1 Objectif

Mise en œuvre des SVM sous R et Python. Etude des points supports et des frontières induites. Réflexions sur le paramétrage.

Ce tutoriel vient compléter le support de cours consacré au « Support Vector Machine » auquel nous nous référerons constamment [SVM][1]. Il met en lumière deux éléments importants de la méthode : la position des points supports et le tracé des frontières dans l'espace de représentation lorsque nous construisons un séparateur linéaire ; la complexité du paramétrage dans le recherche de la solution adéquate pour un problème artificiel dont nous maîtrisons pourtant les caractéristiques. Nous utiliserons tour à tour les logiciels R (package ''e1071'') et Python (package ''scikit-learn'').

Nous nous concentrons sur les aspects didactiques. Pour le lecteur intéressé par les aspects opérationnels de la pratique des SVM (schéma apprentissage-test pour l'évaluation des classifieurs, identification des paramètres optimaux à l'aide des grilles de recherche), je conseille la lecture de notre support de référence [SVM, pages 40 à 44, sous R et Python]. Je préconiserais également la lecture des tutoriels consacrés à la comparaison des logiciels proposant les SVM[2] et à l'étude du comportement des différents classifieurs linéaires[3].

2 Points supports et frontières linéaires

2.1 Rappel sur les SVM

Rappelons que dans un problème de classement binaire où x est un vecteur de p descripteurs, y est la variable cible binaire avec $y \in \{+1, -1\}$, l'approche SVM consiste à construire une fonction de classement linéaire, du moins dans une première approche

$$f(x) = x^T\beta + \beta_0$$

La règle de décision s'écrit : *Si f(x) ≥ 0 alors y = +1 sinon y = -1*

La méthode s'appuie sur le principe de la maximisation de la marge. Deux écritures sont possibles, la formulation primale où l'on cherche à optimiser (dans le cas du « hard margin » où une séparation parfaite existe) [SVM, page 8] :

[1] [SVM] Tutoriel Tanagra, « Support Vector Machine - Diapos », mai 2016.
[2] Tutoriel Tanagra, « SVM - Comparaison de logiciels », octobre 2008.
[3] Tutoriel Tanagra, « Classifieurs linéaires », mai 2013.

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$
$$s.c.\ y_i \times f(x_i) \geq 1, \forall i = 1,\ldots,n$$

$\beta = (\beta_1,\ldots,\beta_p)$ est un vecteur de dimension p. L'objectif de l'apprentissage est d'estimer les valeurs de β et β_0 à partir d'un échantillon d'apprentissage de taille n.

La formulation duale met en exergue la notion de points supports [SVM, page 13] :

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle$$
$$s.c.$$
$$\alpha_i \geq 0,\ \forall i$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

α_i ($i = 1, \ldots, n$) est un poids associé à chaque observation. Les points supports (vecteurs de support, exemples critiques) correspondent aux individus pour lesquels $\alpha_i > 0$. L'apprentissage a pour objectif d'estimer les valeurs des α_i dans cette formulation.

Il y a une relation directe entre les vecteurs β et α [SVM, page 15].

2.2 Objectifs et données

L'idée est de reproduire les calculs présentés dans [SVM, pages 6 à 16] où nous avions calculé les coefficients de la frontière linéaire et des droites « marges » séparant parfaitement les classes dans un espace de représentation à deux dimensions. J'avais utilisé Excel pour expliciter la démarche. Voyons si nous retrouvons les mêmes résultats en utilisant des outils qui font autorité dans le domaine du data mining[4].

Nous traitions un échantillon avec n = 10 observations, p = 2 descripteurs ***x = (x₁, x₂)*** et une variable cible **y**. Nous l'avons réorganisé comme suit dans le fichier « **data_svm.txt** ».

i	x1	x2	y
6	5	1	p
7	7	1	p
8	9	4	p
9	12	7	p

[4] Une démarche un peu analogue est disponible sur le web. Par rapport à nous, les classes sont très légèrement bruitées : https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/ch9.html

```
10 13  6      p
1  1   3      n
2  2   1      n
3  4   5      n
4  6   9      n
5  8   7      n
```

La colonne i est un identifiant qui n'intervient pas dans les calculs. Il servira à repérer les individus dans les représentations graphiques. La cible a été recodée en y ∈ {p, n} pour que nos outils la reconnaissent comme un type *factor()* (variable catégorielle) à l'importation. Par commodité, j'ai placé les individus « positifs » en première position dans le fichier, tout en respectant leurs numéros.

Nous utilisons R dans un premier temps. Nous reproduirons les calculs à l'identique sous Python. Nous serons ainsi à même de comparer l'utilisabilité des logiciels et packages dédiés.

2.3 Analyse sous R

2.3.1 Chargement des données et représentation graphique

Nous chargeons le fichier « **data_svm.txt** » à l'aide de la fonction **read.table()**. La première ligne correspond aux noms des variables (header). Les identifiants des individus sont situés sur la première colonne (row.names).

```
#chargement des données
df <- read.table("data_svm.txt",header=T,sep="\t",row.names=1)

#affichage des statistiques descriptives
print(summary(df))
```

Les variables et leurs types sont correctement reconnus. La colonne « i » n'est pas utilisée.

```
       x1              x2       y
 Min.   : 1.00   Min.   :1.00   n:5
 1st Qu.: 4.25   1st Qu.:1.50   p:5
 Median : 6.50   Median :4.50
 Mean   : 6.70   Mean   :4.40
 3rd Qu.: 8.75   3rd Qu.:6.75
```

Nous projetons les points dans le plan en les coloriant selon leur classe d'appartenance et en les numérotant (**Erreur ! Source du renvoi introuvable.**).

```
#représentation graphique des observations dans l'espace (x1, x2)
plot(df$x1,df$x2,type="n")
text(df$x1,df$x2,rownames(df),col=c("blue","red")[df$y],cex=0.75)
```

Tracer une droite permettant de séparer sans erreur les points bleus des rouges est possible. On parle d'hyperplan séparateur dans un espace à plus de 2 dimensions.

