TP/DM 1 : APP STATS

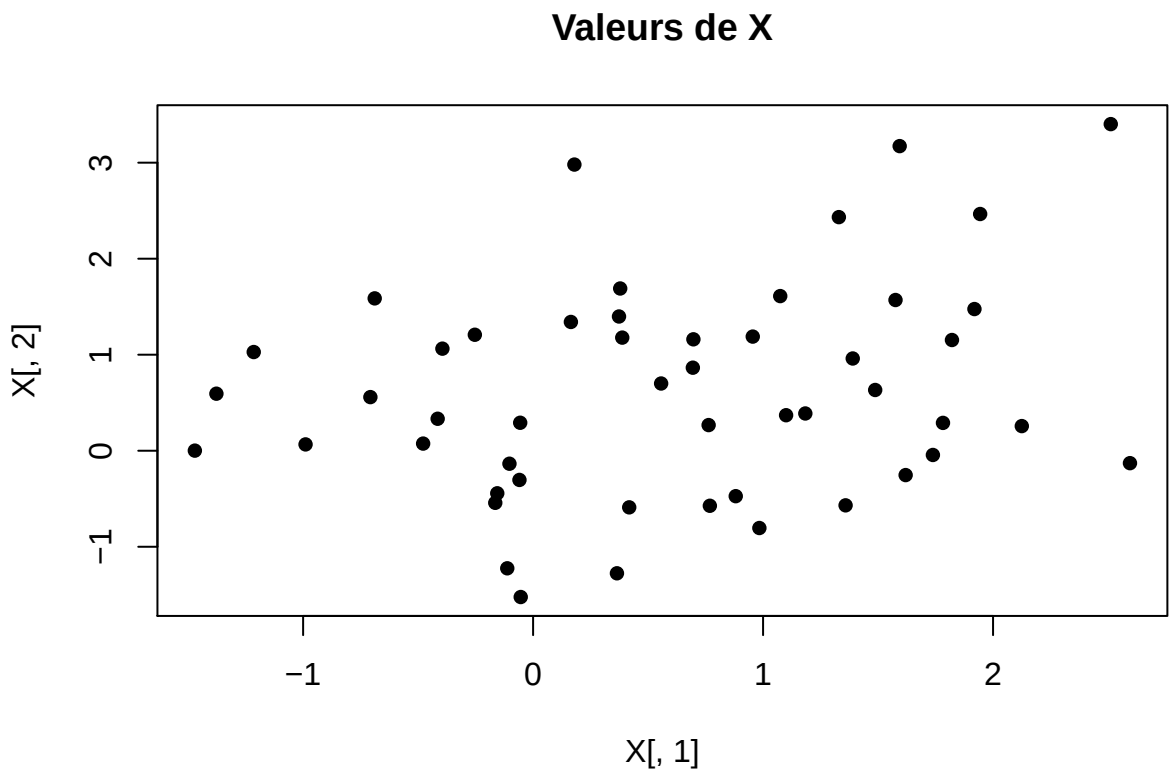
L'objectif de ce TP est d'implémenter les classifieurs à vecteurs de support (SVM) sur des données simulées. Avant de commencer, vous pouvez lire les chapitres 9.1, 9.2 et 9.3 du livre de G. James, D. Witten, T. Hastie et J. Tibshirani, 'An introduction to Statistical Learning'.

Vous pouvez répondre aux questions de ce DM :

- soit en insérant vos réponses à la suite des questions dans le fichier RMarkdown.
- soit dans un document texte contenant votre code R commenté.

