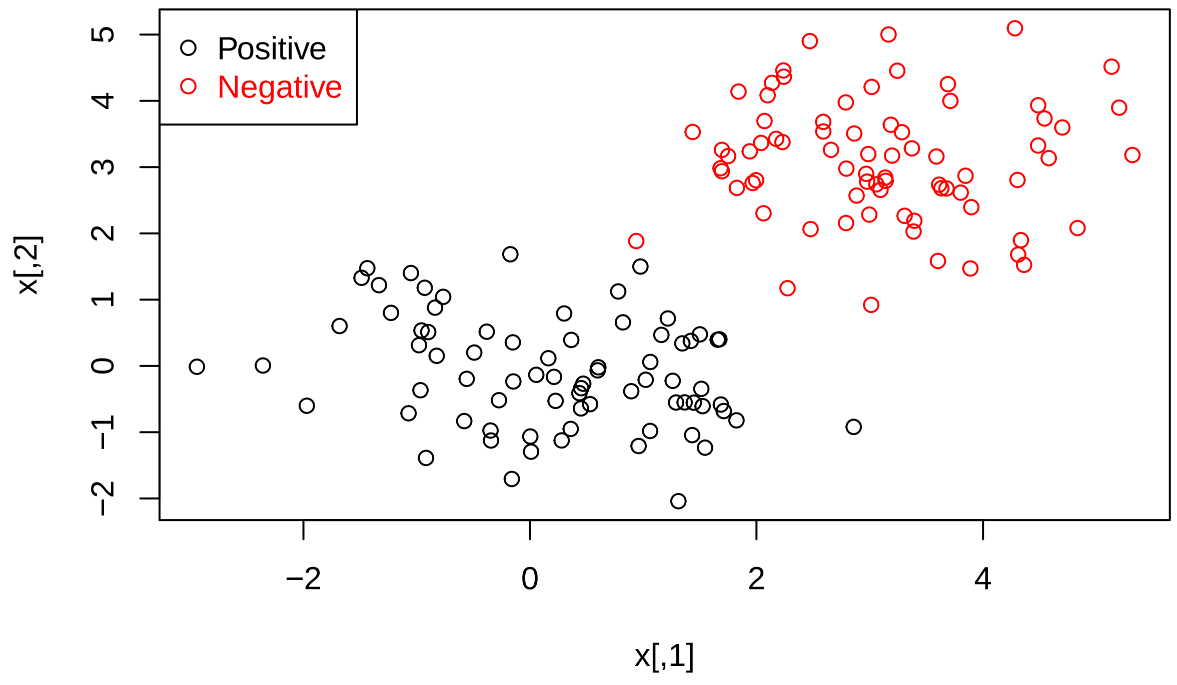
SVM Mini_Projet

Walid Keddad

SVM Lineaire:

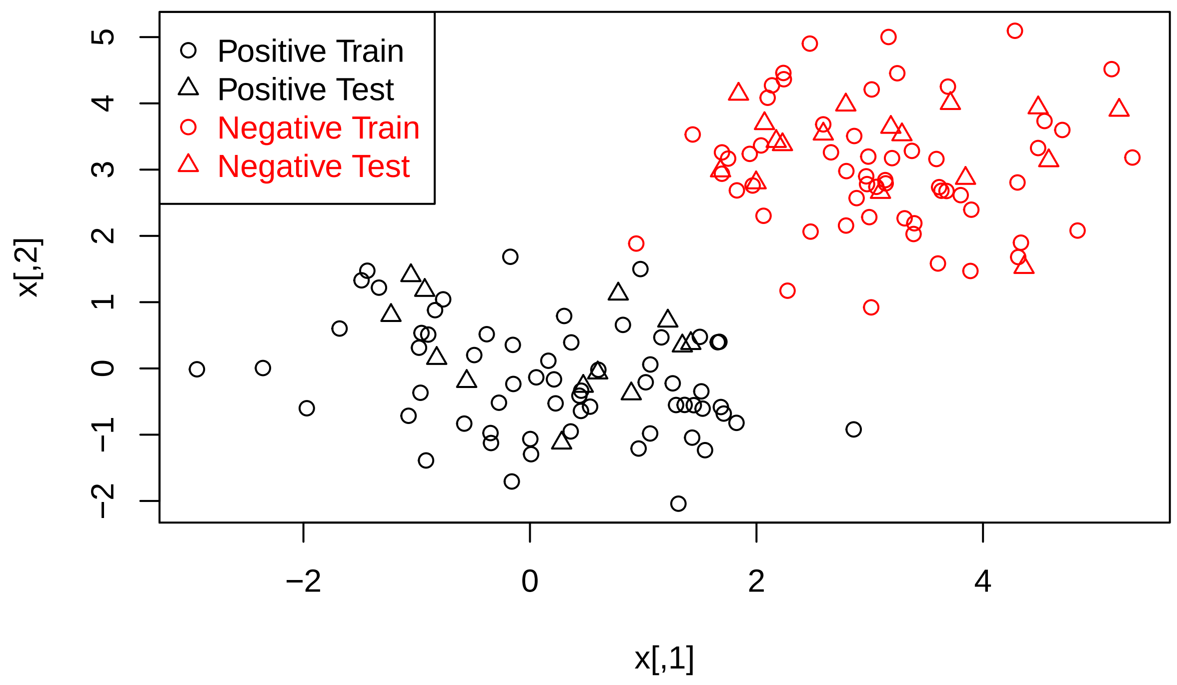
Genereation du jeux de données :

```r
n <- 150 # number of data points
p <- 2 # dimension
sigma <- 1 # variance of the distribution
meanpos <- 0 # centre of the distribution of positive examples
meanneg <- 3 # centre of the distribution of negative examples
npos <- round(n/2) # number of positive examples
nneg <- n-npos # number of negative examples
# Generate the positive and negative examples
xpos <- matrix(rnorm(npos*p,mean=meanpos,sd=sigma),npos,p)
xneg <- matrix(rnorm(nneg*p,mean=meanneg,sd=sigma),npos,p)
x <- rbind(xpos,xneg)
# Generate the labels
y <- matrix(c(rep(1,npos),rep(-1,nneg)))
# Visualize the data
plot(x,col=ifelse(y>0,1,2))
legend("topleft",c('Positive','Negative'),col=seq(2),pch=1,text.col=seq(2))
```



Maintenant, nous divisons les données en un ensemble d'entraînement (80%) et un ensemble de tests (20%)

```
## Prepare a training and a test set ##
ntrain <- round(n*0.8) # number of training examples
tindex <- sample(n,ntrain) # indices of training samples
xtrain <- x[tindex,]
xtest <- x[-tindex,]
ytrain <- y[tindex]
ytest <- y[-tindex]
istrain=rep(0,n)
istrain[tindex]=1
# Visualize
plot(x,col=ifelse(y>0,1,2),pch=ifelse(istrain==1,1,2))
legend("topleft",c('Positive Train','Positive Test','Negative Train','Negative Test'),
       col=c(1,1,2,2),pch=c(1,2,1,2),text.col=c(1,1,2,2))
```



Entrainer le SVM

Maintenant nous formons un SVM linéaire avec le paramètre C = 100 sur l'ensemble d'entraînement.

```
# load the kernlab package
library(kernlab)
```

