Nom et prénoms du binôme : CHEAM Richard, NOUV Ratanakmuny

ENSIIE
Date : 3-Mai-2024

TP Séparateurs à Vaste Marge – « SVM»
- J. Boudy - TSP –

Version sous Python

- 2 Mai 2024 -

Introduction au TP

Les réponses, les équations et schémas correspondant à chaque question doivent être placés directement dans ce compte-rendu électronique (document Word) en précisant bien les noms du binôme à l'emplacement prévu ci-dessus.

Le TP lui-même se déroule en deux parties :

- une première partie préparatoire consacrée à la théorie des SVM avec quelques questions théoriques et petites démonstrations ; les équations peuvent être écrites à la main sur une feuille blanche puis scannées pour inclusion dans ce compte-rendu électronique
- puis une seconde partie sur la mise en œuvre d'une technique classique de l'algorithme des SVM sur un cas pratique (application de Télévigilance) en utilisant l'exemple de programme sous Python donné en Annexe.

Instructions : to implement environments based on *Python* (version 3.10 or previous ones) et *Anaconda* (*Spider*) if you do not have *Matlab* environment:
https://www.python.org/downloads/

https://www.anaconda.com/products/individual#download-section

Partie I : Les Séparateurs à Vaste Marge et Méthodes de Fusion – questions théoriques (partie TD)

N.B. : pour cette partie théorique, il est possible de s'aider des éléments du cours et/ou des articles proposés.
Par exemple celui de Ch. Burges : https://www.di.ens.fr/~mallat/papiers/svmtutorial.pdf
Et celui de S. Graja pour les aspects plus applicatifs (signaux ECG) avec des éléments théoriques (fourni en version PDF avec le sujet de TP)

1) Quel est le titre correspondant en anglais de l'approche de classification « Séparateurs à Vaste Marge » ? Pourquoi ? Rappeler en quelques lignes le principe de cette approche ?

   The corresponding English title for the classification approach "Séparateurs à Vaste Marge" is "Support Vector Machines" (SVM). The principle of SVMs is to find the hyperplane that best separates the data classes in a feature space.

This hyperplane is chosen to maximize the margin, that is, the minimum distance between the hyperplane and the nearest data points from each class. These closest points are called support vectors.
In cases where the data are not linearly separable in the original space, SVMs use kernel functions to project the data into a higher dimensional space where they can be linearly separated, thus allowing for the handling of nonlinear problems while maintaining manageable computational complexity.

2) Qu'est-ce que le passage du Problème d'Optimisation Primal au Problème Dual ? en s'aidant du cours (ou d'articles précisés en TP) expliciter par quelques équations (écrites à la main et scannées pour inclusion dans ce compte-rendu électronique).

According to the course:
The transition from the primal to the dual problem in Support Vector Machines (SVM) is advantageous because the dual formulation allows for the incorporation of the kernel trick, enabling SVMs to handle non-linearly separable data efficiently. This shift simplifies computations, as it deals with Lagrange multipliers rather than high-dimensional weight vectors, and leads to a sparse solution where only support vectors define the decision boundary. Moreover, the dual problem optimizes directly under constraint conditions, ensuring that the solution maximizes the margin between data classes.

The primal problem involves finding a hyperplane that separates classes with the largest possible margin while minimizing the norm of the weight vector ($\boldsymbol{w}$). It can be expressed for linearly separable data as:

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{subject to: } y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1, \forall\, i$$

The use of Lagrange multipliers (α_i) allows for the reformulation of the primal problem into a dual problem. The dual problem focuses on maximizing the margin based on the Lagrange multipliers, while respecting certain constraint conditions. The dual problem is formulated as:

$$\max_{\alpha}\left(\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)$$

$$\text{subject to: } \alpha_i \geq 0 \text{ et } \sum_{i=1}^{n}\alpha_i y_i = 0$$

The Karush-Kuhn-Tucker (KKT) conditions ensure a correspondence between the solutions of the primal and dual problems, indicating that the solutions are optimal and equivalent under certain conditions.

3) Rappeler 3 fonctions noyaux (Kernel) type, en donnant leurs relations mathématiques, utilisées avec les SVM. Quelle propriété importante doit vérifier une fonction noyau et pourquoi? Que se passe-t-il respectivement pour un noyau polynomial d'ordre 1 et un noyau de type sigmoïde, à quels algorithmes connus de classification peut-on les rattacher ? Donner la relation de la fonction de décision utilisée par l'algorithme des SVM faisant alors intervenir la fonction noyau dans le cas général ?