Génération des données

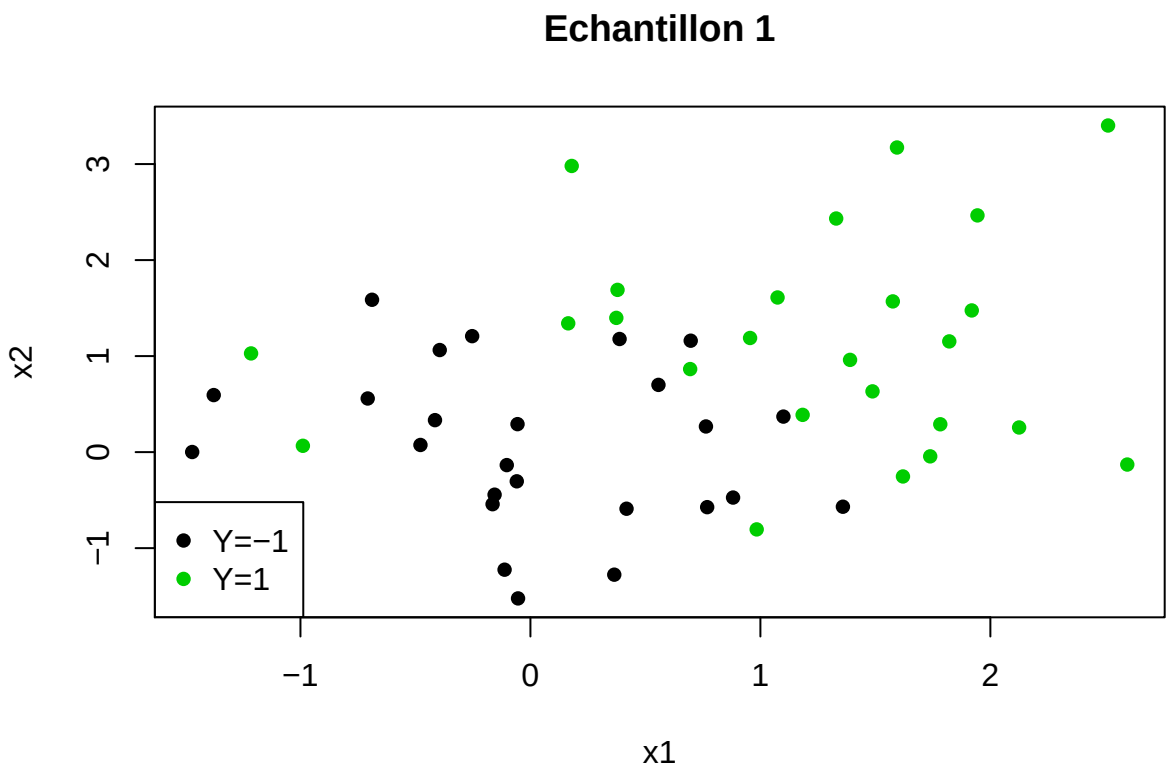
```
set.seed(1)
n = 50 # nombre d'individus
p = 2 # dimension
X = matrix(rnorm(p*n),ncol=p)
Y1 = c(rep(1,n/2),rep(-1,n/2))
X[Y1==1,] = X[Y1==1,]+1
plot(X[,1],X[,2],main="Valeurs de X",pch=16)
```



- *Question 1 :* Parmis les échantillons suivants lesquels sont linéairement séparables ?

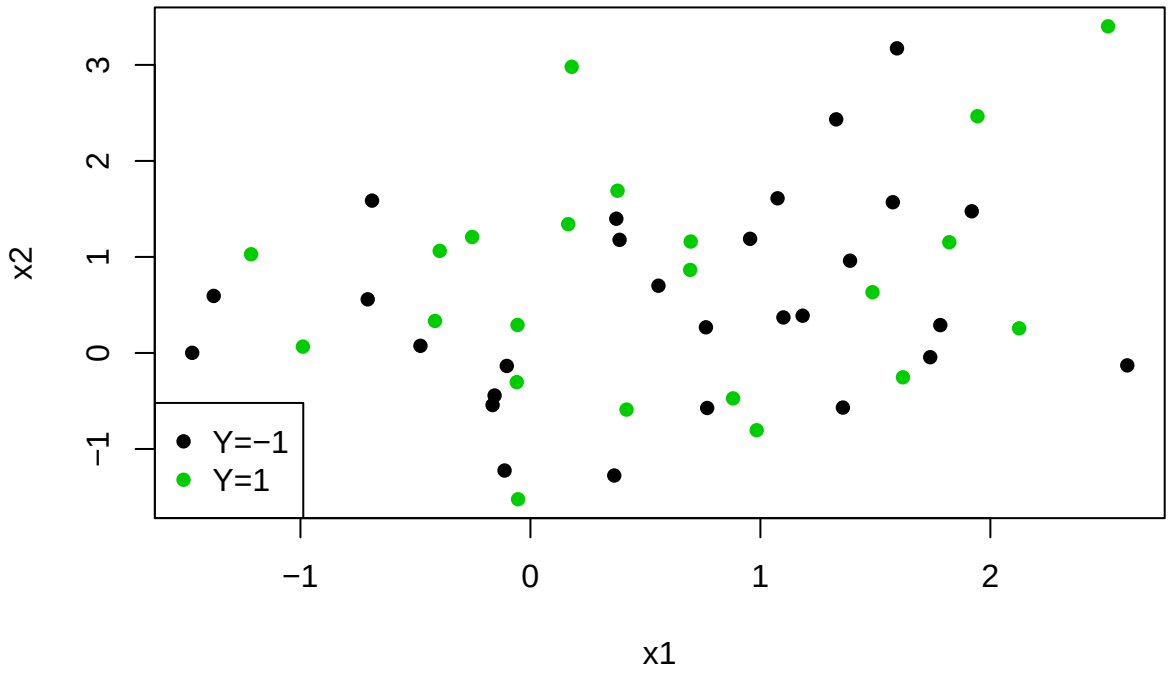
Le seul échantillon liénairement séparable est le 3ème. Les premiers et seconds ne le sont pas (cf détail et justification sous les plots).

```
plot(X[,1],X[,2],main="Echantillon 1",pch=16,col=Y1+2,xlab="x1",ylab="x2")
legend("bottomleft",c("Y=-1","Y=1"),pch=rep(16,2),col=c(1,3))
```



