 31 1.0000000 0.00000000
## 32 1.0000000 0.00000000
## 33 0.5652174 0.43478261
## 34 0.1304348 0.86956522
## 35 1.0000000 0.00000000
## 36 1.0000000 0.00000000
## 37 1.0000000 0.00000000
## 38 1.0000000 0.00000000
## 39 1.0000000 0.00000000
## 40 0.1304348 0.86956522
## 41 1.0000000 0.00000000
## 42 1.0000000 0.00000000
## 43 1.0000000 0.00000000
## 44 0.8260870 0.17391304
## 45 0.9565217 0.04347826
## 46 0.1304348 0.86956522
```

```
auc(roc_knn)
```

```
## Area under the curve: 1
```

Naive Bayes model :

```
model_nb
```

```
## Naive Bayes
##
## 109 samples
##  33 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 98, 99, 98, 98, 98, 98, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.9011818  0.5975284
##    TRUE      0.8450909  0.4869965
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE
##  and adjust = 1.
```

```
cm_nb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 37  1
##          M  1  7
##
##                Accuracy : 0.9565
##                  95% CI : (0.8516, 0.9947)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.008623
##
##                   Kappa : 0.8487
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8750
##             Specificity : 0.9737
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.9737
##              Prevalence : 0.1739
##          Detection Rate : 0.1522
##    Detection Prevalence : 0.1739
##       Balanced Accuracy : 0.9243
##
##        'Positive' Class : M
##
```

SVM model :

```
set.seed(420)

  model_svm <- train(formulaml(),
                     train_data,
                     method="svmRadial",
                     metric="ROC",
                     preProcess=c('center', 'scale'),
                     trace=FALSE,

                                  trControl=fitControl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "ROC" was not
## in the result set. Accuracy will be used instead.
```

```
pred_svm <- predict(model_svm, test_data)
cm_svm <- confusionMatrix(pred_svm, test_data$outcome_classif, positive = "M")
cm_svm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 38  3
##          M  0  5
##
##                Accuracy : 0.9348
##                  95% CI : (0.821, 0.9863)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.03022
##
##                   Kappa : 0.7336
##
##  Mcnemar's Test P-Value : 0.24821
##
##             Sensitivity : 0.6250
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9268
##              Prevalence : 0.1739
##          Detection Rate : 0.1087
##    Detection Prevalence : 0.1087
##       Balanced Accuracy : 0.8125
##
##        'Positive' Class : M
##
```

Neural Networks (NNET) with PCA

```
model_pca_nnet <- train(formulaml(),
                    train_data,
                    method="nnet",
                    metric="ROC",
                    preProcess=c('center', 'scale', 'pca'),
                    tuneLength=10,
                    trace=FALSE,
                    trControl=fitControl)
```

```
pred_pca_nnet <- predict(model_pca_nnet, test_data)
cm_pca_nnet <- confusionMatrix(pred_pca_nnet, test_data$outcome_classif, positive = "M")
cm_pca_nnet
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 37  0
##          M  1  8
##
##                Accuracy : 0.9783
##                  95% CI : (0.8847, 0.9994)
```

```
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.001629
##
##                   Kappa : 0.9279
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 1.0000
##             Specificity : 0.9737
##          Pos Pred Value : 0.8889
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1739
##          Detection Rate : 0.1739
##    Detection Prevalence : 0.1957
##       Balanced Accuracy : 0.9868
##
##        'Positive' Class : M
##
```

Neural Networks (NNET) with LDA

```
model_lda_nnet <- train(outcome_classif~.,
                    train_data_lda,
                    method="nnet",
                    metric="ROC",
                    preProcess=c('center', 'scale'),
                    tuneLength=10,
                    trace=FALSE,
                    trControl=fitControl)
```

```
pred_lda_nnet <- predict(model_lda_nnet, test_data_lda)
cm_lda_nnet <- confusionMatrix(pred_lda_nnet, test_data_lda$outcome_classif, positive = "M")
cm_lda_nnet
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 38  0
##          M  0  8
##
##                Accuracy : 1
##                  95% CI : (0.9229, 1)
##     No Information Rate : 0.8261
##     P-Value [Acc > NIR] : 0.0001525
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
```

```
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1739
##          Detection Rate : 0.1739
##    Detection Prevalence : 0.1739
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : M
##
```

Results of ML models : comparison

Let's compare the models and check their correlation:

```
models<- list(RF=model_rf, RF_under=model_rf_under,RF_rose=model_rf_rose,GBM=model_gbm,
                    PCANNET=model_pca_nnet,LDANNET=model_lda_nnet,
                   KNN = model_knn, SVM=model_svm, NB=model_nb)
resampling <- resamples(models)
```

```
nm <-deparse(substitute(models[1]))
   print(nm)
```

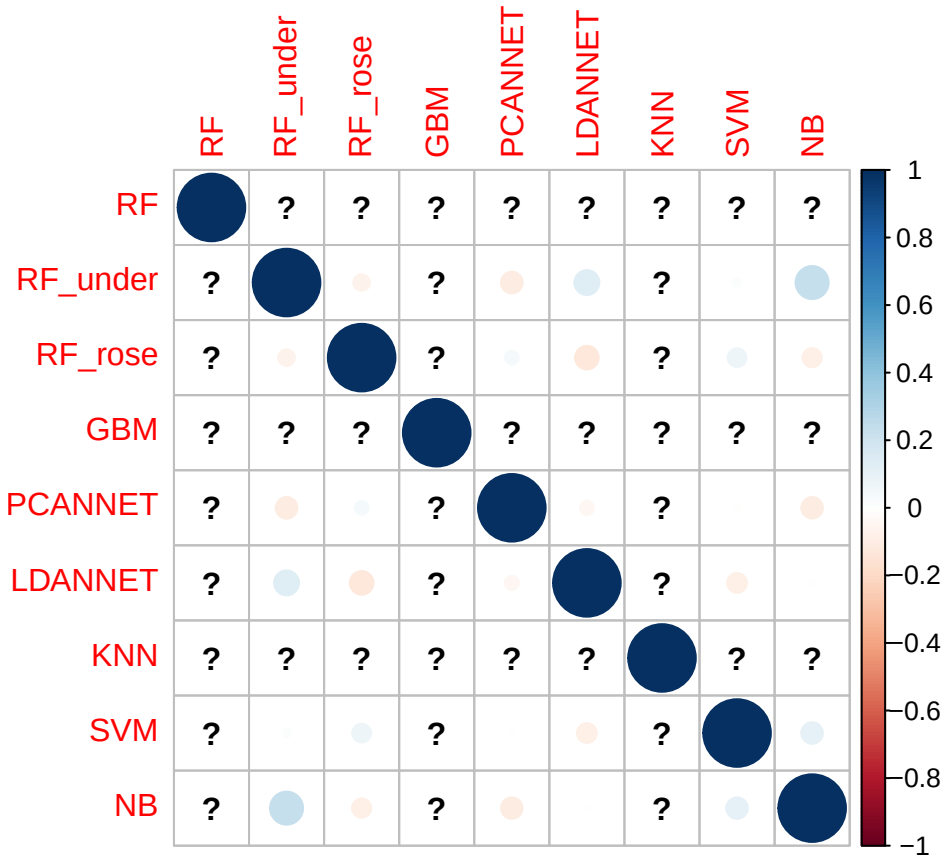
```
## [1] "models[1]"
```

Correlation between models

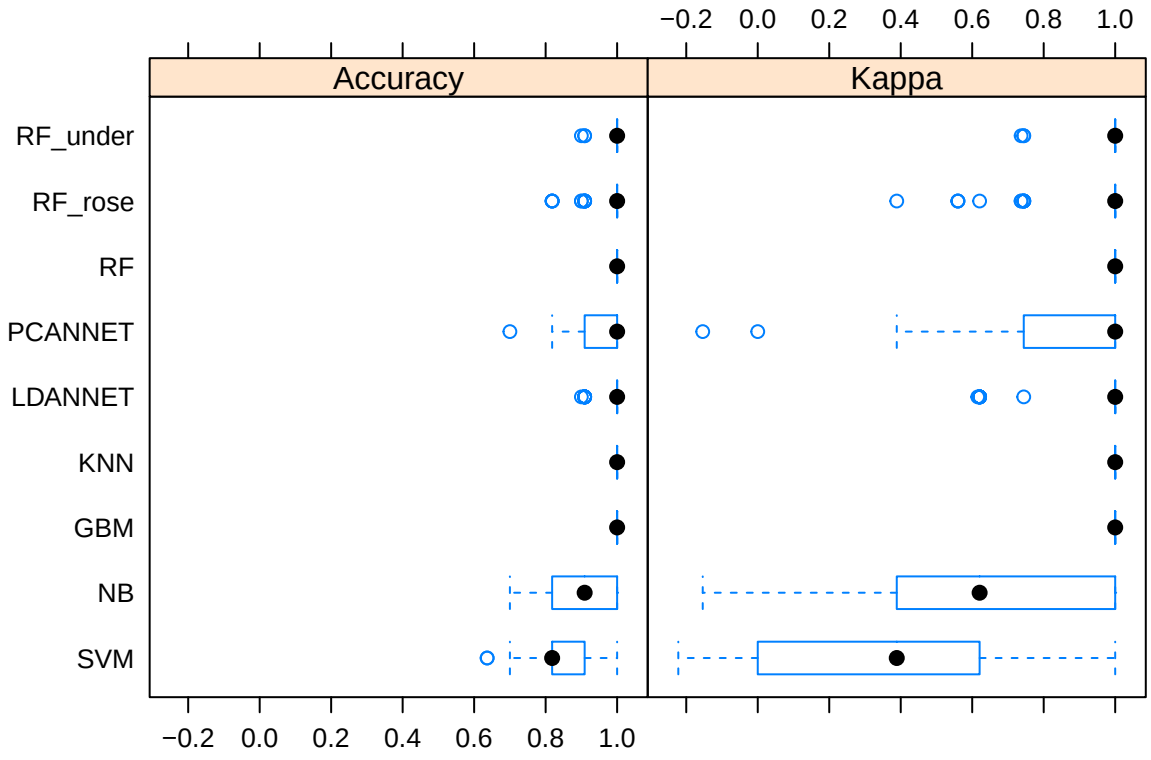
```
model_cor <- modelCor(resampling)
```