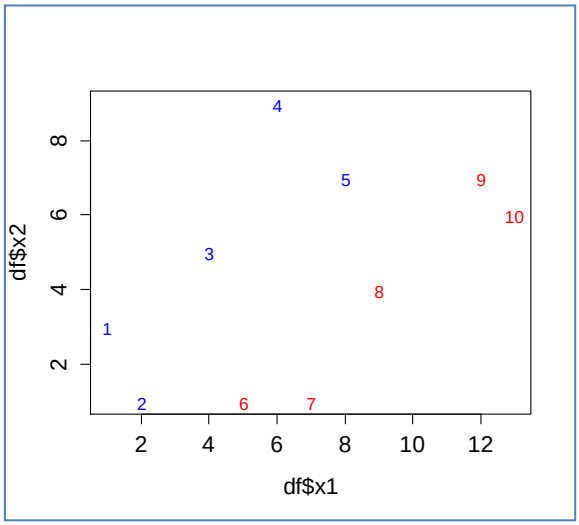

Figure 1 - Représentation des points dans l'espace (x1, x2) - R

2.3.2 Modélisation et lecture des résultats

Nous utilisons le package "e1071" pour l'implémentation des SVM. Nous demandons à la procédure **svm()** de construire un classifieur linéaire (kernel = 'linear') et de ne pas standardiser (centrer et réduire) les données (scale = F).

```
#charger le package e1071
library(e1071)
#SVM linéaire (kernel), pas de standardisation des variables (scale)
mlin <- svm(y ~ x1+x2, data=df, kernel="linear",scale=F)
print(mlin)
```

L'affichage fournit le nombre de points supports *s*.

```
Call:
svm(formula = y ~ x1 + x2, data = df, kernel = "linear", scale = F)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
Number of Support Vectors:  3
```

Habituellement, le nombre de points supports est un bon indicateur. S'il est trop élevé par rapport à la taille de l'échantillon *n*, nous pouvons légitimement penser que la modélisation n'est pas très efficace. Néanmoins, notre taille d'échantillon n = 10 est trop faible pour que nous puissions réellement tirer des conclusions à partir de la valeur s = 3.

En réalité, l'objet est plus riche que l'affichage ne le laisse apparaître. Nous obtenons la liste des propriétés avec la commande **attributes()** :

```
#propriétés de l'objet
print(attributes(mlin))
```

$index notamment indique la liste des points supports

```
$names
 [1] "call"           "type"           "kernel"         "cost"
 [5] "degree"         "gamma"          "coef0"          "nu"
 [9] "epsilon"        "sparse"         "scaled"         "x.scale"
[13] "y.scale"        "nclasses"       "levels"         "tot.nSV"
[17] "nSV"            "labels"         "SV"             "index"
[21] "rho"            "compprob"       "probA"          "probB"
[25] "sigma"          "coefs"          "na.action"      "fitted"
[29] "decision.values" "terms"

$class
[1] "svm.formula" "svm"
```

Il s'agit des observations…

```
#numéro d'individu des points supports
print((rownames(df))[mlin$index])
```

n° **6**, **2** et **5**.

Nous les mettons en évidence dans le graphique (Figure 2) :

```
#mise en évidence des points supports
points(df$x1[mlin$index],df$x2[mlin$index],cex=3,col=rgb(0,0,0))
```

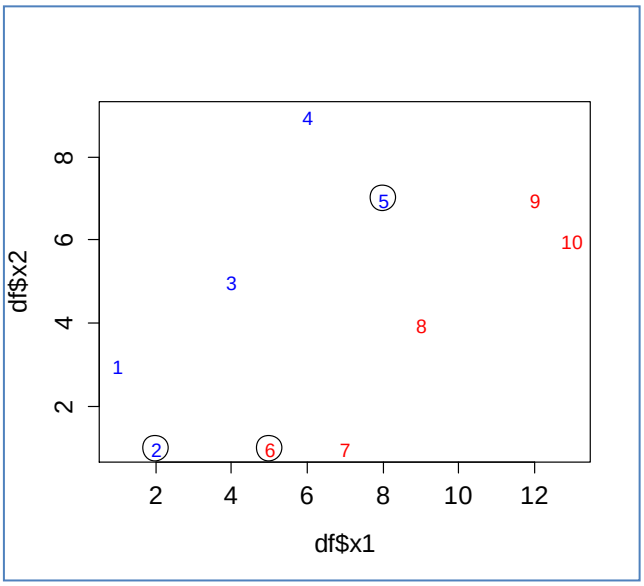

Figure 2 - Mise en évidence des points supports sous R

2.3.3 Représentation des frontières

Hyperplan séparateur. Pour tracer la droite de séparation, nous devons disposer des coefficients de l'équation :

$$\beta_1 x_1 + \beta_2 x_2 + \beta_0 = 0$$

L'objet *mlin* permet d'obtenir directement β_0 avec : β_0 = - *mlin$rho*

```
#constante ➔ -1.666317
beta.0 <- -mlin$rho
print(beta.0)
```

Pour les autres paramètres, il faut passer par la relation [SVM, page 15] :

$\beta_j = \sum_{i=1}^{n} \alpha_i y_i x_{ij}$; sachant que $\alpha_i > 0$ uniquement pour les points supports

La propriété *mlin$coefs* nous renvoie les quantités $\alpha_i y_i$ lorsque ($\alpha_i > 0$); pour obtenir les coefficients β_1 et β_2, il faut multiplier *mlin$coefs* par respectivement les valeurs de x_1 et x_2

```
#coefficients
beta.1 <- sum(mlin$coefs*df$x1[mlin$index])
beta.2 <- sum(mlin$coefs*df$x2[mlin$index])
print(paste(beta.1,beta.2))
```