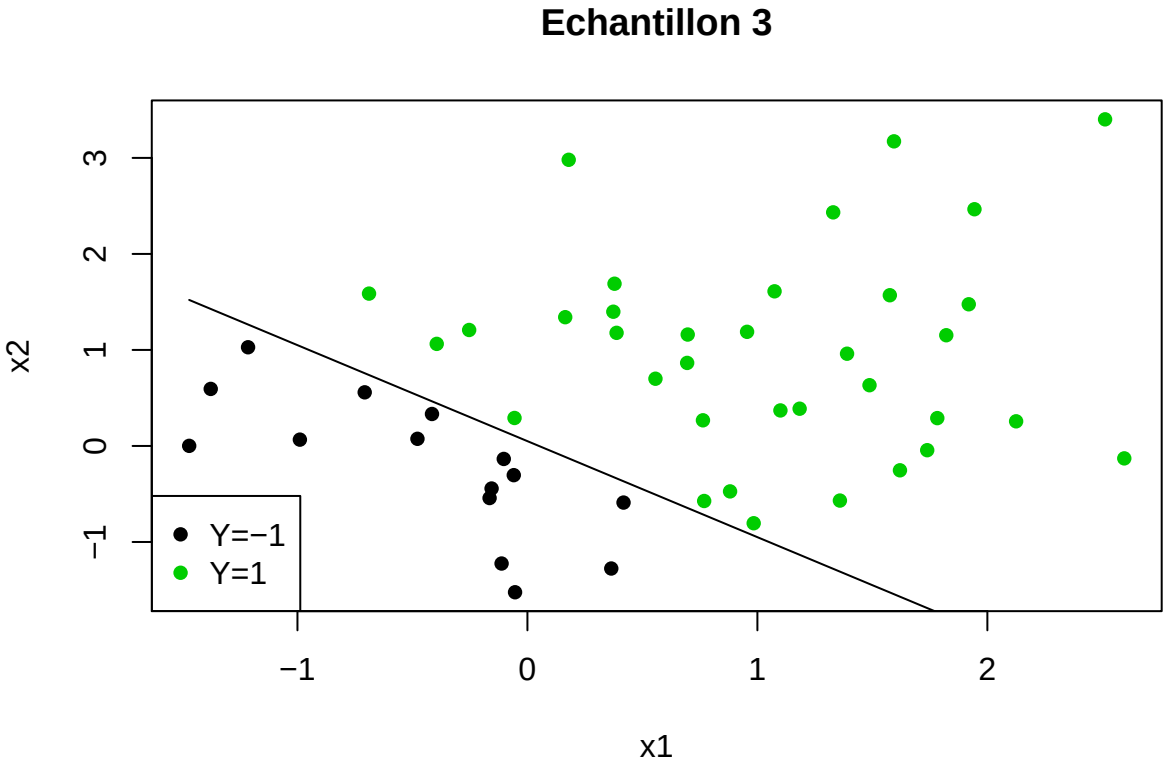
On distingue une lègére tendance : les “positifs” sont plus en haut à gauche et les “négatifs” en bas à droite. Cependant, il est impossible de séparer cet échantillon avec un hyperplan (droite en l’occurence). Il serait tout de même possible de séparer les deux cluster sans pour autant être totalement juste (du moins, avec un hyperplan).

```
Y2 = 2*round(runif(n))-1
plot(X[,1],X[,2],main="Echantillon 2",pch=16,col=Y2+2,xlab="x1",ylab="x2")
legend("bottomleft",c("Y=-1","Y=1"),pch=rep(16,2),col=c(1,3))
```



On ne distingue aucune tendance, les “positifs” et les “négatifs” sont totalements mélangés. Il est impossible de séparer liénairement cet échantillon.