1. **Linear Kernel**:

$$(K(x, x') = x \cdot x')$$

This kernel is simply the dot product between two vectors. It is used when the data is linearly separable in the input space.

2. **Polynomial Kernel**:

$$(K(x, x') = (x \cdot x' + C)^d)$$

Where (C) is adjustable parameter and (d) is the degree of the polynomial.
This kernel enables the capture of higher-degree feature interactions.

3. **Gaussian or RBF (Radial Basis Function) Kernel**:

$$(K(x, x') = exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{\sigma}) = \exp(-\mathrm{C}|x - x'|^2))$$

Where (C) is a parameter that determines the extent of the window or influence area of a single training example.
This kernel is particularly useful for handling cases where the class attribute relationship is non-linear.

Important Property of Kernel Functions is a kernel function must satisfy the **Mercer's condition**, meaning it must correspond to a dot product in a Hilbert space. Specifically, the Gram matrix constructed from the kernel (K) must be semi-definite positive. This property is crucial as it ensures that the optimization problem remains convex, thereby allowing for the finding of a globally optimal solution.

Polynomial Kernel of Order 1: This kernel corresponds to a linear classifier. When the degree ($d = 1$), the polynomial kernel simplifies to $(K(x, x') = x \cdot x' + C)$, which is equivalent to linear regression or linear classification SVM depending on the intercept term (C).
Sigmoid Kernel: $(K(x, x') = \tanh(x \cdot x' + r))$. This kernel mimics the activation of a neuron in artificial neural networks, equating to a multi-layer perceptron without a hidden layer when used in SVMs. It is related to neural networks of the perceptron type.

The decision function used by the SVM algorithm, incorporating the kernel function, is given by:

$$f(x) = \left(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b\right)$$

where (α_i) are the Lagrange multipliers of the support vectors (x_i), (y_i) are the class labels of the support vectors, and (b) is the bias. This formulation allows for classifying data in a potentially infinite-dimensional space without having to explicitly perform transformations to that space, thanks to the properties of kernel functions.

Partie II : Simulation de l'algorithme SVM sous Python (partie TP)

4) De quel type sont les données à traiter ? A quoi correspondent les différentes colonnes ? Quels paramètres différentient les données normales de celles anormales (situations de détresses). Pour ce faire consulter les fichiers de données ***Donnees_Televigilance_09-10-08.txt***

N.B. : attention de ne pas modifier le fichier de données (sortir du fichier sans sauvegarde).

Exemple de données extrait du fichier en question :
0 07-06-2006 11:04:18 0 0 0 0 15 042
0 07-06-2006 11:04:48 0 0 0 0 15 041
0 07-06-2006 11:05:00 0 0 0 0 15 061
0 07-06-2006 11:05:30 0 0 0 0 15 082

0 07-06-2006 14:30:00 0 0 1 0 15 028
0 07-06-2006 14:30:30 0 0 1 0 10 039
0 07-06-2006 14:31:00 0 0 1 0 00 039
0 07-06-2006 14:31:30 0 0 1 0 00 036

From the left side:

- The **first column** represents **user index** (bit)
- The **second column** represents **date**
- The **third column** represents record of **timestamp (heure)** (variation in average of 30s, after every 30s new data is given)
- The **fourth column** represents **battery index** (bit), where:
  - 0 means normal
  - 1 means exchanged
- The **fifth column** represents **fall detection (chute détection)** (bit), where:

- 0 means normal
- 1 means fall detected, indicating a potential emergency situation

- The **sixth column** represents **posture** (bit), where:
  - 0 means standing posture (debout)
  - 1 means lying posture (couché)
- The **seveth column** represents **emergency call (bouton d'appel d'urgence)** (bit), where:
  - 0 means normal (no call)
  - 1 means call
- The **eigth column** represents **movement, rest or active (au repos / achuité normale)** (4 bits), where the bits vary from:
  - 0 to 15 (16 values)
- The **last column** represents the measurement of **pulse cardiac (pouls)** (8 bits), where the bits vary from:
  - 0 to 255 bpm (256 values), bpm indicates beats per minute (heart rate)

Hence,

- from the **1st** to the **7th** column, they are equal to 1 octet (8 bits)
- the **8th** column is equal to 1 octet (8 bits) as well.