Nous obtenons β_1 = 0.666492 et β_2 = -0.666492

Il ne reste plus qu'à tracer la frontière de séparation correspondant à : $x_2 = -\frac{\beta_1}{\beta_2} x_1 - \frac{\beta_0}{\beta_2}$

```
#représentation de la droite de séparation
abline(-beta.0/beta.2,-beta.1/beta.2,col="green")
```

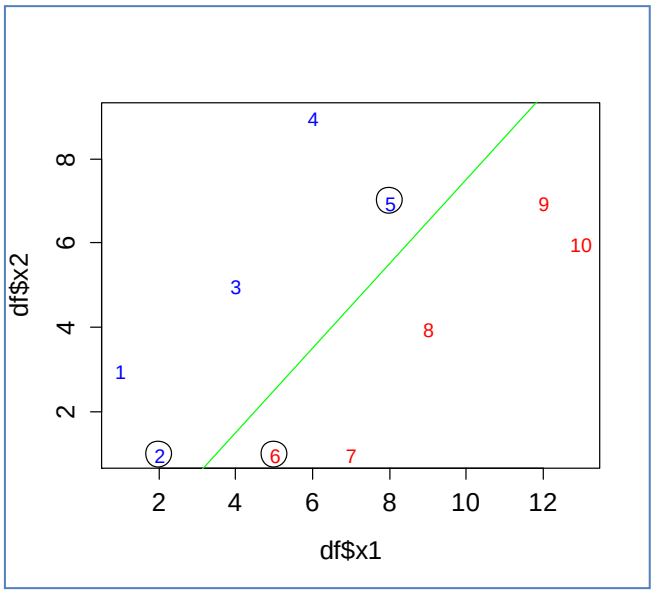

Figure 3- Droite de séparation sous R

Les observations situées au-dessus de la droite (en vert) seront classées négatifs (y = -1), celles situées en dessous se verront attribuées l'étiquette positive (y = +1).

Droites délimitant les « marges ». Il ne reste plus qu'à représenter les droites « marges » sur lesquelles s'appuient les points supports. Elles correspondent à la saturation des contraintes d'optimisation [SVM, page 8] :

$$y_i \times f(x_i) = 1$$

Nous avons deux droites, l'une pour y_i = -1, l'autre pour y_i = +1

```
#the frontières « marginales »
abline((-beta.0-1.0)/beta.2,-beta.1/beta.2,col="gray")
abline((-beta.0+1.0)/beta.2,-beta.1/beta.2,col="gray")
```

Soit,

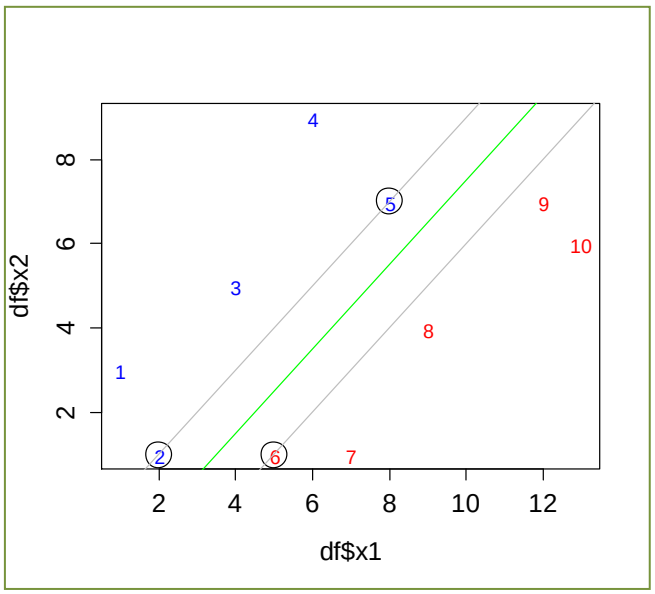

Figure 4 - Représentation graphique finale de l'étude sous R

Remarque : Dans le cadre de la séparation sans erreurs, les droites « marges » passent exactement sur les points supports. Ce n'est plus vrai lorsque la variable cible est bruitée c.-à-d. lorsque certains points peuvent être situés du mauvais côté de la frontière [SVM, soft margin, pages 20 à 23].

2.4 Analyse sous Python

Dans cette section, nous reproduisons à l'identique l'analyse ci-dessus (section 2.3) à l'aide du couple Python / scikit-learn. Certaines de leurs spécificités nous rendent la tâche plus facile (ex. accès direct aux coefficients de l'hyperplan séparateur pour le noyau linéaire),

d'autres en revanche m'ont plongé dans un abime de perplexité tant j'ai du batailler pour obtenir ce que je voulais (ex. les graphiques, que je ne maîtrise pas assez peut être…). Le forum « Stack Overflow » m'a énormément aidé.

2.4.1 Une fonction ad hoc pour les graphiques

Les graphiques apparaissant plusieurs fois tout au long de l'étude. J'ai écrit une procédure spécifique pour l'enchaînement des commandes nécessaires à la représentation du nuage de points avec leurs numéros et les couleurs associées aux classes.

```
#fonction pour graphique nuage de points
#entrées :   data.frame avec l'ensemble des individus
#            data.frame relatifs aux individus positifs et négatifs
def myscatter(df,dfpos,dfneg):
    #nuage de points « blanc » pour définir les dimensions du graphique
    plt.scatter(df.iloc[:,0],df.iloc[:,1],color="white")

    #annotate - positive instances
    for i in dfpos.index:
        plt.annotate(i,xy=(df.loc[i,'x1'],df.loc[i,'x2']),xytext=(-3,-3),textcoords='offset points',color='red')

    #annotate - negative instances
    for i in dfneg.index:
        plt.annotate(i,xy=(df.loc[i,'x1'],df.loc[i,'x2']),xytext=(-3,-3),textcoords='offset points',color='blue')

    return None
#fin de la fonction
```

plt est un alias vers le module « matplotlib.pyplot ».