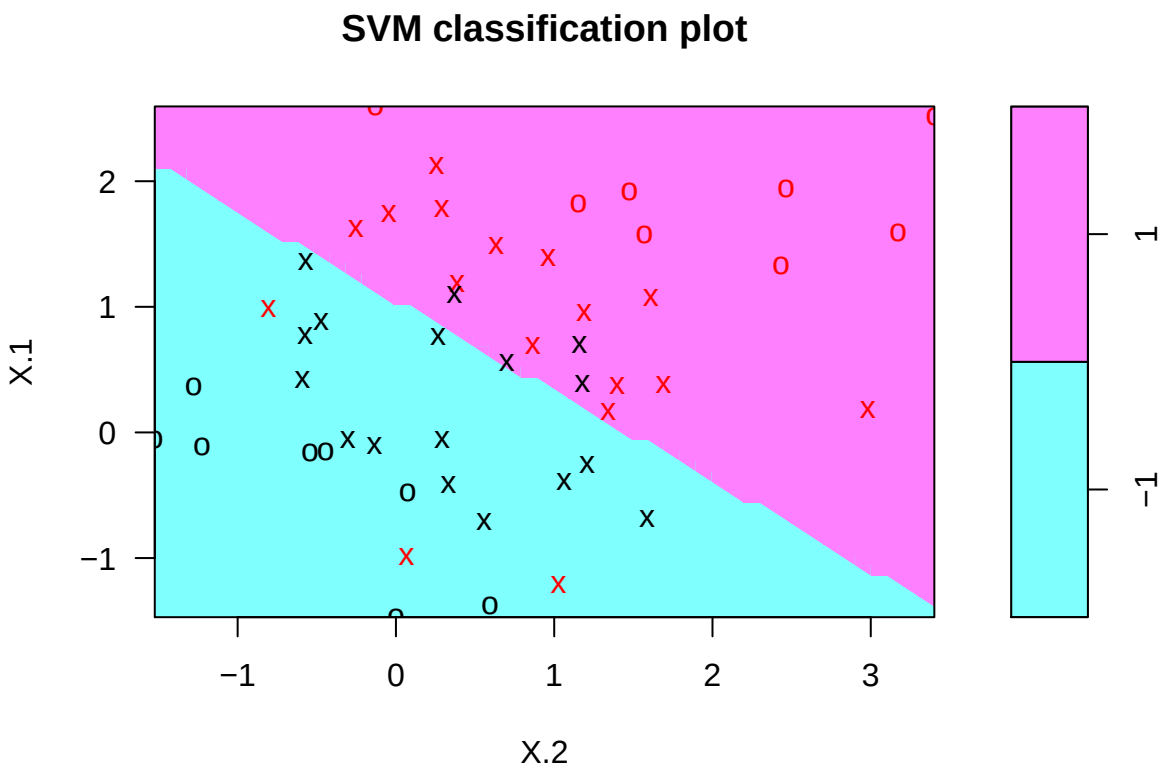
```
Y3 = 2*(X[,1]+X[,2]>0)-1
plot(X[,1],X[,2],main="Echantillon 3",pch=16,col=Y3+2,xlab="x1",ylab="x2")
legend("bottomleft",c("Y=-1","Y=1"),pch=rep(16,2),col=c(1,3))

curve(-x+0.05,add=TRUE)
```