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

```r
# train the SVM
svp <- ksvm(xtrain,ytrain,type="C-svc",kernel='vanilladot',C=100,scaled=c())
```

```
##  Setting default kernel parameters
```

```r
# Look and understand what svp contains
# General summary
svp
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 3
##
## Objective Function Value : -19.2102
## Training error : 0
```

```r
# Attributes that you can access
attributes(svp)
```

```
## $param
## $param$C
## [1] 100
##
##
## $scaling
## `\001NULL\001`
##
## $coef
## $coef[[1]]
## [1]  19.204605  -2.709603 -16.495002
##
##
## $alphaindex
## $alphaindex[[1]]
## [1] 18 24 36
##
##
## $b
## [1] -12.04132
##
## $obj
## [1] -19.21016
##
## $SVindex
## [1] 18 24 36
##
## $nSV
## [1] 3
##
## $prior
## $prior[[1]]
## $prior[[1]]$prior1
## [1] 62
##
## $prior[[1]]$prior0
## [1] 58
##
##
##
## $prob.model
## $prob.model[[1]]
## NULL
##
##
## $alpha
## $alpha[[1]]
## [1] 19.204605  2.709603 16.495002
##
##
## $type
## [1] "C-svc"
##
## $kernelf
## function (x, y = NULL)
## {
##     if (!is(x, "vector"))
##         stop("x must be a vector")
##     if (!is(y, "vector") && !is.null(y))
##         stop("y must be a vector")
##     if (is(x, "vector") && is.null(y)) {
##         crossprod(x)
##     }
##     if (is(x, "vector") && is(y, "vector")) {
##         if (!length(x) == length(y))
##             stop("number of dimension must be the same on both data points")
##         crossprod(x, y)
##     }
## }
## <environment: 0x0000000006069998>
## attr(,"kpar")
## list()
## attr(,"class")
## [1] "vanillakernel"
## attr(,"class")attr(,"package")
## [1] "kernlab"
##
## $kpar
## list()
##
## $xmatrix
## $xmatrix[[1]]
##           X1       X2
## 18 0.9752459 1.499407
## 24 2.2741594 1.172822
## 36 0.9385512 1.884582
##
##
## $ymatrix
##   [1] -1  1 -1  1 -1  1  1 -1  1  1 -1 -1 -1 -1  1  1 -1  1  1  1 -1  1  1
##  [24] -1  1 -1  1 -1  1 -1  1  1  1 -1  1 -1 -1  1  1  1 -1 -1  1 -1  1  1
##  [47] -1  1  1  1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1  1  1 -1
##  [70]  1  1 -1 -1  1  1 -1  1 -1  1  1  1 -1 -1 -1 -1  1 -1  1 -1  1 -1 -1
##  [93]  1 -1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1  1  1  1 -1
## [116] -1  1 -1  1 -1
##
## $fitted
##   [1] -1  1 -1  1 -1  1  1 -1  1  1 -1 -1 -1 -1  1  1 -1  1  1  1 -1  1  1
##  [24] -1  1 -1  1 -1  1 -1  1  1  1 -1  1 -1 -1  1  1  1 -1 -1  1 -1  1  1
##  [47] -1  1  1  1 -1 -1 -1 -1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1  1  1 -1
##  [70]  1  1 -1 -1  1  1 -1  1 -1  1  1  1 -1 -1 -1 -1  1 -1  1 -1  1 -1 -1
##  [93]  1 -1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1  1  1  1 -1
## [116] -1  1 -1  1 -1
##
## $lev
## [1] -1  1
##
## $nclass
## [1] 2
##
## $error
## [1] 0
##
## $cross
## [1] -1
##
## $n.action
## function (object, ...)
## UseMethod("na.omit")
## <bytecode: 0x0000000006ea85b0>
## <environment: namespace:stats>
##
## $terms
## `\001NULL\001`
##
## $kcall
## .local(x = x, y = ..1, scaled = ..5, type = "C-svc", kernel = "vanilladot",
##     C = 100)
##
## $class
## [1] "ksvm"
## attr(,"package")
## [1] "kernlab"
```

```r
# For example, the support vectors
alpha(svp)
```

```
## [[1]]
## [1] 19.204605  2.709603 16.495002
```

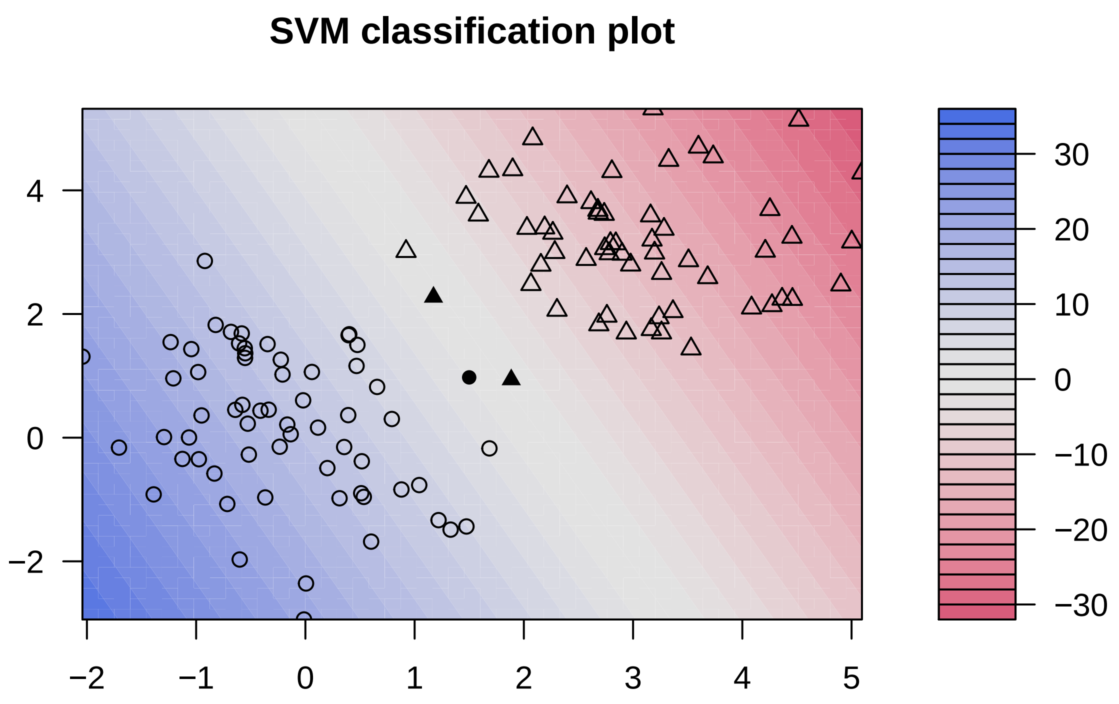
```
alphaindex(svp)
```

```
## [[1]]
## [1] 18 24 36
```

```
b(svp)
```

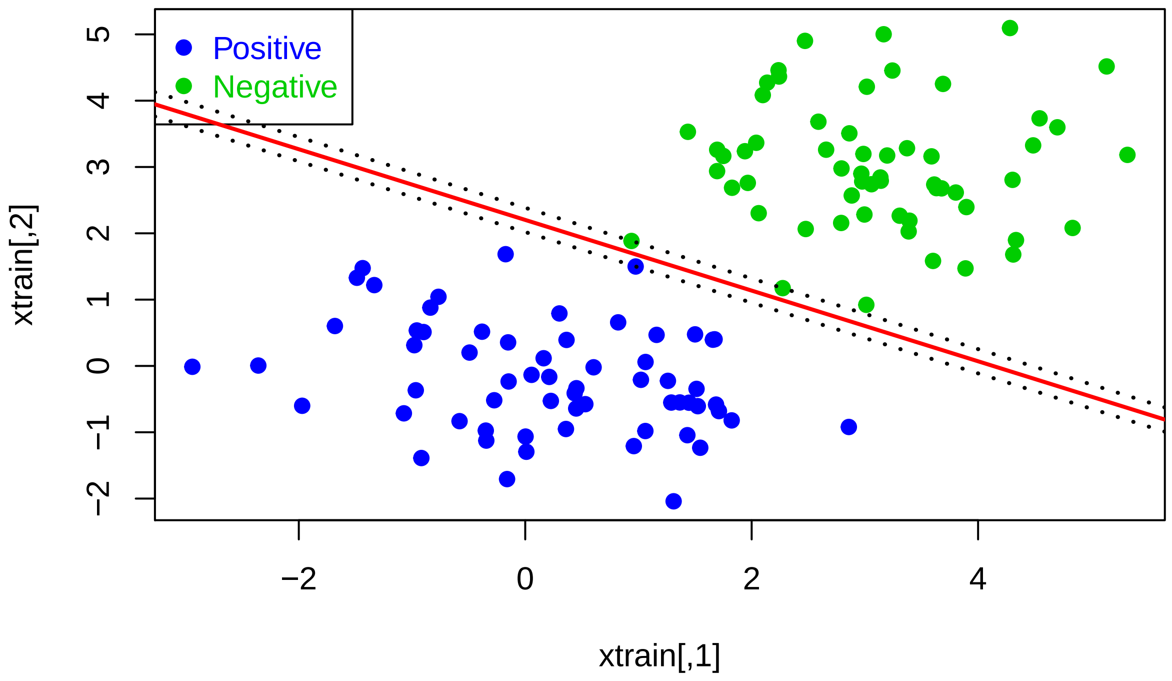
```
## [1] -12.04132
```

```
# Use the built-in function to pretty-plot the classifier
plot(svp,data=xtrain)
```



Question 1 :

```
plotlinearsvm <- function(svp, xtrain){
  plot(xtrain, col = ifelse(ytrain > 0, 4, 11) , pch=19)
  legend("topleft", c("Positive", "Negative"), col = c(4,11) , pch =19 , text.col = c(4,11))
  intercept = b(svp)
  slope = colSums(unlist(alpha(svp)) * ytrain[unlist(alphaindex(svp))]*xtrain[unlist(alphaindex(svp)),]
  abline(a= intercept / slope[2], b = -slope[1]/slope[2] , col = 2 , lwd = 2)
  abline(a= (intercept+1)/ slope[2], b = -slope[1]/slope[2], lty = 3 , lwd = 2)
  abline(a= (intercept-1)/ slope[2], b = -slope[1]/slope[2], lty = 3 , lwd = 2)
}
plotlinearsvm(svp, xtrain)
```



Prédire :

