TP - SVM

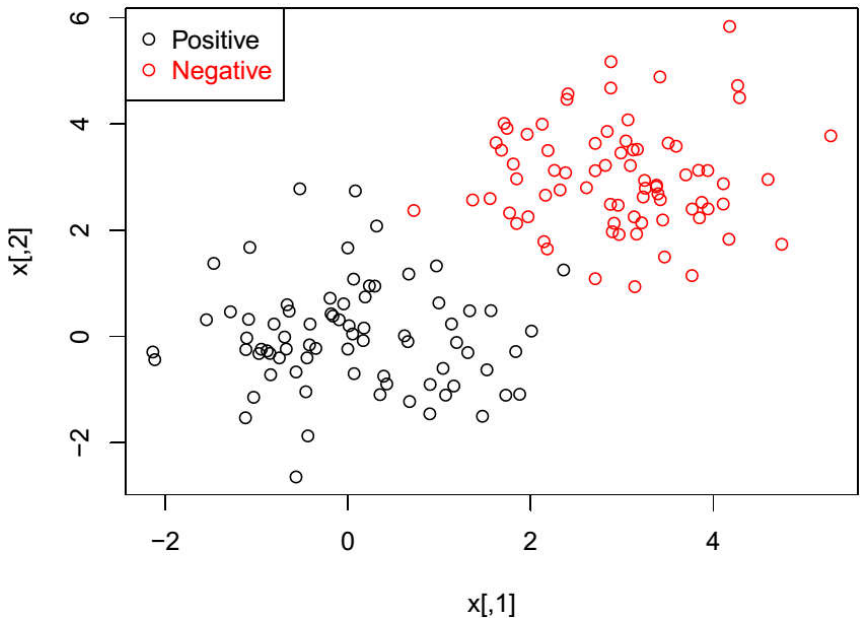
Problème : SVM étude de cas (à Rendre)

1- SVM Linéaire

1.1. Génération de jeu de données

D'abord on génère un ensemble d'exemples positifs et négatifs de 2 Gaussiens.

```
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%)

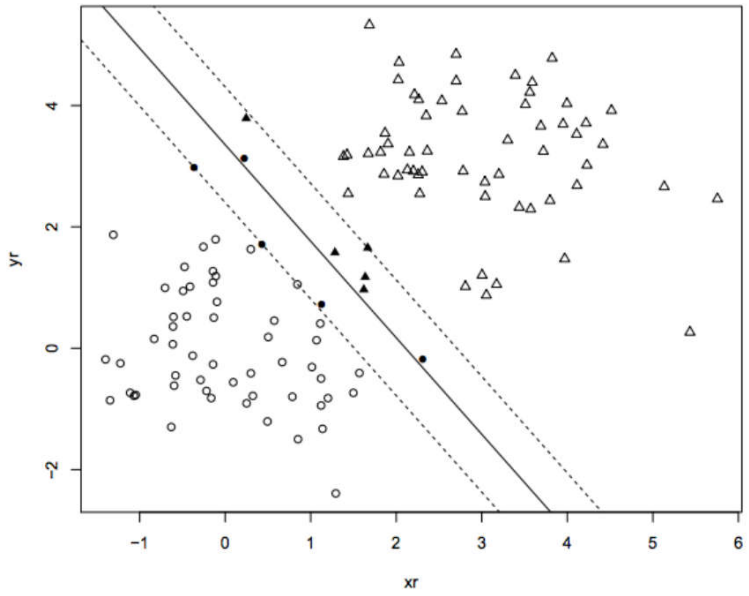
```
# Prepare a training and a test set
ntrain <- round(n * 0.8) # number of training examples
tindex <- sample(n, ntrain) # indices of training samples
xtrain <- x[tindex, ]
xtest <- x[-tindex, ]
ytrain <- y[tindex]
ytest <- y[-tindex]
istrain <- rep(0, n)
istrain[tindex] <- 1

# 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
```

1.2. Entrainer le SVM

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

```
# load the kernlab package
# install.packages("kernlab")
library(kernlab)