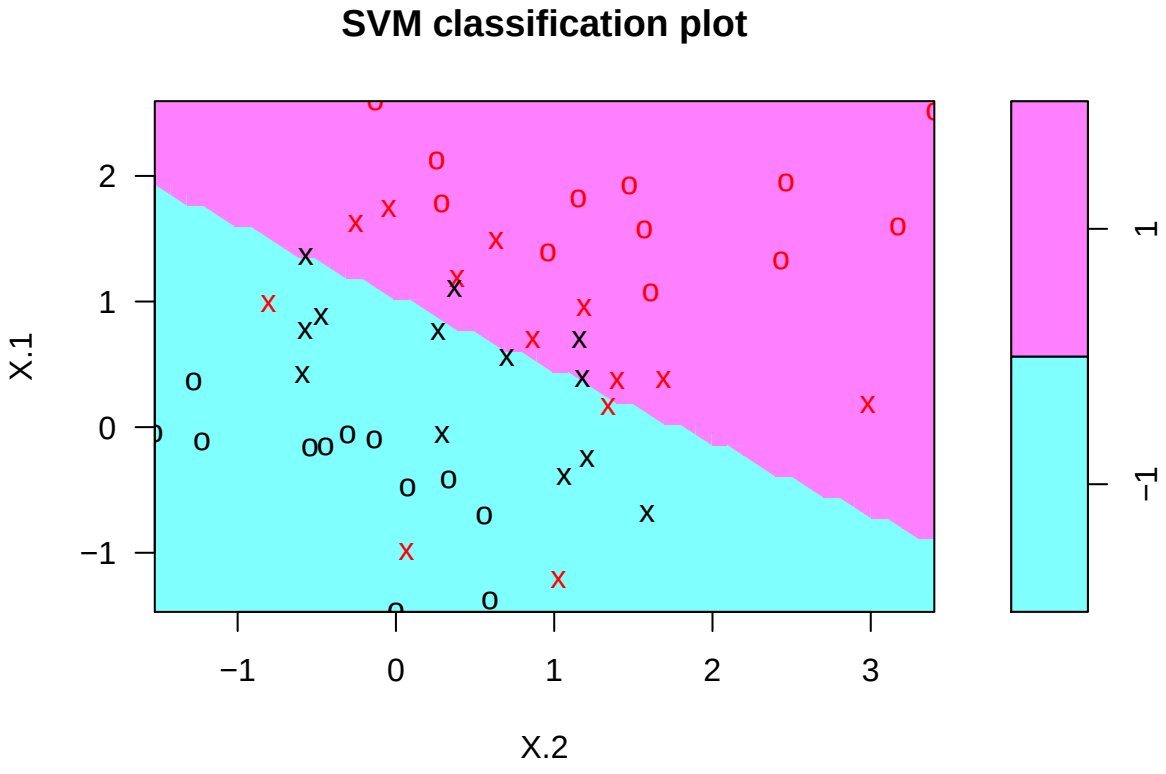
Ici l'échantillon est clairement séparable liénairement comme le montre la droite (non optimale vis à vis d'une marge maximale) traçée "à la main".