Je passe par annotate() pour indiquer explicitement des étiquettes (les numéros d'individus) dans le graphique. Il a fallu décaler légèrement [xytext = (-3,-3)] la position pour qu'elles (les étiquettes) soient positionnées exactement à l'emplacement précis des points. Sinon, les droites « marginales » apparaîtraient comme décalées par rapport aux points supports.

Nous sommes parés maintenant pour lancer l'étude avec une trame identique à celle sous R.

2.4.2 Chargement des données et représentation graphique

Première étape toujours, chargement des données et construction du nuage de points.

```
#package pandas pour la manipulation des données
import pandas

#chargement du fichier texte dans une structure data.frame de Pandas
```

```
df = pandas.read_table("data_svm.txt",sep="\t",header=0,index_col=0)
print(df.shape)

#création de deux sous-data.frame correspondants aux individus y=+1 (p) et y=-1 (n)
dfpos = df[df['y']=='p']
dfneg = df[df['y']=='n']

#importation de la librairie pour les graphiques
import matplotlib.pyplot as plt

#représentation graphique
myscatter(df,dfpos,dfneg)
plt.show()
```

Nous obtenons…

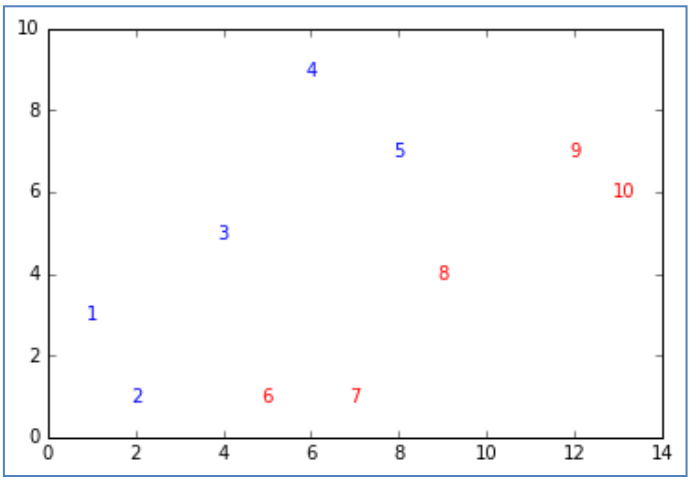

Figure 5 - Représentation des individus dans le plan sous Python

… exactement le même graphique que sous R. La démarche est un peu plus complexe quand même. Passer par une boucle sur **annotate()** ne me satisfait pas vraiment.

2.4.3 Modélisation et lecture des résultats

Nous exploitons la classe SVC de scikit-learn pour réaliser la modélisation. Elle fournit plusieurs informations :

```
#importation de la classe SVC
from sklearn.svm import SVC

#instanciation de l'objet : noyau linéaire
svm = SVC(kernel='linear')

#apprentissage
svm.fit(df.as_matrix()[:,0:2],df.as_matrix()[:,2])
```

```
#nombre de points supports…
print(svm.support_.shape)

#... et leurs numéros
print(df.index[svm.support_])
```

Nous avons les mêmes informations que sous R

```
#nombre de points supports
(3,)
#numéro des points supports → n°2, 5 et 6
Int64Index([2, 5, 6], dtype='int64', name='i')
```

Les poids α_i des points supports sont également disponibles

```
#weigth of the support vectors
print(svm.dual_coef_)
```

A savoir [SVM, page 14],

```
#αi pour i = 2, 5 et 6
[[-0.33325096 -0.11109464  0.44434559]]
```

Il nous reste à les représenter dans notre graphique en utilisant leurs coordonnées.

```
#coordonnées des points supports c1 pour x1, c2 pour x2
c1 = svm.support_vectors_[:,0]
c2 = svm.support_vectors_[:,1]

#mise en évidence des points supports dans la représentation des points
myscatter(df,dfpos,dfneg)
plt.scatter(c1,c2,s=200,facecolors='none',edgecolors='black')
plt.show()
```

Elle correspond à la Figure 2 sous R :

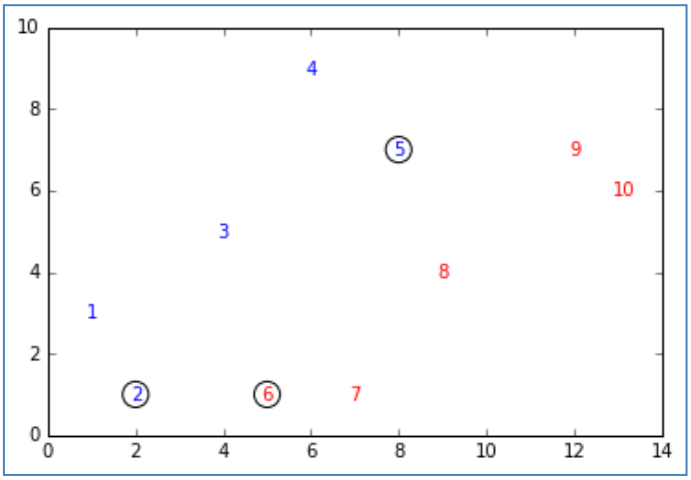

Figure 6 - Mise en évidence des points supports sous Python

2.4.4 Représentation des frontières

SVC intègre une spécificité intéressante, l'objet fournit automatiquement les coefficients de la droite de séparation lorsque le noyau est linéaire.

```
#coefficients β1 et β2
print(svm.coef_)

#constante β0
print(svm.intercept_)
```

En revanche, je n'ai pas trouvé l'équivalent de la fonction abline() de R. L'astuce passe par le calcul des coordonnées de deux points dans le plan, puis de tracer une droite les reliant.

```
#coordonnées des points pour tracer les droites frontières et « marges »
import numpy as np
xf = np.array([3,12])
yf = -svm.coef_[0][0]/svm.coef_[0][1]*xf-svm.intercept_/svm.coef_[0][1]
xb = np.array([4.5,12])
yb = -svm.coef_[0][0]/svm.coef_[0][1]*xb-(svm.intercept_-1.0)/svm.coef_[0][1]
xh = np.array([2,11])
yh = -svm.coef_[0][0]/svm.coef_[0][1]*xh-(svm.intercept_+1.0)/svm.coef_[0][1]

#représentation graphique
myscatter(df,dfpos,dfneg)
plt.scatter(c1,c2,s=200,facecolors='none',edgecolors='black')
plt.plot(xf,yf,c='green')
plt.plot(xb,yb,c='gray')
plt.plot(xh,yh,c='gray')
plt.show()
```