```
## Warning in cor(dat, ...): l'écart type est nulle
```

```
corrplot(model_cor)
```




```
bwplot(resampling)
```



We see here that some models have a great variability depending of the processed sample (NB,SVM).

```
cm_list <- list(RF=cm_rf,RF_under=cm_rf_under,RF_rose=cm_rf_rose,
                  PCANNET=cm_pca_nnet,LDANNET=cm_lda_nnet,
                GBM=cm_gbm,
                 KNN = cm_knn, SVM=cm_svm, NB=cm_nb)
cm_list_results <- sapply(cm_list, function(x) x$byClass)
cm_list_results
```

```
##                            RF  RF_under   RF_rose   PCANNET  LDANNET
## Sensitivity          0.8750000 0.8750000 0.8750000 1.0000000 1.000000
## Specificity          1.0000000 1.0000000 1.0000000 0.9736842 1.000000
## Pos Pred Value       1.0000000 1.0000000 1.0000000 0.8888889 1.000000
## Neg Pred Value       0.9743590 0.9743590 0.9743590 1.0000000 1.000000
## Precision            1.0000000 1.0000000 1.0000000 0.8888889 1.000000
## Recall               0.8750000 0.8750000 0.8750000 1.0000000 1.000000
## F1                   0.9333333 0.9333333 0.9333333 0.9411765 1.000000
## Prevalence           0.1739130 0.1739130 0.1739130 0.1739130 0.173913
## Detection Rate       0.1521739 0.1521739 0.1521739 0.1739130 0.173913
## Detection Prevalence 0.1521739 0.1521739 0.1521739 0.1956522 0.173913
## Balanced Accuracy    0.9375000 0.9375000 0.9375000 0.9868421 1.000000
##                            GBM       KNN       SVM        NB
## Sensitivity          0.8750000 0.8750000 0.6250000 0.8750000
## Specificity          1.0000000 1.0000000 1.0000000 0.9736842
## Pos Pred Value       1.0000000 1.0000000 1.0000000 0.8750000
## Neg Pred Value       0.9743590 0.9743590 0.9268293 0.9736842
## Precision            1.0000000 1.0000000 1.0000000 0.8750000
## Recall               0.8750000 0.8750000 0.6250000 0.8750000
## F1                   0.9333333 0.9333333 0.7692308 0.8750000
## Prevalence           0.1739130 0.1739130 0.1739130 0.1739130
## Detection Rate       0.1521739 0.1521739 0.1086957 0.1521739
## Detection Prevalence 0.1521739 0.1521739 0.1086957 0.1739130
## Balanced Accuracy    0.9375000 0.9375000 0.8125000 0.9243421
```