SVM

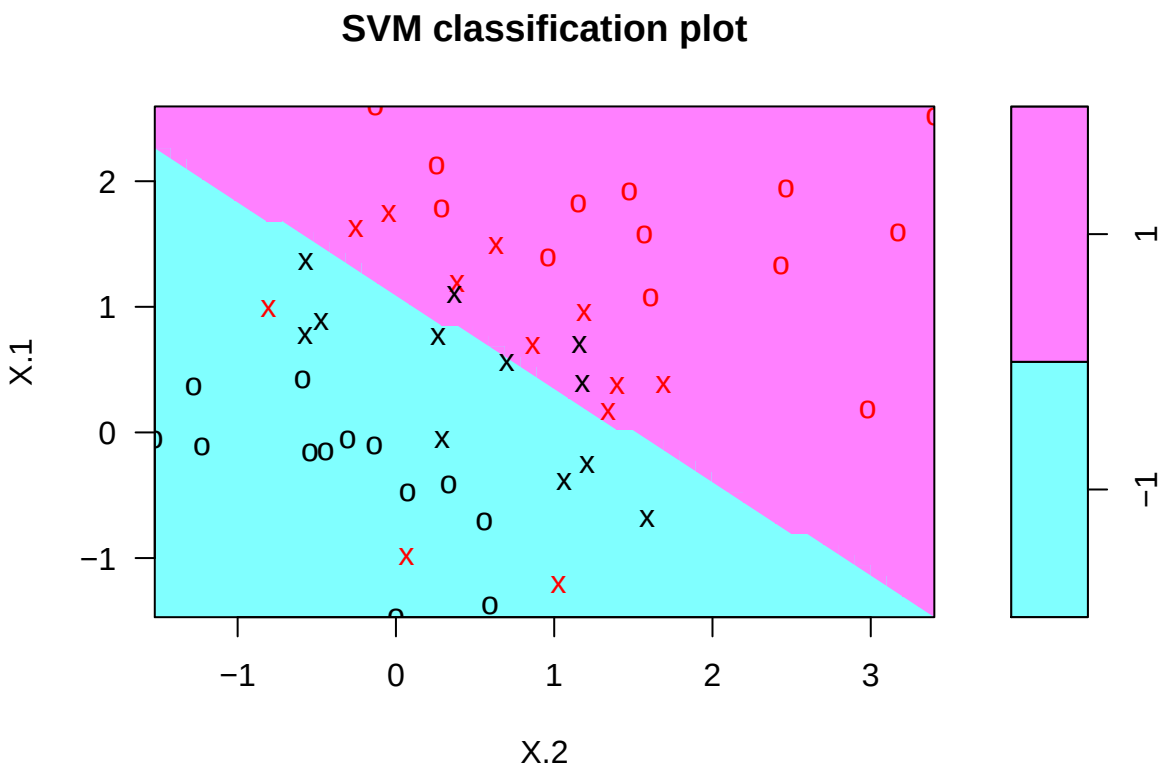
```
#install.packages("e1071") # ligne a decommenter si le package n'a pas ete installe
library(e1071)
dat1 = data.frame(X=X,Y=as.factor(Y1))

#On garde la ligne donnée, elle est utile par la suite :
svmfit1.lin = svm(Y~.,data=dat1,kernel="linear",cost=10) # SVM lineaire (kernel="linear") avec fonction

#On construit les différents SVM avec des fonctions de coût différentes :
svmfit1.lin.C01 = svm(Y~.,data=dat1,kernel="linear",cost=0.1) # SVM lineaire
                                                #(kernel="linear") avec fonction de cout C=0.1
svmfit1.lin.C05 = svm(Y~.,data=dat1,kernel="linear",cost=0.5) # SVM lineaire
                                                #(kernel="linear") avec fonction de cout C=0.5
svmfit1.lin.C1 = svm(Y~.,data=dat1,kernel="linear",cost=1) # SVM lineaire
                                                #(kernel="linear") avec fonction de cout C=1
svmfit1.lin.C5 = svm(Y~.,data=dat1,kernel="linear",cost=5) # SVM lineaire
                                                #(kernel="linear") avec fonction de cout C=5
svmfit1.lin.C20 = svm(Y~.,data=dat1,kernel="linear",cost=20) # SVM lineaire
                                                #(kernel="linear") avec fonction de cout C=20
svmfit1.lin.C50 = svm(Y~.,data=dat1,kernel="linear",cost=50) # SVM lineaire
                                                #(kernel="linear") avec fonction de cout C=50
plot(svmfit1.lin.C01,dat1, main = "cout C = 0.1")
```