Le graphique est complètement identique à celui de R (Figure 4) (*j'adore l'informatique !*):

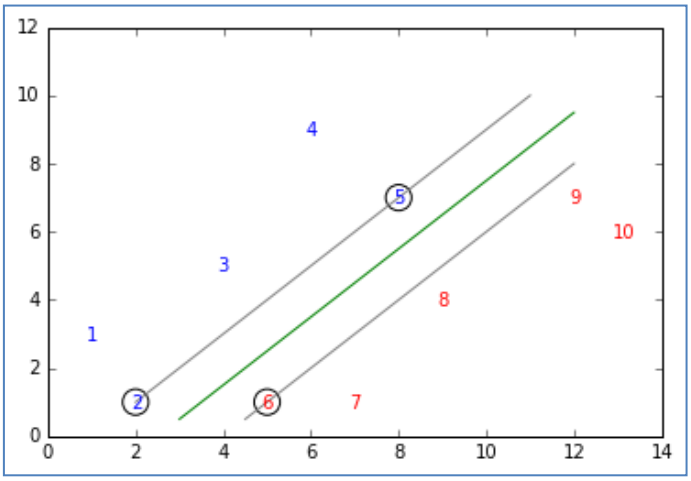

Figure 7- Représentation graphique finale de l'étude sous Python

3 Importance du paramétrage des SVM

3.1 SVM non linéaire - Introduction des fonctions noyau

La maximisation de la marge est un prisme possible pour traiter un problème de classement, au même titre que la maximisation de la vraisemblance ou les moindres carrés. L'énorme intérêt des SVM est la possibilité d'utiliser les fonctions « noyau » $K(x_i,x_{i'})$ dans la formulation duale, nous permettant de nous projeter dans un espace à plus grande dimension sans avoir à créer explicitement les variables intermédiaires. De fait, nous pouvons moduler à notre guise les capacités de représentation de notre classifieur.

Dans le cadre « soft margin », le problème d'optimisation s'écrit [SVM, page 28]

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} K(x_i, x_{i'})$$

$$s.c.$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \ \forall i$$

Le paramètre de coût (''cost parameter'') **C** est crucial. Il indique notre tolérance aux individus mal classés. Trop élevé associé un noyau inadéquat, il peut nous mener au sur-apprentissage car notre modèle colle trop aux spécificités de l'échantillon d'apprentissage, difficilement généralisable dans la population. Trop faible, nous n'exploitons pas suffisamment les données, nous menant au sous-apprentissage.

Fixer la bonne valeur de **C** n'est pas évident. Dans notre étude, travaillant sur des données synthétiques générées artificiellement, il nous sera facile de suivre les bonnes pistes. En situation réelle, sur des données et problèmes dont nous ne maîtrisons pas tous les tenants et aboutissants, nous sommes obligés de tâtonner.

Malgré cela, je note que si la question du noyau revient souvent dans les publications scientifiques, la problématique de **C** est peu abordée dans les expérimentations, lesquelles pourtant se basent exclusivement sur la recherche des meilleures performances.

Nous menons exclusivement notre étude sous R dans cette partie. Le lecteur curieux devrait pouvoir facilement transposer les opérations sous Python.

3.2 Objectif et données

Nous travaillons toujours dans un espace à 2 dimensions pour faciliter la compréhension de la disposition des points. Nous avons généré aléatoirement des données dans le carré [0, 1], dans lequel nous avons inclus un triangle isocèle. Les individus situés à l'intérieur du triangle appartiennent à la première classe, ceux situés à l'extérieur à la seconde.

Ci-dessous le programme R associé. Nous générons n = 30 individus pour l'apprentissage, et n_{test} = 5000 pour l'évaluation des performances (échantillon test).

```
#pour disposer toujours de la même séquence de données
set.seed(10)

#fonction générant le concept à modéliser
generate.data <- function(n){
  x2 <- runif(n)
  x1 <- runif(n)
  y <- factor(ifelse((x2>2*x1)|(x2>(2-2*x1)),1,2))
  return(data.frame(x1,x2,y))
}

#échantillon d'apprentissage - n = 30
dtrain <- generate.data(30)
#représentation graphique
plot(dtrain$x1,dtrain$x2,col=c("blue","red")[dtrain$y],xlim=c(0,1),ylim=c(0,1))
abline(0,2)
abline(2,-2)

#échantillon test - n_test = 5000
dtest <- generate.data(5000)
#représentation graphique
plot(dtest$x1,dtest$x2,col=c("blue","red")[dtest$y],xlim=c(0,1),ylim=c(0,1))
abline(0,2)
abline(2,-2)
```

Deux remarques essentiellement (Figure 8) :

1. La discrimination parfaite est possible. Encore faut-il disposer d'une fonction de classement capable de représenter un triangle. Cela va agir sur le biais (de l'erreur).
2. L'échantillon d'apprentissage ne « remplit » pas assez l'espace de représentation, notamment dans la partie basse du triangle (points rouges). Cela va engendrer de l'incertitude et agir sur la variance (de l'erreur).