Maintenant, nous pouvons utiliser le SVM entraîné pour prédire l'étiquette des points dans le jeu de test, et nous analysons les résultats en utilisant plusieurs variantes de métriques.

```
# Predict labels on test
ypred = predict(svp,xtest)
table(ytest,ypred)
```

```
##       ypred
## ytest -1  1
##    -1 17  0
##    1   0 13
```

```
# Compute accuracy
sum(ypred==ytest)/length(ytest)
```

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

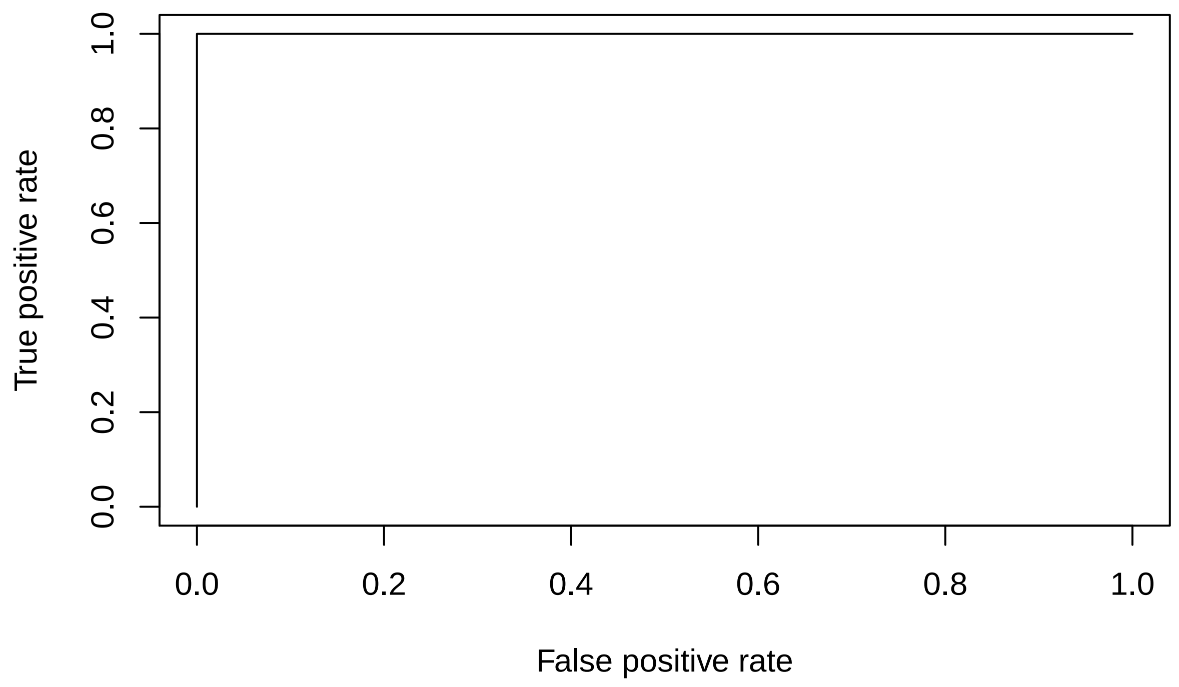
```
# Compute at the prediction scores
ypredscore = predict(svp,xtest,type="decision")
# Check that the predicted labels are the signs of the scores
table(ypredscore > 0,ypred)
```

```
##        ypred
##          -1  1
##   FALSE 17  0
##   TRUE   0 13
```

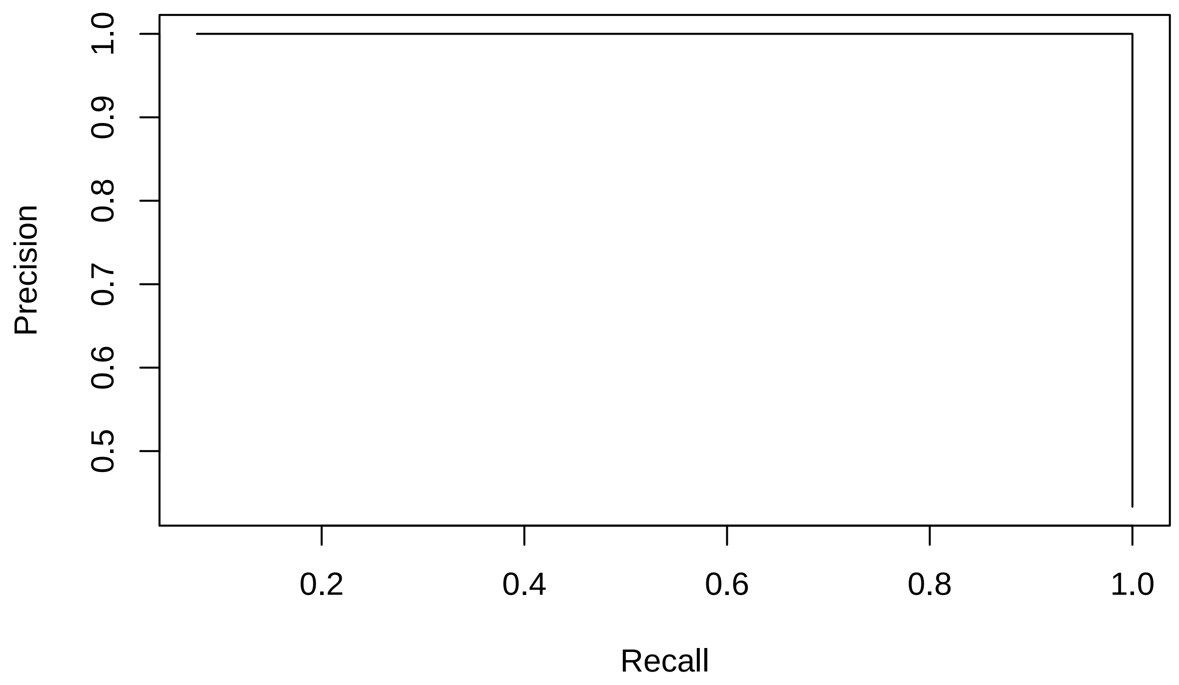
```
# Package to compute ROC curve, precision-recall etc...
library(ROCR)
```

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

## Loading required package: gplots

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

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

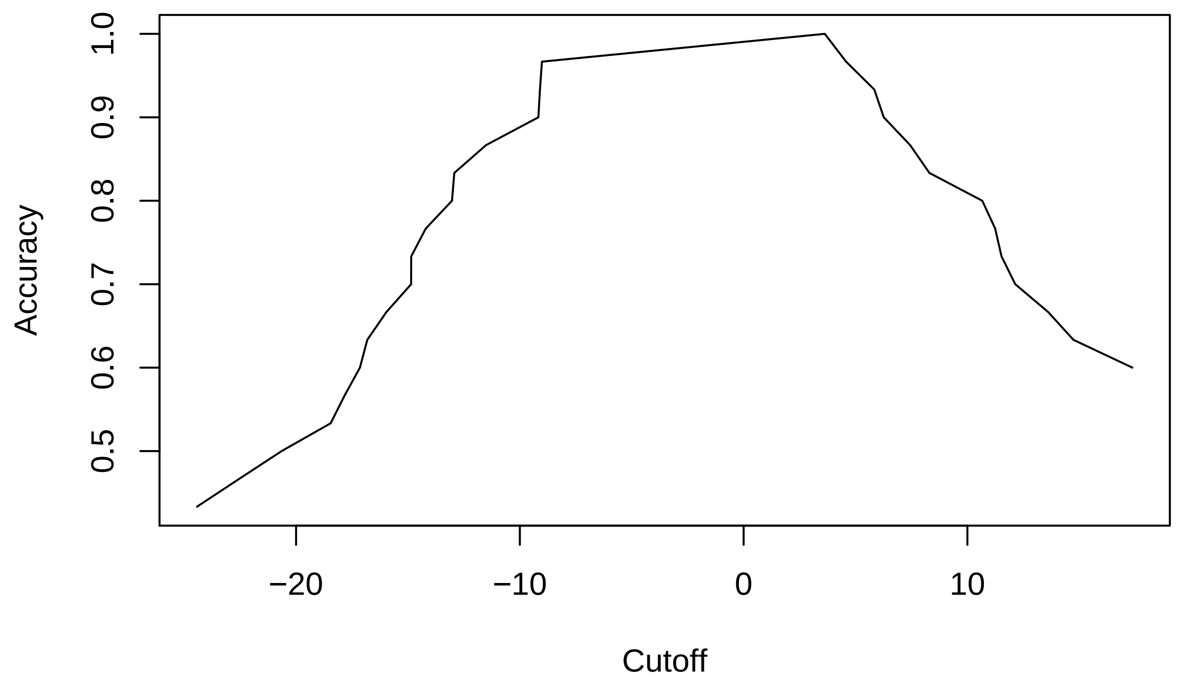
```
pred <- prediction(ypredscore,ytest)
# Plot ROC curve
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
```



```r
# Plot precision/recall curve
perf <- performance(pred, measure = "prec", x.measure = "rec")
plot(perf)
```



```r
# Plot accuracy as function of threshold
perf <- performance(pred, measure = "acc")
plot(perf)
```