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

#Look and understand what svp contains
# General summary
svp

# Attributes that you can access
attributes(svp)

# For example, the support vectors
alpha(svp)
alphaindex(svp)
b(svp)
# Use the built-in function to pretty-plot the classifier
plot(svp, data = xtrain)
```

Question 1 : Ecrire une fonction **plotlinearsvm = function (svp, xtrain)** pour tracer les points et les limites de décision d'une SVM linéaire, comme montré dans la figure ci-dessous. Pour ajouter une droite à une courbe, vous pouvez utiliser la fonction **abline**.



1.3. Prédire avec un SVM

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)
```

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

# 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)

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

```
pred <- prediction(ypredscore, ytest)

# Plot ROC curve
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
# Plot precision/recall curve
perf <- performance(pred, measure = "prec", x.measure = "rec")
plot(perf)
# Plot accuracy as function of threshold
perf <- performance(pred, measure = "acc")
plot(perf)
```

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

Question 2 : Ecrire une fonction **cv.ksvm = function (x, y, folds = 3, ...)** qui renvoie un vecteur **ypred** du score de décision prédit pour tous les points par **k-fold cross-validation**.

Question 3 : Calculez les différentes performances du **SVM** par 5 fois la validation croisée. Alternativement, la fonction **ksvm** peut calculer automatiquement la précision de la validation croisée **k-fold**:

```
svp <- ksvm(x, y, type = "C-svc", kernel = "vanilladot", C = 100, scaled=c(), cross = 5)
print(cross(svp))
print(error(svp))
```

Question 4 : Comparez la **validation croisée 5-fold** estimé par votre fonction et **ksvm**.

1.4. Effet d C

Les paramètres C équilibrent le compromis entre avoir une grande marge et séparer les données positifs et le non étiquetés sur l'ensemble d'apprentissage. Il est important de le bien choisir pour avoir une bonne généralisation.

Question 5 : Tracez les fonctions de décision de SVM entraînées sur le jeu de données pour différentes valeurs de C dans la plage $\mathbf{2}^{\mathbf{seq\,(-10,14)}}$. Pour visualiser les différents tracés, vous pouvez utiliser la fonction par (**ask = T**) qui vous demandera d'appuyer sur une touche entre les tracés successifs. Alternativement, vous pouvez utiliser par (**mfrow = c (5,5)**) pour voir toutes les parcelles dans la même fenêtre.

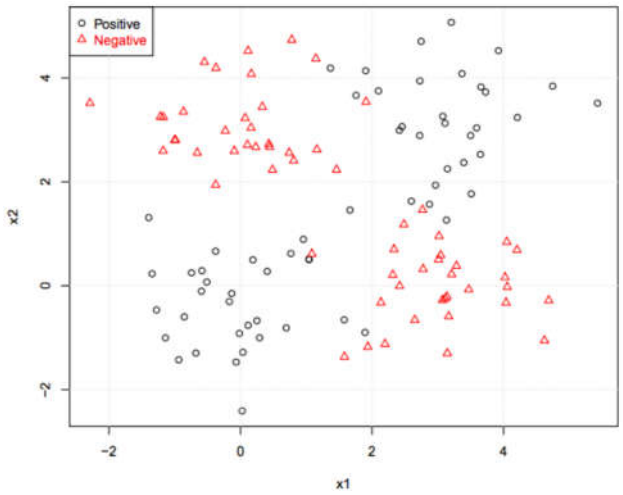
Question 6 : Tracer l'erreur de validation croisée 5 fois en fonction de C.

Question 7 : Faire de même sur les données avec plus de chevauchement entre les deux classes, par exemple, régénérer des jeux données avec le sens étant 1.

2- SVM Non Linéaire

Parfois les SVM linéaire ne sont pas suffisants. Par exemple, générer un jeu de données où les exemples positifs et négatifs sont le mélange de deux gaussiens qui ne sont pas linéairement séparables.