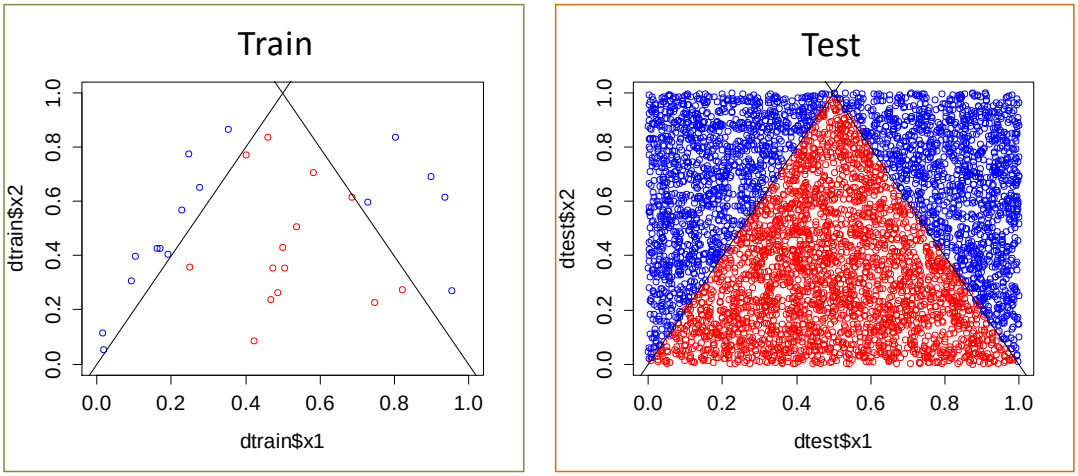

Figure 8 - Concept à apprendre - Echantillons d'apprentissage (TRAIN) et de test (TEST)

3.3 SVM avec noyau linéaire

Manifestement, une droite ne permet pas de séparer les points bleus des rouges. Mais, sans représentation graphique, nous ne le saurions pas. Faisons comme si de rien n'était, et lançons un SVM linéaire.

```
#e1071
library(e1071)
#SVM linéaire
mlin <- svm(y ~ x1+x2, data=dtrain, kernel="linear",scale=F)
print(mlin)
#représentation des points supports dans le plan
plot(dtrain$x1,dtrain$x2,col=c("blue","red")[dtrain$y],xlim=c(0,1),ylim=c(0,1))
points(dtrain$x1[mlin$index],dtrain$x2[mlin$index],pch=5,cex=1.75,col=rgb(0,0,0))
abline(0,2)
abline(2,-2)
```

s = 28 points supports sont annoncés…

```
Call:
svm(formula = y ~ x1 + x2, data = dtrain, kernel = "linear", scale = F)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
Number of Support Vectors:  28
```

… soit quasiment tous les points de l'échantillon d'apprentissage. Ce n'est pas très bon signe généralement. Dans le plan (Figure 9), nous constatons qu'il y a un problème à l'évidence. Les points supports sont disséminés partout, leurs positions n'ont absolument rien à voir avec un triangle.

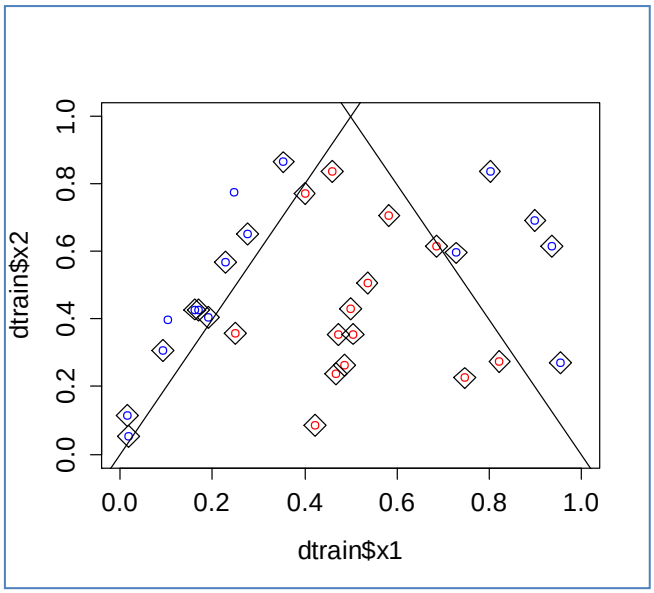

Figure 9 - Points supports (entourés d'un losange) - Noyau linéaire

Traçons maintenant la frontière,

```
#constante β0
beta.0 <- -mlin$rho
#coefficients β1 et β2
beta.1 <- sum(mlin$coefs*dtrain$x1[mlin$index])
beta.2 <- sum(mlin$coefs*dtrain$x2[mlin$index])
#tracé de la droite de séparation
plot(dtrain$x1,dtrain$x2,col=c("blue","red")[dtrain$y],xlim=c(0,1),ylim=c(0,1))
abline(-beta.0/beta.2,-beta.1/beta.2,col="green")
```

Elle est complètement à l'ouest (*à l'est plutôt si on veut être rigoureux*),

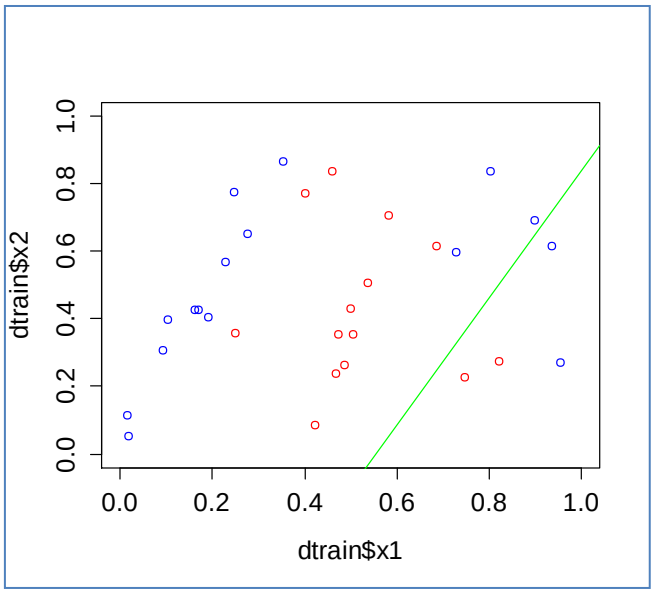

Figure 10 - Frontière linéaire - Noyau linéaire

28 individus sont classés « bleus » ; 4 sont classés « rouges », dont **2** à bon escient, **2** à tort.