Cross__Validation

Au lieu de fixer un ensemble d'apprentissage et un ensemble de tests, nous pouvons améliorer la qualité de ces estimations en exécutant k-fold cross-validation. Nous divisons l'ensemble d'entraînement en k groupes d'environ la même taille, puis itérativement former un SVM en utilisant k-1 groupes et faire de la prédiction sur le groupe qui a été laissé de côté. Lorsque k est égal au nombre de points de formation, on parle de leave-one-out (LOO) cross-validation. Pour générer une partition aléatoire de n points dans k plis (fold), nous pouvons par exemple créer la fonction suivante :

```
cv.folds <- function(n,folds=3)
  ## randomly split the n samples into folds
{
  split(sample(n),rep(1:folds,length=length(y)))
}
```

Question 3 :

```
svp <- ksvm(x,y,type="C-svc",kernel='vanilladot',C=1,scaled=c(),cross=5)
```

```
##  Setting default kernel parameters
```

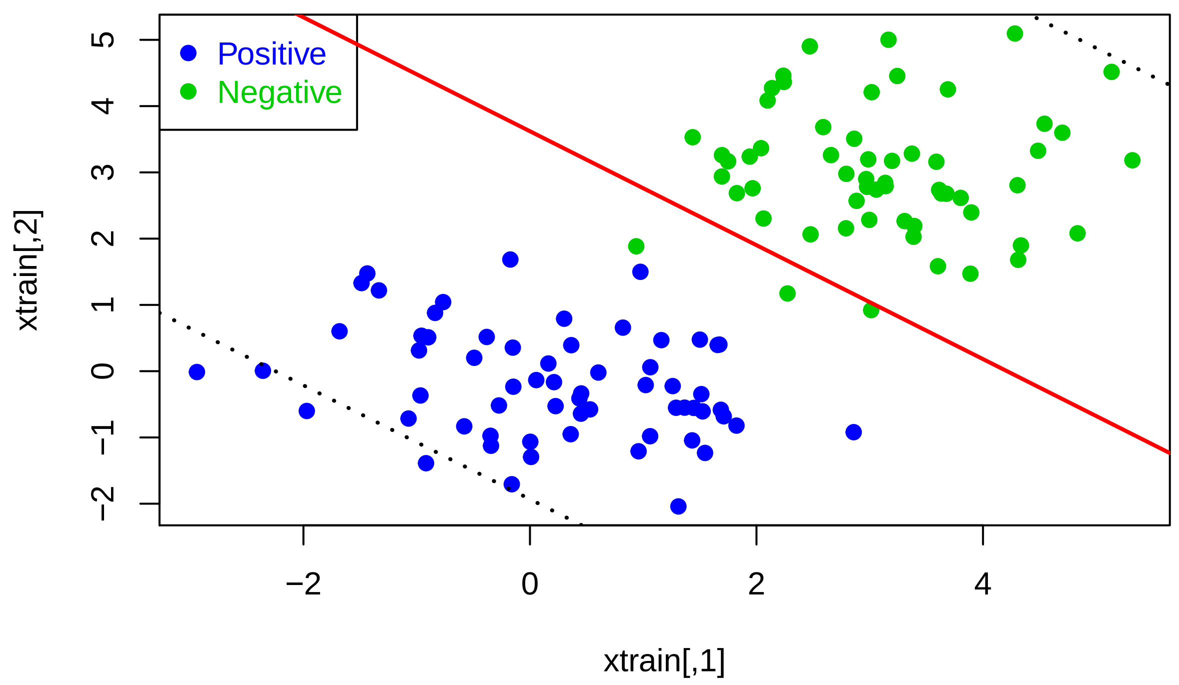
```
print(cross(svp))
```

```
## [1] 0.006666667
```

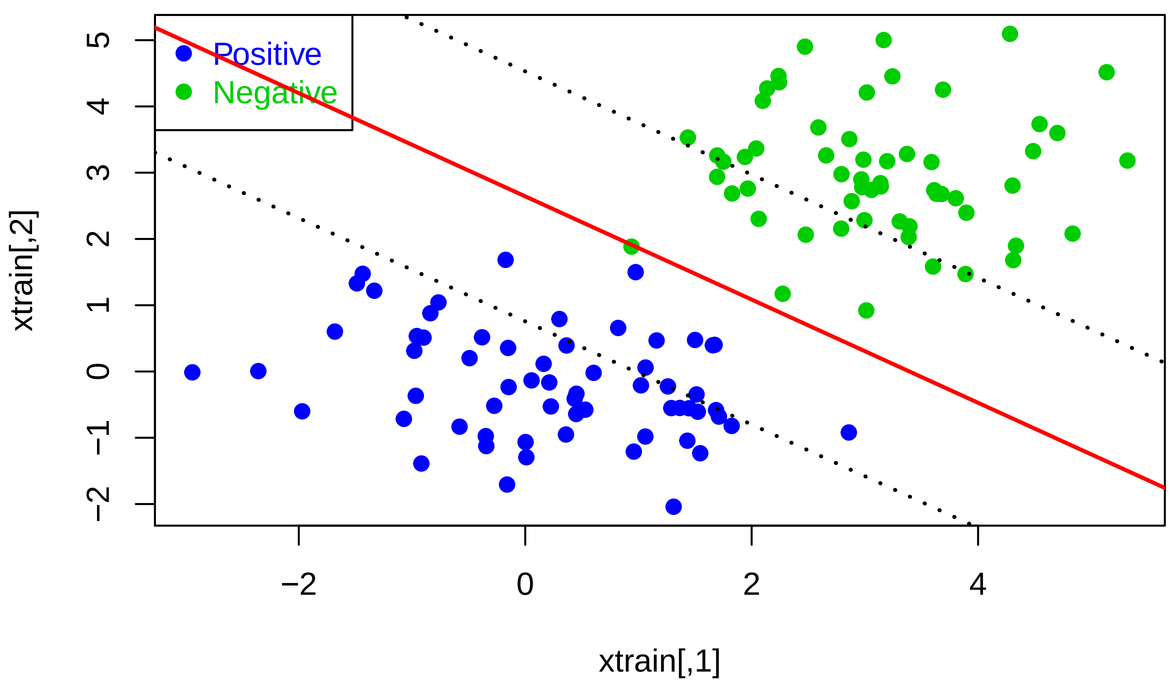
Question 5 :

```
cost = 2^(seq(-10, 14, by=5))
for (c in cost){
  svp = ksvm(xtrain, ytrain, type = "C-svc", kernel = "vanilladot", C=c, scaled=c())
  plotlinearsvm(svp, xtrain)
}
```

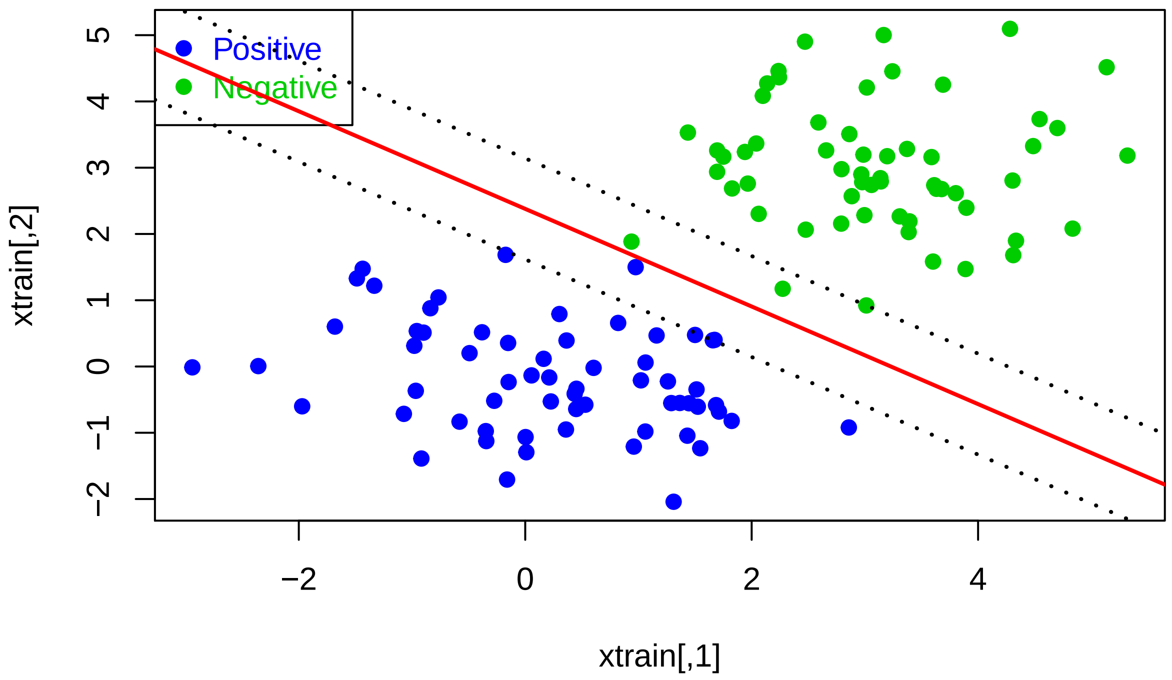
```
##  Setting default kernel parameters
```