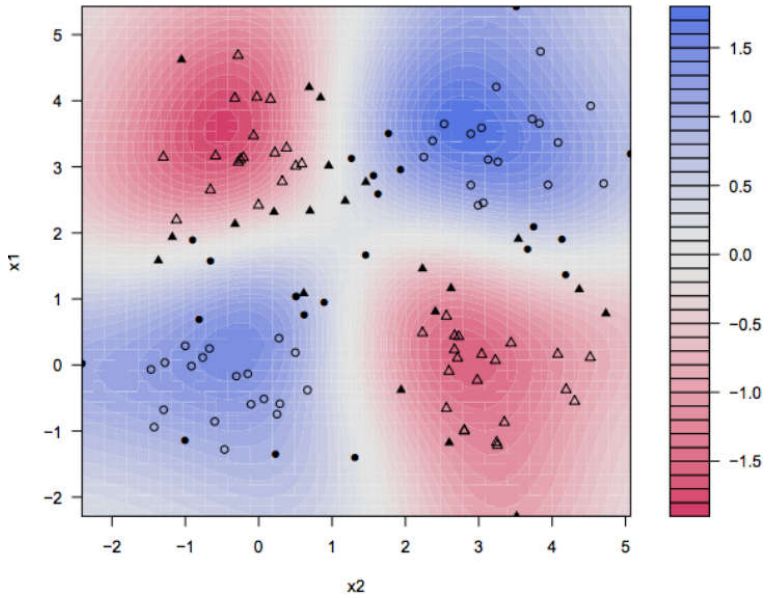
Question 8 : Construire un exemple de données qui ressemble à la figure ci-dessous, et tester un SVM linéaire avec des valeurs différentes de C.



Pour résoudre ce problème, nous devons plutôt utiliser un SVM non linéaire. Ceci est obtenu simplement en changeant le paramètre du noyau. Par exemple, pour utiliser un noyau Gaussien RBF avec σ = 1 et C = 1:

```
# Train a nonlinear SVM
svp <- ksvm(x, y, type = "C-svc", kernel="rbf", kpar = list(sigma = 1), C = 1)

# Visualize it
plot(svp, data = x)
```

Vous devriez obtenir quelque chose qui ressemble à la figure ci-dessous. Beaucoup mieux que le SVM linéaire, non? Le SVM non linéaire a maintenant deux paramètres : σ et C. Les deux jouent un rôle dans la capacité de généralisation de la SVM.



Question 9 : Visualiser et calculer l'erreur de validation croisée 5-fold pour différentes valeurs de C et de σ. Observez leur influence.

Une heuristique utile pour choisir σ est implémentée dans **kernlab**. Il est basé sur les quantiles des distances entre le point d'entraînement.

```
# Train a nonlinear SVM with automatic selection of sigma by heuristic
svp <- ksvm(x, y, type = "C-svc", kernel = "rbf", C = 1)

# Visualize it
plot(svp, data = x)
```

Question 10 : Construire un SVM non linéaire avec différents C avec détermination automatique de σ. En fait, de nombreux autres noyaux non linéaires sont mis en œuvre. Consultez la documentation de **kernlab** pour les consulter : taper la commande **? Kernels**.

Question 11 : Testez le **polynôme**, la **tangente hyperbolique**, le **Laplacien**, le **Bessel** et **l'ANOVA** sur les exemples de données.

3- Mini-Projet : le diagnostic du cancer à partir des données d'expression génique

En tant qu'application du monde réel, testons la capacité des SVM à prédire la classe d'une tumeur à partir des données d'expression génique. Nous utilisons un ensemble de données d'information sur l'expression génétique publiquement disponibles pour 128 individus atteints de leucémie lymphoblastique aiguë (base ALL).

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

# Inspect them
?ALL
show(ALL)
print(summary(pData(ALL)))
```

Nous nous concentrons ici sur la prédiction du type de la maladie (cellule B ou lymphocyte T). On obtient les données d'expression et le type de maladie comme suit :

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

Question 12 : Tester la capacité d'une SVM à prédire la classe de la maladie à partir de l'expression des gènes. Vérifier l'influence des paramètres.

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 :

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

Heureusement, **kernlab** implémente automatiquement SVM multi-classes par une stratégie all-versus-all pour combiner plusieurs SVM binaire.

Question 13 : Tester la capacité d'une SVM à prédire la classe et le stade de la maladie de l'expression des gènes.

Bonne Chance