Nous pouvons confirmer ce diagnostic graphique en calculant la matrice de confusion sur l'échantillon d'apprentissage :

```
#prédiction sur l'échantillon d'apprentissage, matrice de confusion
ylin.train <- predict(mlin,dtrain)
table(dtrain$y,ylin.train)
```

Le résultat est conforme à ce que nous observons dans la Figure 10.

```
   ylin.train
     1  2
  1 14  2
  2 12  2
```

Il n'est pas nécessaire de mesurer les performances sur l'échantillon test. Le résultat sera catastrophique. Un classifieur linéaire n'est visiblement pas approprié dans notre situation.

3.4 SVM avec noyau polynomial

3.4.1 Choix du noyau polynomial

A bien regarder la disposition de nos points (Figure 8), on peut se dire qu'il est possible d'approximer la frontière sous forme de triangle avec une parabole à concavité tournée vers le bas. L'équation s'écrirait :

$$x_2 = ax_1^2 + bx_1 + c$$

Un noyau polynomial devrait convenir [SVM, page 29]. Nous fixons le paramètre **''coef0 = 1''** pour que les termes (x_1, x_1^2,x_2) soient pris en compte dans la modélisation [SVM, page 27].

Voyons ce que cela donne en apprentissage.

```
#utilisation d'un noyau polynomial de degré 2
mpoly <- svm(y ~ x1+x2, data=dtrain, kernel="polynomial", scale=F, coef0=1, degree=2)
print(mpoly)
```

Par défaut, nous constatons que **svm()** utilise la valeur **''C = 1''** pour le paramètre de coût lors de l'induction. La sortie R nous annonce 29 points supports. Ce n'est pas très bon signe.

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  2
      gamma:  0.5
     coef.0:  1
```

```
Number of Support Vectors:  29
```

Passons outre et appliquons le modèle sur l'échantillon test pour évaluer ses performances.

```
#prédiction sur l'échantillon test
ypoly.test <- predict(mpoly,dtest)
#matrice de confusion
mc.poly <- table(dtest$y,ypoly.test)
print(mc.poly)
#taux d'erreur en test
err.poly <- 1-sum(diag(mc.poly))/sum(mc.poly)
print(err.poly)
```

R nous annonce un taux d'erreur catastrophique de 49.4%.

```
#matrice de confusion
   ypoly.test
       1    2
  1 1987  530
  2 1940  543
```

En essayant de visualiser la frontière induite sur l'échantillon test,…

```
#plotting
plot(dtest$x1,dtest$x2,col=c("blue","red")[ypoly.test],xlim=c(0,1),ylim=c(0,1))
abline(0,2,lwd=2)
abline(2,-2,lwd=2)
```

…nous constatons que le classifieur dessine une droite pour discriminer les classes.

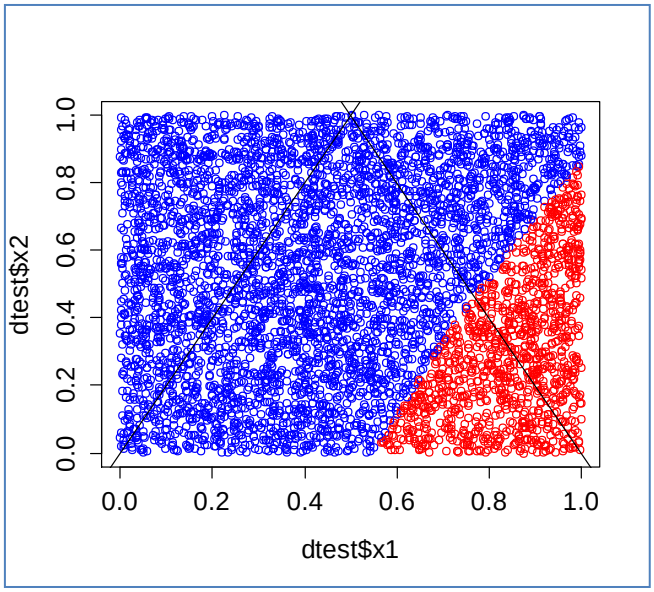

Figure 11 - Frontière induite par un SVM à noyau polynomial de degré 2 (C = 1)

Pourtant, l'idée d'approximer le triangle par une parabole ne semblait pas farfelue. Pourquoi la méthode n'en tient pas compte et dessine plutôt une droite ?

3.4.2 Modification du paramètre de coût

Si la parabole semble adéquate pour approximer le concept sous-jacent reliant la cible aux descripteurs, les insuffisances proviennent forcément d'un mauvais paramétrage de l'algorithme de modélisation. Dans le cas des SVM, le paramètre de coût, qui correspond *grosso modo* à une tolérance aux erreurs, joue un rôle fondamental. Sur nos données artificielles, nous ***savons*** qu'une discrimination parfaite est possible, l'espace n'est pas bruité, et la dimensionnalité est faible (*p = 2* descripteurs seulement pour *n = 30* observations). Il ne paraît pas inconsidéré d'augmenter la valeur de C que nous fixons à ''C = 1000''. Voyons ce que donne l'apprentissage.

```
#nouveau paramétrage avec C = 1000
mpoly <- svm(y ~ x1+x2, data=dtrain, kernel="polynomial", scale=F, cost=1000, coef0=1, degree=2)
print(mpoly)
```

svm() nous annonce 10 points supports…

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1000
     degree:  2
      gamma:  0.5
     coef.0:  1
Number of Support Vectors:  10
```

… qui se positionnent opportunément le long du triangle, surtout dans sa partie haute.