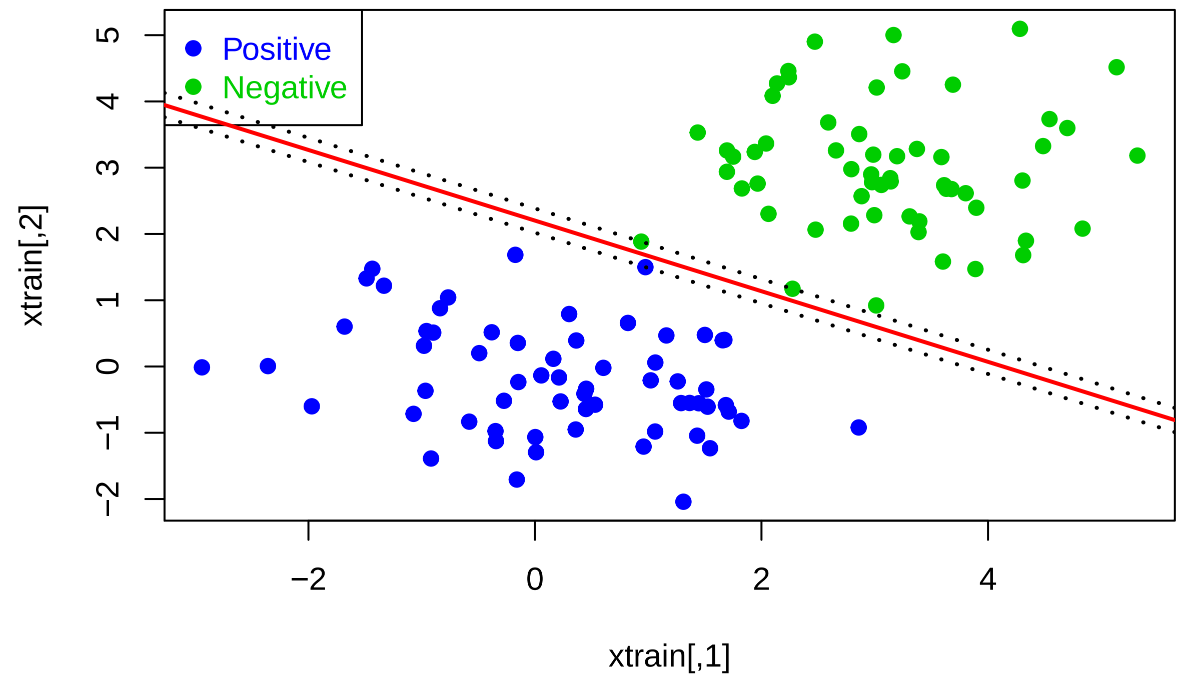
```
##  Setting default kernel parameters
```



```
##  Setting default kernel parameters
```



```
##  Setting default kernel parameters
```



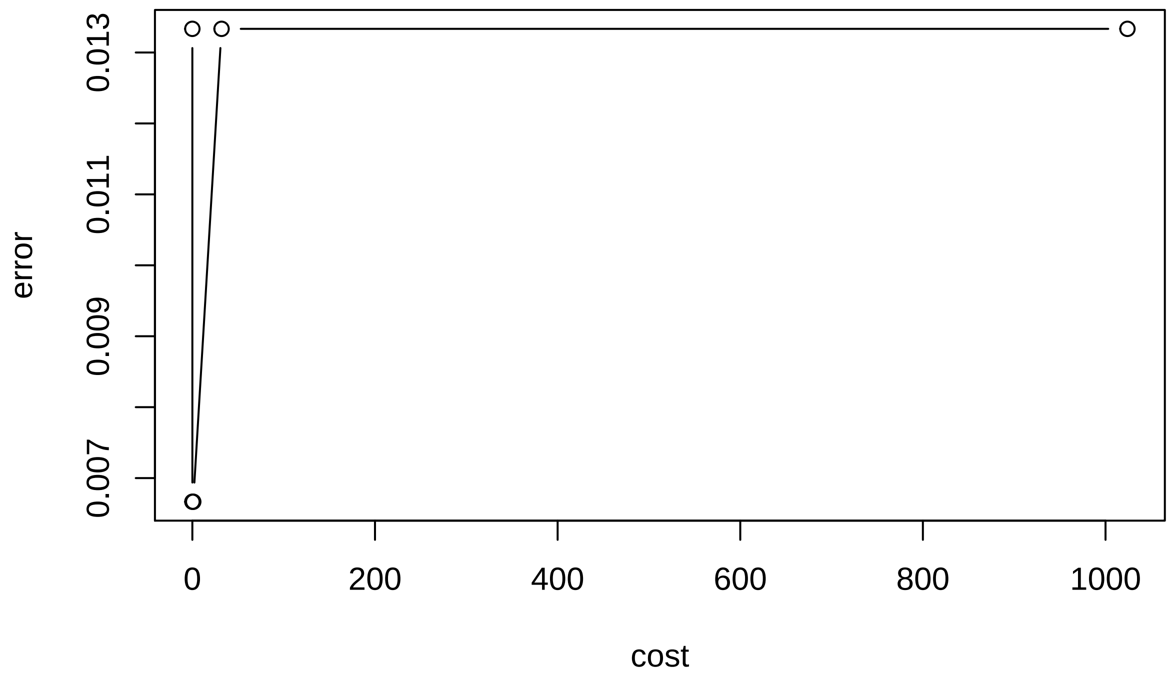
```
##  Setting default kernel parameters
```

Question 6 :

```
error = sapply(cost, function(c){
  svm = ksvm(x, y, type = "C-svc", kernel = "vanilladot", C = c, scaled=c(), cross = 5)
  cross(svm)
})
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
plot(cost, error, type='b')
```

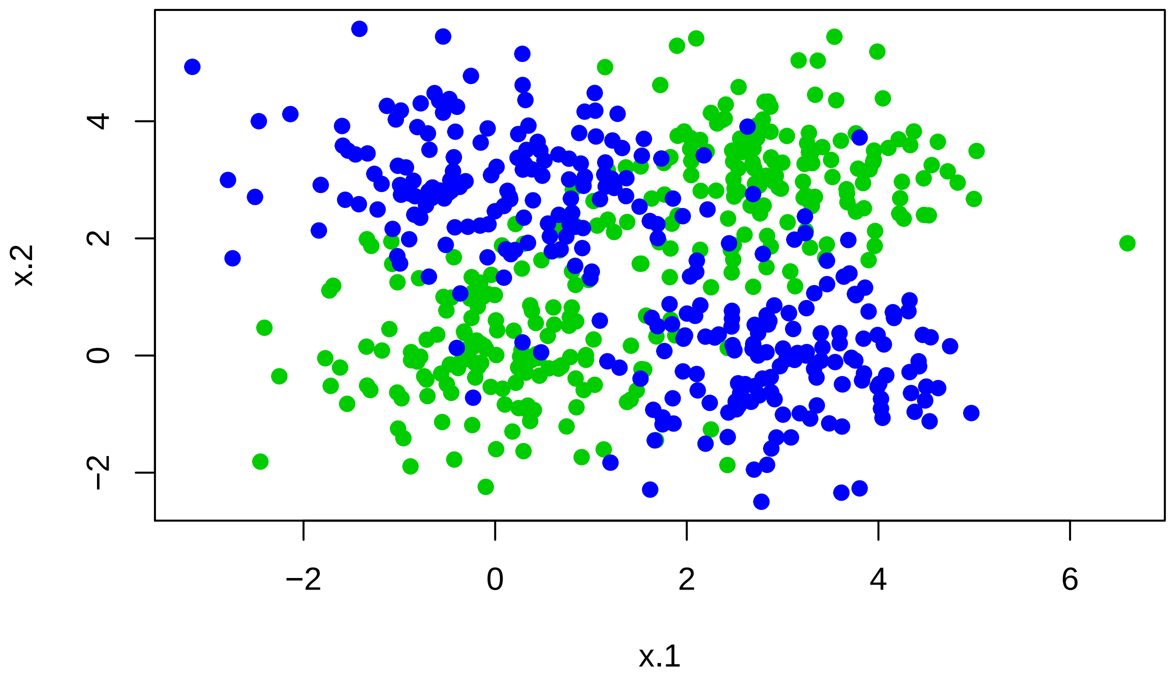


SVM Non Lineaire :

Question 8 :

Generation aléatoire :