```
cm_results_max <- apply(cm_list_results, 1, which.is.max)
output_report <- data.frame(metric=names(cm_results_max),
                            best_model=colnames(cm_list_results)[cm_results_max],
                            value=mapply(function(x,y) {cm_list_results[x,y]},
                            names(cm_results_max),
                            cm_results_max))
rownames(output_report) <- NULL
output_report
```

```
##                  metric best_model     value
## 1           Sensitivity    LDANNET 1.0000000
## 2           Specificity   RF_under 1.0000000
## 3        Pos Pred Value    LDANNET 1.0000000
## 4        Neg Pred Value    LDANNET 1.0000000
## 5             Precision        SVM 1.0000000
## 6                Recall    LDANNET 1.0000000
## 7                    F1    LDANNET 1.0000000
## 8            Prevalence        SVM 0.1739130
## 9        Detection Rate    LDANNET 0.1739130
```

```
## 10 Detection Prevalence    PCANNET 0.1956522
## 11    Balanced Accuracy    LDANNET 1.0000000
```

The best results for sensitivity (detection of breast cases) are LDANNET and PCANNET which also have a great F1 score.

First approche : predicting prob with ML

We calculate the average relapse probability at 24 months for each ML model

```
prob_rechute_mean <- function(model,data){
  pred_prob <- predict(model, data, type="prob")
  return(mean(pred_prob[,'M']))
}
```

```
# without LDANNET & NB
models2<- list(RF=model_rf, RF_under=model_rf_under,RF_rose=model_rf_rose,GBM=model_gbm,
                    PCANNET=model_pca_nnet,
                   KNN = model_knn, SVM=model_svm)
prob_rechute_list <- sapply(models2, function(mod) prob_rechute_mean(mod,dataml) )
ml_prob_report <- data.frame(model=names(models2),prob_rechute_mean=prob_rechute_list )
rownames(ml_prob_report) <- NULL
ml_prob_report
```

```
##      model prob_rechute_mean
## 1       RF         0.1805290
## 2 RF_under         0.1775613
## 3  RF_rose         0.2139613
## 4      GBM         0.1742541
## 5  PCANNET         0.1873006
## 6      KNN         0.1576438
## 7      SVM         0.1893333
```

Second approch : Data sampling & ML

We try another method by sampling training data randomly

```
generate_data<-function(data,n){
  df=data_frame(time=runif(n,min(dataml[,"time"]),max(dataml[,"time"])))
  for (l in names(dataml) ){
  if (l!="outcome_classif" & l!="time"){
     df[,l]<-runif(n,min(dataml[,l]),max(dataml[,l]))
  }}
      return(df)

}
```

```
prob_chute_mean_sp <-function(model,n){
  df=generate_data(dataml,n)
  pred_prob <- predict(model, df, type="prob")
  return(mean(pred_prob[,'M']))
}
```

```
size_data_sp <-10000
prob_rechute_list_sp <- sapply(models2, function(mod) prob_chute_mean_sp(mod,size_data_sp) )
ml_prob_report_sp <- data.frame(model=names(models2),prob_rechute_mean=prob_rechute_list_sp )
rownames(ml_prob_report_sp) <- NULL
ml_prob_report_sp
```

```
##      model prob_rechute_mean
## 1       RF         0.2234928
## 2 RF_under         0.1703788
## 3  RF_rose         0.1819078
## 4      GBM         0.1746615
## 5  PCANNET         0.4367130
## 6      KNN         0.1565092
## 7      SVM         0.1373955
```

Conclusion : ML vs Survival models

We have for survival models :

```
surv_prob_report
```

```
##   prob_rechute_mean.KM prob_rechute_mean.COX prob_rechute_mean.RSF
## 1            0.1729175             0.1108799             0.1768696
```

and for ML methods (2 approaches) :

- Directely from ML output :

```
ml_prob_report
```

```
##      model prob_rechute_mean
## 1       RF         0.1805290
## 2 RF_under         0.1775613
## 3  RF_rose         0.2139613
## 4      GBM         0.1742541
## 5  PCANNET         0.1873006
## 6      KNN         0.1576438
## 7      SVM         0.1893333
```

- By sampling new data (size=10000):

```
ml_prob_report_sp
```

```
##      model prob_rechute_mean
## 1       RF         0.2234928
## 2 RF_under         0.1703788
## 3  RF_rose         0.1819078
## 4      GBM         0.1746615
## 5  PCANNET         0.4367130
## 6      KNN         0.1565092
## 7      SVM         0.1373955
```