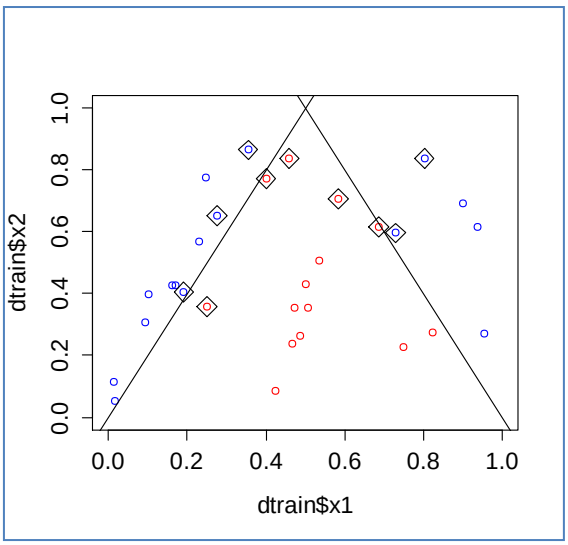

Figure 12 - Position des points supports - Noyau polynomial de degré 2 - C = 1000

On peut s'attendre à de meilleures performances et, effectivement, nous avons un taux d'erreur de 10.76% sur l'échantillon test.

Lorsque nous traçons la frontière induite sur nos données tests,…

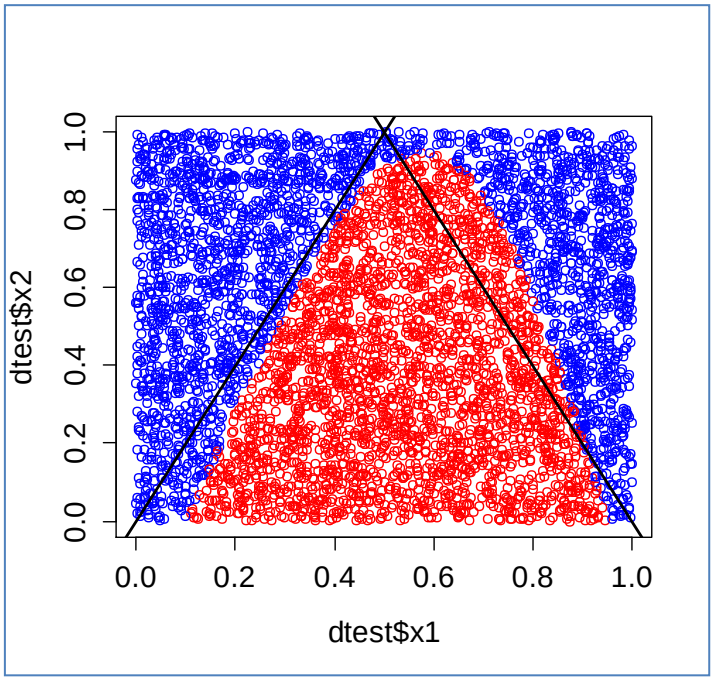

Figure 13 - Frontière induite par un noyau polynomial de degré 2 (C = 1000)

…nous constatons que l'approximation du triangle par une parabole était une idée viable. Une partie de l'imprécision tient à la faible taille de l'échantillon d'apprentissage *n* qui se traduit notamment par « l'absence » de points supports rouges dans la partie basse du triangle (Figure 12).

Remarque 1 : Le lecteur pourra le vérifier aisément en modifiant le programme R qui accompagne ce document : lorsque nous augmentons la taille de l'échantillon d'apprentissage, les points supports de couleurs opposées s'aligneront mieux le long des bords du triangle, la parabole est mieux positionnée et, par conséquent, le taux d'erreur s'améliore naturellement.

Remarque 2 : Si nous avions laissé coef0 à sa valeur par défaut (coef0 = 0), ni x_1 ni x_2 n'apparaissent dans la transformation sous-jacente à la fonction noyau [SVM, page 27]. Il s'avère impossible d'obtenir une frontière prenant la forme d'une parabole, la modélisation est mauvaise quelle que soit la valeur du paramètre de coût. C'est la forme de la fonction utilisée pour retracer la relation entre les *x* et *y* qui n'est pas adaptée dans ce cas. Nous sommes dans une situation analogue à celle observée pour l'utilisation d'un noyau linéaire.

Remarque 3 : Non rapportée ici mais disponible dans le programme R, le noyau RBF (radial basis function) gagne en efficacité également lorsque nous augmentons résolument la valeur du paramètre de coût C. Nous atteignons un taux d'erreur en test de 6.78%.

4 Conclusion

Ce tutoriel est essentiellement à visée pédagogique. Dans la première partie, j'ai essayé d'illustrer concrètement les innombrables formules assez ardues que l'on retrouve dans les très nombreux supports de cours qui présentent l'approche SVM. Je le dis souvent à mes étudiants, disséquer une expression mathématique n'a d'intérêt, dans nos domaines tout du moins, que si elle nous permet de mieux saisir les idées sous-jacentes à la méthode. Je m'étais « amusé » à détailler les calculs sous Excel dans le support de cours préalable à ce document. L'idée était de reproduire la démarche en utilisant R et Python avec les bibliothèques dédiées au data mining.

Dans la seconde partie, nous nous sommes intéressés au paramétrage. L'affaire n'est pas simple. L'exemple permet de toucher du doigt les principales difficultés. Outre le choix du noyau, le paramètre de coût C, souvent passé sous silence dans les résultats d'expérimentations publiées, joue un rôle absolument fondamental. Personne ne fixe de valeur C = 1000 dans des problèmes réels (dans les publications que j'ai lues en tous les cas), on s'est rendu compte pourtant que c'est une valeur envisageable dans notre configuration très spécifique (absence de bruit, faible dimensionnalité). Cela veut dire surtout qu'il faut lui accorder une attention toute particulière quand nous menons une analyse.

5 Références

Abe S., « Support Vector Machines for Pattern Classification », Springer, 2010.

Cristianini N., Shawe-Taylor J., « Support Vector Machines and other kernle-based learning algorithms », Cambridge University Press, 2000.