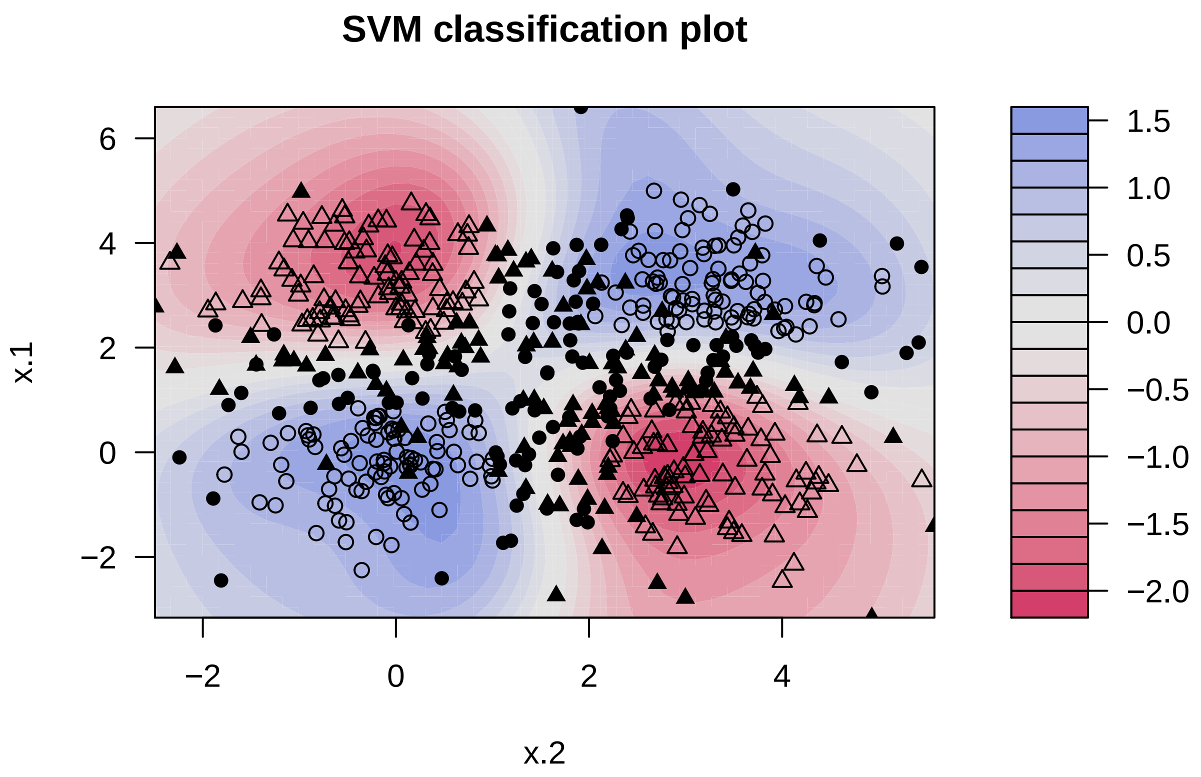
```
p = 2
n = 150
mat1 <- matrix( rnorm(n*p , mean = 0), n, p)
mat2 <- matrix( rnorm(n*p , mean = 3), n, p)
mat3 <- matrix( rnorm(n*p , mean = 0), n, p)
mat4 <- matrix( rnorm(n*p , mean = 3), n, p)
mat5 <- cbind( mat3[,1], mat4[,2] )
mat6 <- cbind( mat4[,1], mat3[,2] )
y <- c( rep( 1, 2 * n ), rep( -1, 2 * n ) )
xtrain <- sample( 300 , 200 )
train <- rep( 0, 300 )
train[xtrain] <- 1
data =data.frame( x=rbind( mat1, mat2, mat5, mat6 ), y=y, train=train )
plot(data[,1:2], col = ifelse(data[,3]<0,4,11) , pch=19 )
```



```
x = as.matrix(data[,1:2])
y = matrix(data[,3])
```

Entrainer un SVM

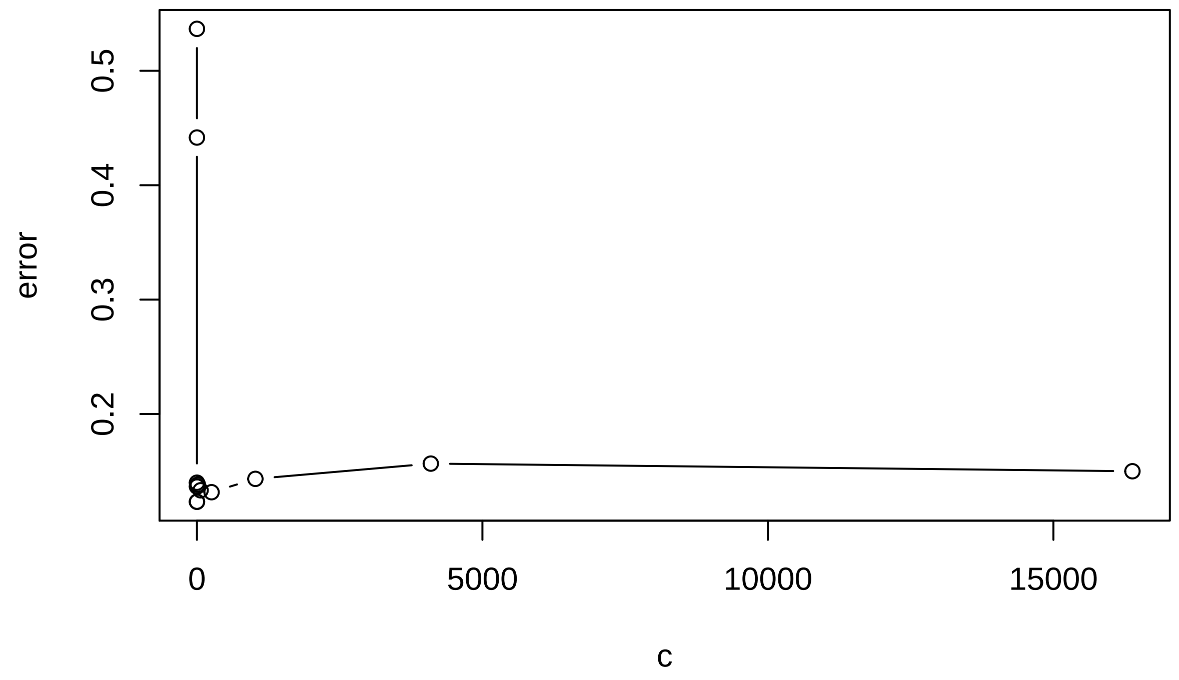
```
svp <- ksvm(x,y,type="C-svc",kernel='rbf',kpar=list(sigma=1),C=0.5)
plot(svp,data=x)
```



Question 10 :

Construire un SVM non linéaire avec différents C avec détermination automatique de ??.

```
c = 2^(seq(-10, 14, by=2))
error = sapply(1:length(c), function(i){
  svp = ksvm(x, y, type = "C-svc", kernel = "rbf", C = c[i], cross = 5)
  cross(svp)
})
plot(c, error, type="b")
```



Question 11 :

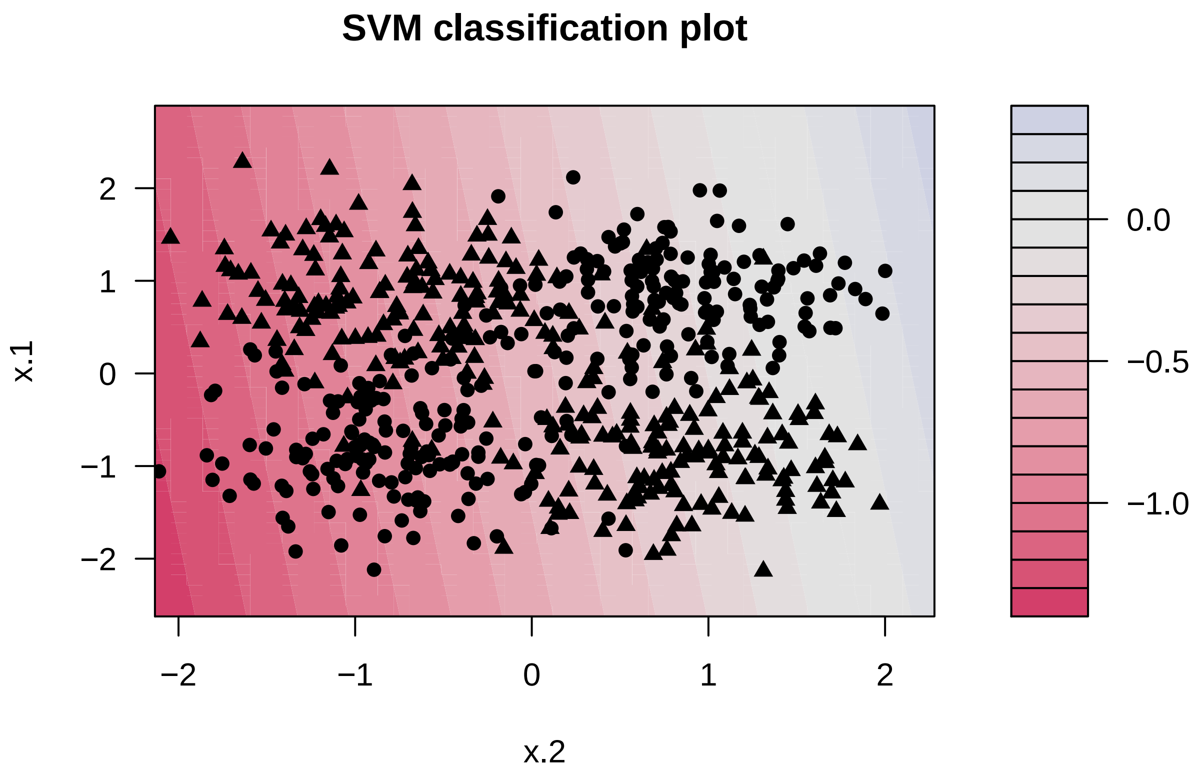
Testons les kernel **polynôme**, la **tangente hyperbolique**, le **Laplacien**, le **Bessel** et **l'ANOVA** sur les exemples de données.