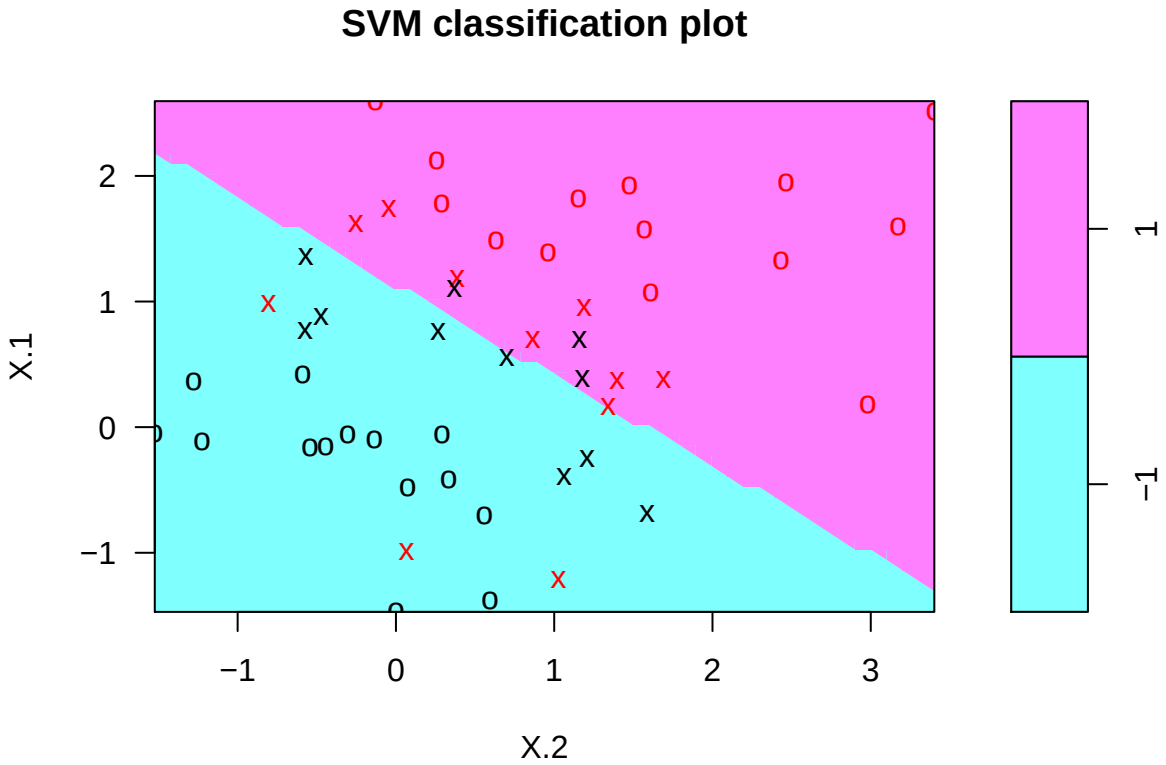

```
plot(svmfit1.lin.C05,dat1, main = "cout C = 0.5")
```




```
plot(svmfit1.lin.C1,dat1, main = "cout C = 1")
```




```
plot(svmfit1.lin.C5,dat1, main = "cout C = 5")
```




```
plot(svmfit1.lin.C20,dat1, main = "cout C = 20")
plot(svmfit1.lin.C50,dat1, main = "cout C = 50")
```

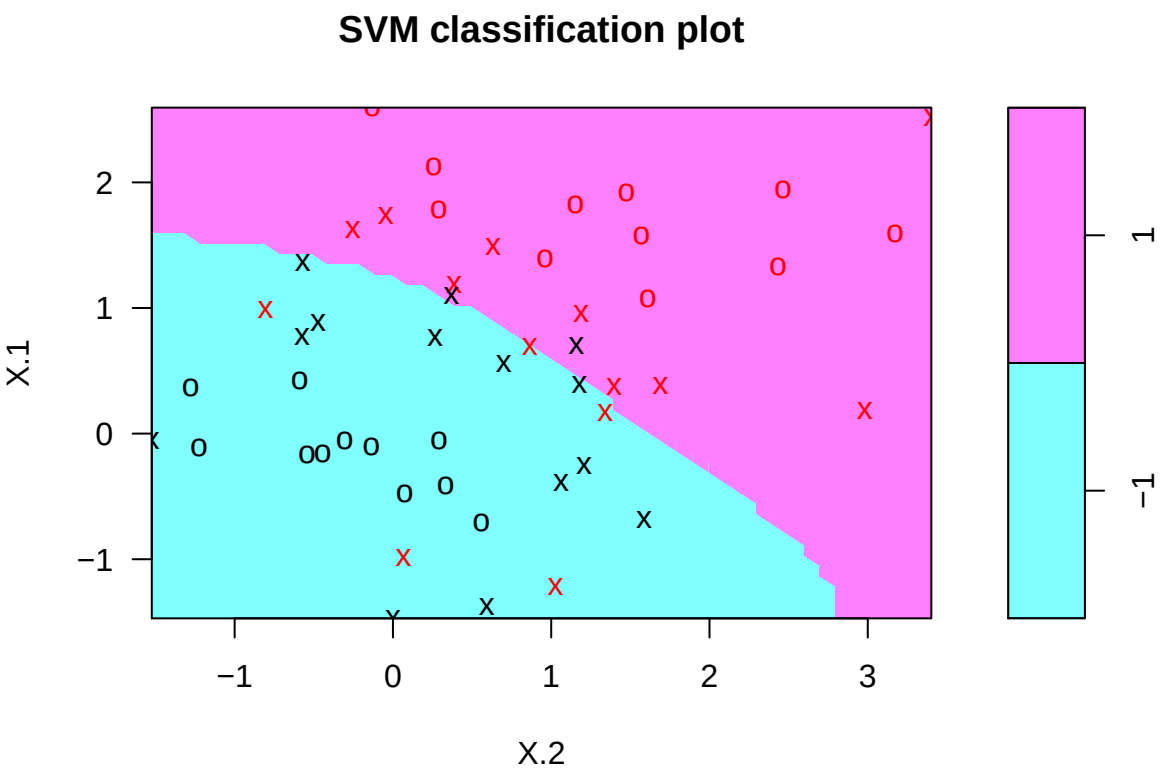
Les croix représentent les vecteurs de support. Faire varier la fonction de coût et observer.