As a result, each row represents 3 octets (16 bits) of information.

Parameters that differentiate normal data from abnormal data (distress situations) are noticeably the **6th, 8th, 9th** columns.

For example, the data in red suggests the abnormal since:

- The value of ones in the **6th** column indicates a fall detection situation.
- The values in the **8th** column start to drop indicating weak movement to movement at all.
- The values in the **9th** column suggest a low heart beat rate, below 40bpm (Bradycardia).

Which suggests that the user could possibly fell down unconsciously.

5) Implémenter le programme de SVM provenant de la librairie ScikitLearn (sous Python) dans votre environnement (par ex. Anaconda) dont des éléments de programmation sont donnés en Annexe 1 avec trois jeux différents de données simulées de Télévigilance. **Exécuter le programme pour les différents types de noyaux (linéaire, gaussien et polynomial). Que remarquer en commentant les résultats ?**

Response will be highlight in yellow to avoid confusion.

Code :

```
from pylab import *
import numpy as np
from sklearn import svm
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn import metrics

########### Données d'Apprentissage et de Test indépendantes pour le SVM ##########

#------- Jeu de données n°1 --------
#--- Apprentissage à partir de 8 exemples de dimension N=2 Feature X=(Mvt,Pouls)
X1 = [[15, 42], [15, 45], [14, 61], [3, 70], [0, 30], [15, 10], [4, 38], [2, 42]]
y1 = [0, 0, 0, 0, 1, 1, 1, 1] #4 premiers sont normales
```

```
# Données de Test (prédiction)
Lab_reels1 = [0, 1, 1, 1, 0]
#X_test1=[[15., 60.], [2., 42.], [4, 39], [2, 35], [15, 41]]
X_test1=[[15., 60.], [2., 22.], [4, 39], [0, 40], [15, 36]]

#------- Jeu de données n°2 --------
#--- Apprentissage à partir de 16 exemples de dimension N=2 Feature X=(Mvt,Pouls)
X2 = [[15, 42], [15, 41], [14, 61], [3, 70], [13, 40], [14, 43], [11, 65], [3, 70], [0, 58], [15, 37], [4, 38], [2, 42], [2, 59],
[13, 33], [5, 38], [0, 35]]
y2 = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

# Données de Test (prédiction)
Lab_reels2 = [0, 0, 1, 1, 1, 1, 0, 0]
X_test2=[[15., 60.], [2., 42.], [4, 39], [2, 35], [15, 36], [14, 39], [3, 80], [3, 60]]

#------- Jeu de données n°3 --------
#--- Apprentissage à partir de 16 exemples de dimension N=3 Feature X=(Mvt,Pouls, SpO2)
X3 = [[15, 56, 92], [15, 58, 93], [14, 61, 90], [3, 70, 89], [13, 41, 86], [14, 70, 91], [11, 65, 92], [3, 70, 90], [0, 58, 85],
[15, 37, 80], [4, 38, 75], [2, 42, 84], [2, 40, 85], [13, 33, 74], [5, 38, 84], [0, 35, 80]]
y3 = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

# Données de Test (prédiction)
Lab_reels3 = [0, 0, 1, 1, 1, 1, 0, 0]
X_test3=[[15., 60., 90.], [2., 42., 89.], [4., 39., 75.], [2., 35., 80.], [15., 36., 82.], [14., 39., 79.], [3., 80., 91.], [3.,
60.,93.]]

# functions to train model and make prediction for 3 different kernel

def linear_SVM(X, y, X_test, Lab_reels):