POLYDOT

```
svp =ksvm(x, y, type = "C-svc", kernel = "polydot", C = 1)
```

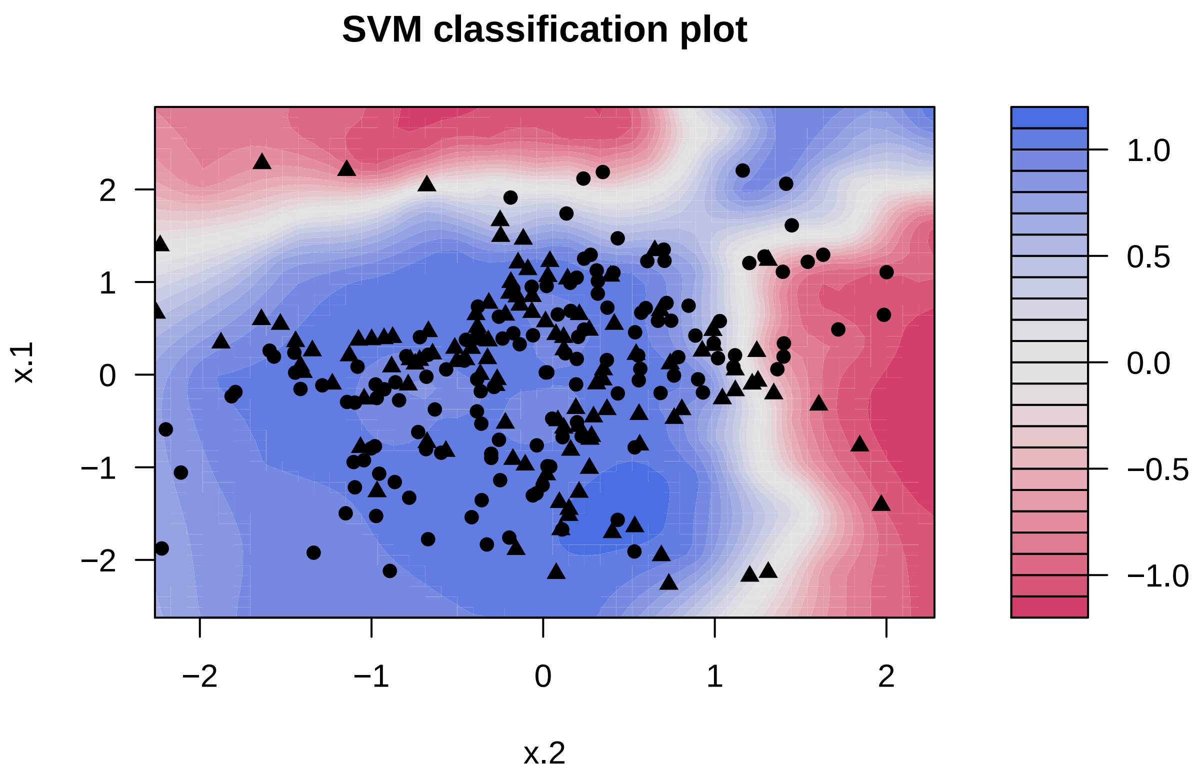
```
##  Setting default kernel parameters
```

```
plot(svp )
```



LAPLACEDOT

```
svp =ksvm(x, y, type = "C-svc", kernel = "laplacedot", C = 1)
plot(svp )
```

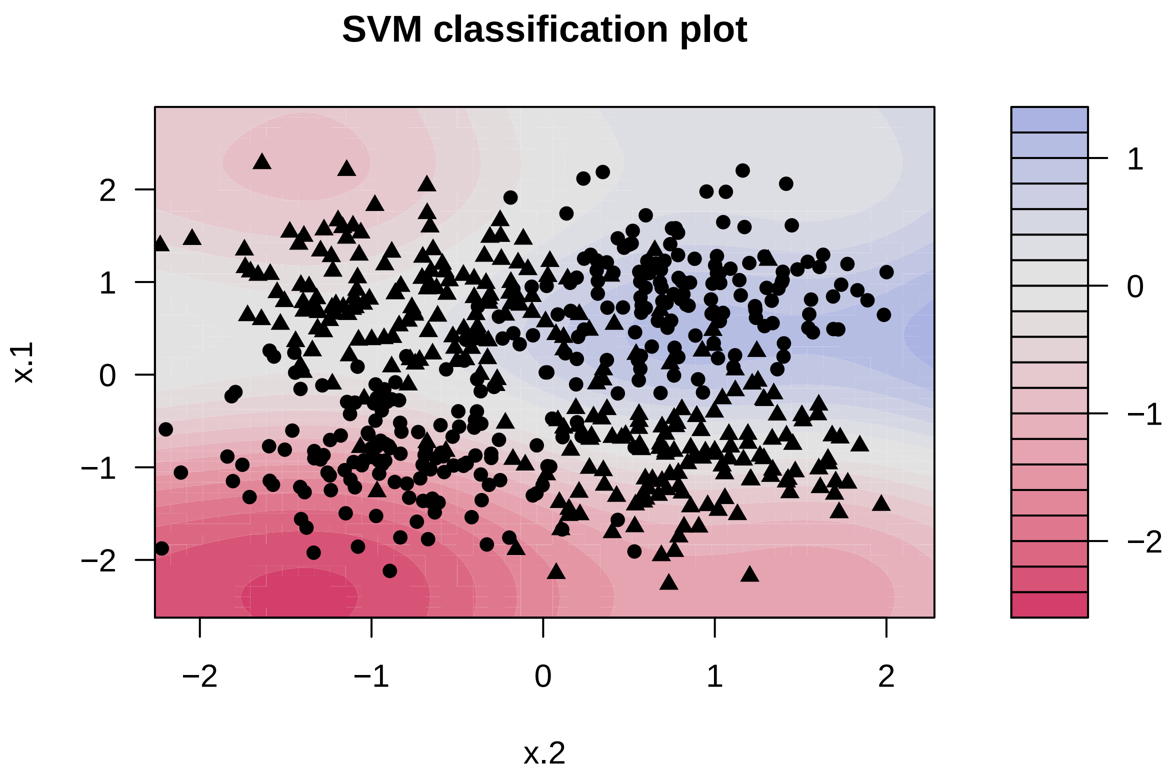


ANOVADOT

```
svp =ksvm(x, y, type = "C-svc", kernel = "anovadot", C = 1)
```

```
##  Setting default kernel parameters
```

```
plot(svp )
```