Il semblerait que la "limite" fasse moins de "sauts" avec une fonction de coût égale à 1.

- *Question 2 :* On comparera avec le résultat sans préciser les options `kernel` et `cost`. Quel noyau la fonction `svm()` a-t'elle choisi ? Quelles est la valeur du paramètre de coût ? Comments ces valeurs ont-elles été choisies ?

R utilise les paramètres suivants par défaut : kernel = 'radial', cost = '1'. le noyau par défaut est donc 'radial' (ou RBF), c'est un des plus utilisés et il agit comme un comparateur de "similarité" entre les points. le coût par défaut est ici de '1' et sert de borne inférieure. Il est généralement recommandé d'augmenter ce coût.

```
svmfit1.def = svm(Y~.,data=dat1)
plot(svmfit1.def,dat1)
```




```
summary(svmfit1.def)
```

```
##
## Call:
## svm(formula = Y ~ ., data = dat1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  28
##
##  ( 14 14 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  -1 1
```

Prédiction à l'aide de la fonction `svm()`

Nous pouvons faire des prédictions à l'aide de la fonction `svm()` de la manière suivante :

```
#set.seed(1)
ntest = 50 # nombre d'individus dans l'échantillon de test
Xtest = matrix(rnorm(p*ntest),ncol=p)
Y1test = c(rep(1,ntest/2),rep(-1,ntest/2))
Xtest[Y1test==1,] = Xtest[Y1test==1,]+1
dat.test = data.frame(X=Xtest,Y=as.factor(Y1test))
Y1pred.lin = predict(svmfit1.lin,dat.test)
table(Y1test,Y1pred.lin)
```

```
##       Y1pred.lin
## Y1test -1  1
##     -1 21  4
##     1  13 12
```

- *Question 3 :* Faire de même avec le classifieur `svmfit1.def` estimé par défaut par la fonction `svm()` et comparer les résultats.

Le code suivant donne en sortie les résultats :

```
#Mise de la table résultats du SVM linéaire dans une variable
results_linear_svm = table(Y1test,Y1pred.lin)

#SVM par défaut
Y1pred.def = predict(svmfit1.def,dat.test)
results_default_svm = table(Y1test,Y1pred.def)

#Calcul des taux de prédictions :
tx.global <- function(table){
  return((table[1,1]+table[2,2])/(sum(table)))
}

tx.class <- function(table){
  liste <- list(negatif = numeric(1), positif = numeric(1))
  liste$negatif = table[1,1]/(sum(table[1,]))
  liste$positif = table[2,2]/(sum(table[2,]))
  return(liste)
}

Default_class <- tx.class(results_default_svm)
Lin_class <- tx.class(results_linear_svm)

cat("taux de prédiction du SVM linéaire :", "\n",
    "global = ", tx.global(results_linear_svm),"%",
    "\n","par classe : ", "negatif =", Lin_class$negatif,
    "%", " // positif = ", Lin_class$positif,"%", "\n","\n")
```

```
## taux de prédiction du SVM linéaire :
```

```
##  global =  0.66 %
##  par classe :  negatif = 0.84 %  // positif =  0.48 %
##
```

```
cat("taux de prédiction du SVM defaut :", "\n",
    "global = ", tx.global(results_default_svm),"%",
    "\n","par classe : ", "negatif =", Default_class$negatif,
    "%", " // positif = ", Default_class$positif,"%", "\n")
```

```
## taux de prédiction du SVM defaut :
##  global =  0.64 %
##  par classe :  negatif = 0.8 %  // positif =  0.48 %
```

Il semblerait que le SVM linéaire soit ici meilleur pour le jeux de données que l'on cherche à classifier.