    clf = svm.SVC(kernel='linear')
    clf.fit(X, y)
    # get support vectors
    SV=clf.support_vectors_
    print("Support vector:\n",SV)
    # get indices of support vectors
    Index_SV=clf.support_
    print("Index SV:", Index_SV)
    # get number of support vectors for each class
    Number_SV=clf.n_support_
    print("Number of support vectors for each class:", Number_SV)
    # ------ Prediction  (Test) -----
    Lab_pred_lin=clf.predict(X_test)
    delta_Lab = Lab_pred_lin - Lab_reels
    print('Erreurs Kernel Linéaire :', delta_Lab)
    Nb_erreurs=sum(abs(delta_Lab))
    Taux_erreurs=Nb_erreurs/size(Lab_reels)
    Taux_Reco=(1-Taux_erreurs)*100
    print('Taux_Reco Noyau Linéaire en % :',Taux_Reco)
    print("Accuracy of linear kernel :", metrics.accuracy_score(Lab_reels, Lab_pred_lin))
    return

def polynomial_SVM(X, y, X_test, Lab_reels):
    clf = svm.SVC(kernel='poly') #degree = 3 by default
    clf.fit(X, y)
    # get support vectors
    SV=clf.support_vectors_
```

```
    print("Support vector:\n",SV)
    # get indices of support vectors
    Index_SV=clf.support_
    print("Index SV:", Index_SV)
    # get number of support vectors for each class
    Number_SV=clf.n_support_
    print("Number of support vectors for each class:", Number_SV)
    # ------ Prediction  (Test) -----
    Lab_pred_lin=clf.predict(X_test)
    delta_Lab = Lab_pred_lin - Lab_reels
    print('Erreurs Kernel Polynomiale :', delta_Lab)
    Nb_erreurs=sum(abs(delta_Lab))
    Taux_erreurs=Nb_erreurs/size(Lab_reels)
    Taux_Reco=(1-Taux_erreurs)*100
    print('Taux_Reco Noyau Polynomiale en % :',Taux_Reco)
    print("Accuracy of polynomial kernel :", metrics.accuracy_score(Lab_reels, Lab_pred_lin))
    return

def gaussian_SVM(X, y, X_test, Lab_reels):
    clf = svm.SVC(kernel='rbf')
    clf.fit(X, y)
    # get support vectors
    SV=clf.support_vectors_
    print("Support vector:\n",SV)
    # get indices of support vectors
    Index_SV=clf.support_
    print("Index SV:", Index_SV)
    # get number of support vectors for each class
    Number_SV=clf.n_support_
    print("Number of support vectors for each class:", Number_SV)
    # ------ Prediction  (Test) -----
    Lab_pred_lin=clf.predict(X_test)
    delta_Lab = Lab_pred_lin - Lab_reels
    print('Erreurs Kernel RBF :', delta_Lab)
    Nb_erreurs=sum(abs(delta_Lab))
    Taux_erreurs=Nb_erreurs/size(Lab_reels)
    Taux_Reco=(1-Taux_erreurs)*100
    print('Taux_Reco Noyau RBF en % :',Taux_Reco)
    print("Accuracy of RBF kernel :", metrics.accuracy_score(Lab_reels, Lab_pred_lin))
    return

print("################# Jeu de données n°1 #################\n")
# Noyau Linéaire
print("------- Noyau Linéaire -------")
linear_SVM(X1, y1, X_test1, Lab_reels1)
print("\n")
# Noyau Polynomiale
print("------- Noyau Polynomiale -------")
polynomial_SVM(X1, y1, X_test1, Lab_reels1)
# Noyau Gaussian
print("------- Noyau RBF --------")
gaussian_SVM(X1, y1, X_test1, Lab_reels1)
```