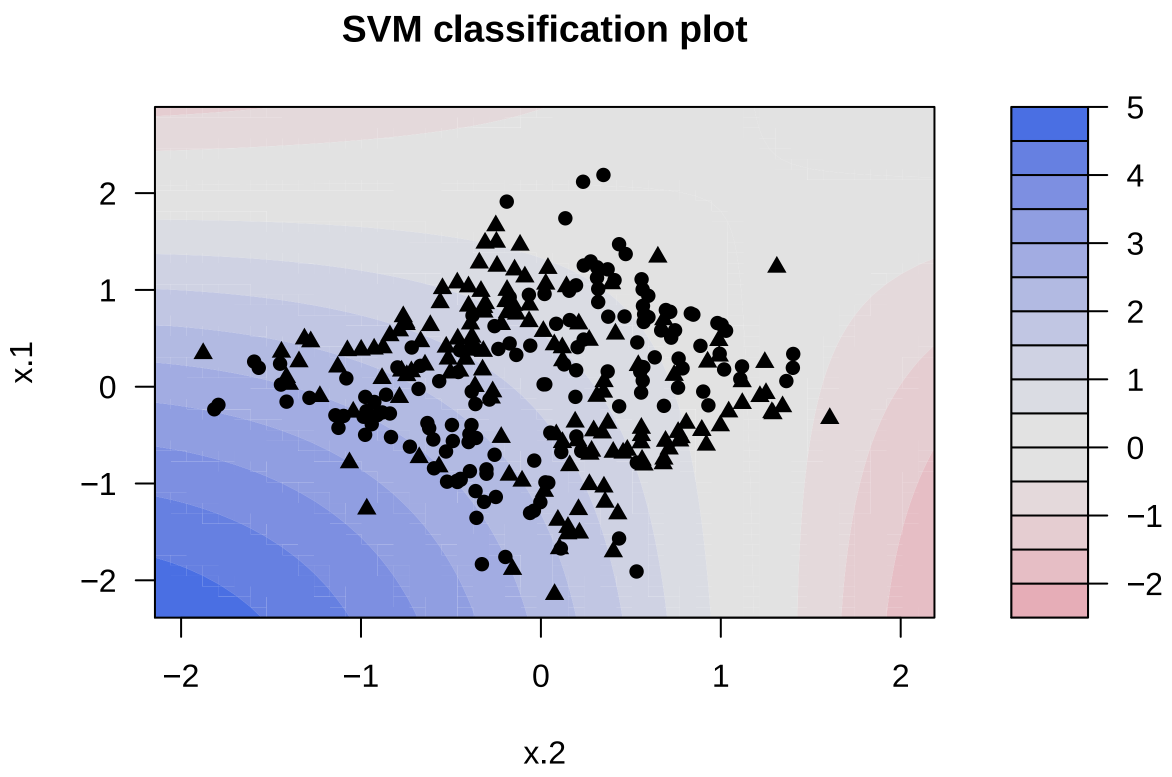


BESSELDOT

```
svp =ksvm(x, y, type = "C-svc", kernel = "besseldot", C = 1)
```

```
##  Setting default kernel parameters
```

```
plot(svp )
```



3- Application : le diagnostic du cancer à partir des données d’expression génique

```r
# Load the ALL dataset
library(ALL)
```

```
## Loading required package: Biobase
```

```
## Loading required package: BiocGenerics
```

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

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, do.call, duplicated, eval, evalq, Filter, Find, get,
##     grep, grepl, intersect, is.unsorted, lapply, lengths, Map,
##     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, Position, rank, rbind, Reduce, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unlist, unsplit

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
library(kernlab)
data(ALL)
# Inspect them
?ALL
```

```
## starting httpd help server ...

##  done
```

```
show(ALL)
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 12625 features, 128 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 01005 01010 ... LAL4 (128 total)
##   varLabels: cod diagnosis ... date last seen (21 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
##   pubMedIds: 14684422 16243790
## Annotation: hgu95av2
```

```
print(summary(pData(ALL)))
```

```
##      cod              diagnosis            sex          age
##  Length:128         Length:128         F   :42   Min.   : 5.00
##  Class :character   Class :character   M   :83   1st Qu.:19.00
##  Mode  :character   Mode  :character   NA's: 3   Median :29.00
##                                                  Mean   :32.37
##                                                  3rd Qu.:45.50
##                                                   Max.   :58.00
##                                                   NA's   :5
##         BT     remission      CR              date.cr
##  B2     :36   CR  :99   Length:128         Length:128
##  B3     :23   REF :15   Class :character   Class :character
##  B1     :19   NA's:14   Mode  :character   Mode  :character
##  T2     :15
##  B4     :12
##  T3     :10
##  (Other):13
##   t(4;11)         t(9;22)        cyto.normal        citog
##  Mode :logical   Mode :logical   Mode :logical   Length:128
##  FALSE:86        FALSE:67        FALSE:69        Class :character
##  TRUE :7         TRUE :26        TRUE :24        Mode  :character
##  NA's :35        NA's :35        NA's :35
##
##
##
##      mol.biol    fusion protein   mdr          kinet       ccr
##  ALL1/AF4:10   p190     :17     NEG :101   dyploid:94   Mode :logical
##  BCR/ABL :37   p190/p210: 8     POS : 24   hyperd.:27   FALSE:74
##  E2A/PBX1: 5   p210     : 8     NA's:  3   NA's   : 7   TRUE :26
##  NEG     :74   NA's     :95                             NA's :28
##  NUP-98  : 1
##  p15/p16 : 1
##
##   relapse        transplant          f.u             date last seen
##  Mode :logical   Mode :logical   Length:128         Length:128
##  FALSE:35        FALSE:91        Class :character   Class :character
##  TRUE :65        TRUE :9         Mode  :character   Mode  :character
##  NA's :28        NA's :28
##
##
##
```

```
x <- t(exprs(ALL))
y <- substr(ALL$BT,1,1)
```

Question 12 :

Division des données en ensembles d'entrainement et test :

```
n = length(y)
train_set <- n * 0.75
train_set
```

```
## [1] 96
```

```
train_index <- sample(n, train_set)
train_index
```

```
##   [1]  17   6  39  38  24  62  50   4  35  13  27  32 102  42  74  29 103
## [18]  46  54  98  81  80  73  40 123  82 126 110 116 128 109  15 112  53
## [35]  65 119  25  28  76  79  56  20 118  52  69  63 111  11 124  72 101
## [52] 115  22  77  44  16  68  78   8 125  31  70  10 121   5  67   9  61
## [69]  71 108 117 100 120  75  89  18  87  64  97  19  92  94 106  90  85
## [86]  59  91  96 104 122 105  26  58  95  60  23
```

```
xtrain <- x[train_index, ]
xtest <- x[-train_index, ]
ytrain <- y[train_index]
ytrain
```

```
##  [1] "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "T" "B" "B" "B" "T"
## [18] "B" "B" "T" "B" "B" "B" "B" "T" "B" "T" "T" "T" "T" "T" "B" "T" "B"
## [35] "B" "T" "B" "B" "B" "B" "B" "B" "T" "B" "B" "B" "T" "B" "T" "B" "T"
## [52] "T" "B" "B" "B" "B" "B" "B" "B" "T" "B" "B" "B" "T" "B" "B" "B" "B"
## [69] "B" "T" "T" "T" "T" "B" "B" "B" "B" "B" "T" "B" "B" "B" "T" "B" "B"
## [86] "B" "B" "T" "T" "T" "T" "B" "B" "B" "B" "B"
```

```
ytest <- y[-train_index]
ytest
```

```
##  [1] "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B"
## [18] "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "T" "T" "T" "T" "T"
```

Entrainer le model :

```
svm = ksvm(xtrain, ytrain, type = "C-svc", kernel = "vanilladot", C = 10)
```

```
##  Setting default kernel parameters
```

Prédiction et Accuracy:

```
pred = predict(svm, xtest)
pred
```

```
##  [1] B B B B B B B B B B B B B B B B B B B B B B B B B B B T T T T T
## Levels: B T
```

```
table(pred , ytest)
```

```
##     ytest
## pred  B  T
##    B 27  0
##    T  0  5
```

```
accuracy = mean(pred == ytest)
accuracy
```

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

Question 13 :

Enfin, nous pouvons vouloir prédire le type et le stade des maladies. Nous sommes alors confrontés à un problème de classification multi-classe, puisque la variable à prédire peut prendre plus de deux valeurs

```r
y <- ALL$BT
print(y)
```

```
##   [1] B2 B2 B4 B1 B2 B1 B1 B1 B2 B2 B3 B3 B3 B2 B3 B  B2 B3 B2 B3 B2 B2 B2
##  [24] B1 B1 B2 B1 B2 B1 B2 B  B  B2 B2 B2 B1 B2 B2 B2 B2 B2 B4 B4 B2 B2 B2
##  [47] B4 B2 B1 B2 B2 B3 B4 B3 B3 B3 B4 B3 B3 B1 B1 B1 B1 B3 B3 B3 B3 B3 B3
##  [70] B3 B3 B1 B3 B1 B4 B2 B2 B1 B3 B4 B4 B2 B2 B3 B4 B4 B4 B1 B2 B2 B2 B1
##  [93] B2 B  B  T  T3 T2 T2 T3 T2 T  T4 T2 T3 T3 T  T2 T3 T2 T2 T2 T1 T4 T
## [116] T2 T3 T2 T2 T2 T2 T3 T3 T3 T2 T3 T2 T
## Levels: B B1 B2 B3 B4 T T1 T2 T3 T4
```

On utilise le type **kbb-svc** pour un SVM multi_classe :

```r
svm = ksvm(xtrain, ytrain, type = "kbb-svc", kernel = "rbf", C = 10)
pred = predict(svm, xtest)
table(pred , ytest)
```

```
##     ytest
## pred  B  T
##    B 27  0
##    T  0  5
```

```r
accuracy = mean(pred == ytest)
accuracy
```

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