Result for Dataset 1:

################ Jeu de données n°1 ################

------- Noyau Linéaire -------
Support vector:
[[15. 42.]
[3. 70.]
[4. 38.]]
Index SV: [0 3 6]
Number of support vectors for each class: [2 1]
Erreurs Kernel Linéaire : [0 0 0 0 0]
Taux_Reco Noyau Linéaire en % : 100.0
Accuracy of linear kernel : 1.0

- **From the result above, for linear kernel, there are 3 support vectors in total out of 8:**
  - **Two of them are from normal class (0): [15, 42] [3, 70]**
  - **One is from abnormal class (1): [4, 38]**

------- Noyau Polynomiale -------
Support vector:
[[15. 42.]
[4. 38.]
[2. 42.]]
Index SV: [0 6 7]
Number of support vectors for each class: [1 2]
Erreurs Kernel Polynomiale : [0 0 0 0 0]
Taux_Reco Noyau Polynomiale en % : 100.0
Accuracy of polynomial kernel : 1.0

- **For 3rd degree polynomial kernel there are 3 support vectors in total out of 8:**
  - **One of them is from normal class (0): [15, 42]**
  - **Two of them are from abnormal class (1): [4, 38], [2, 42]**
- **The same support vector of normal class ([15, 42]) and abnormal class ([4, 38]) were chosen. The difference is the support vector of abnormal class ([2, 42]). However, despite the fact that linear kernel chose 2 out of 3 support vectors from class 0 and polynomial chose 2 out of 3 support vectors from class 1, the predictions are both 100% between polynomial and linear kernel.**

------- Noyau RBF --------
Support vector:
[[15. 42.]
[15. 45.]
[14. 61.]
[3. 70.]
[0. 30.]
[15. 10.]
[4. 38.]
[2. 42.]]
Index SV: [0 1 2 3 4 5 6 7]
Number of support vectors for each class: [4 4]
Erreurs Kernel RBF : [0 0 0 0 1]
Taux_Reco Noyau RBF en % : 80.0
Accuracy of RBF kernel : 0.8

- **Whereas, for rbf kernel, every vector was chosen, 8 in total.**
- **However, the accuracy of prediction drop from 100% (linear or polynomial) to 80% (rbf).**

```
print("################# Jeu de données n°2 #################\n")
print("------- Noyau Linéaire --------")
# Noyau Linéaire
```

```
linear_SVM(X2, y2, X_test2, Lab_reels2)
print("\n")
print("------- Noyau Polynomiale -------")
polynomial_SVM(X2, y2, X_test2, Lab_reels2)
print("------- Noyau RBF --------")
gaussian_SVM(X2, y2, X_test2, Lab_reels2)
```

Result for Dataset 2:

```
################ Jeu de données n°2 ################

------- Noyau Linéaire --------
Support vector:
 [[13. 40.]
 [14. 43.]
 [15. 37.]
 [ 2. 59.]
 [13. 33.]]
Index SV: [ 4  5  9 12 13]
Number of support vectors for each class: [2 3]
Erreurs Kernel Linéaire : [ 0  1  0  0 -1 -1  0  1]
Taux_Reco Noyau Linéaire en % : 50.0
Accuracy of linear kernel : 0.5
```

- **From the result above, there are 5 support vectors in total out of 16:**
  - **Two of them are from normal class (0): [13, 40], [14, 43]**
  - **Three of them are from abnormal class (1): [15, 37], [2, 59], [13, 33]**
- **The prediction is 50% accurate.**

```
------- Noyau Polynomiale -------
Support vector:
 [[15. 41.]
 [13. 40.]
 [14. 43.]
 [ 3. 70.]
 [15. 37.]
 [ 2. 59.]
 [13. 33.]
 [ 5. 38.]]
Index SV: [ 1  4  5  7  9 12 13 14]
Number of support vectors for each class: [4 4]
Erreurs Kernel Polynomiale : [ 0  1  0  0 -1 -1  0  0]
Taux_Reco Noyau Polynomiale en % : 62.5
Accuracy of polynomial kernel : 0.625
```

- **Whereas, for polynomial kernel, 8 support vectors were chosen 4 of them are from normal class (0) and the other 4 are from abnormal class (1). Hence, the decision boundary between the classes is equally influenced by the data points from both classes.**
- **We notice that polynomial kernel performs better than the linear in which it has 62.5.**

```
------- Noyau RBF --------
Support vector:
 [[15. 42.]
 [15. 41.]
 [14. 61.]
 [ 3. 70.]
```

[13. 40.]
[14. 43.]
[3. 70.]
[0. 58.]
[15. 37.]
[4. 38.]
[2. 42.]
[2. 59.]
[13. 33.]
[5. 38.]]
Index SV: [0 1 2 3 4 5 7 8 9 10 11 12 13 14]
Number of support vectors for each class: [7 7]
Erreurs Kernel RBF : [0 1 0 0 0 0 0 0]
Taux_Reco Noyau RBF en % : 87.5
Accuracy of RBF kernel : 0.875

- **For gaussian kernel, 14 out of 16 were chosen as support vectors. The two data points that are not support vectors are [11, 65] and [0, 35].**
- **7 support vectors are from class 0 (normal) and the other 7 are from class 1 (abnormal) which once again indicate the equivalent influence of both classes.**
- **Even better that the other two kernels, RBF has 87.5 accuracy rate which is the best**.

```
print("################# Jeu de données n°3 #################\n")
print("------- Noyau Linéaire -------")
linear_SVM(X3, y3, X_test3, Lab_reels3)
print("\n")
print("------- Noyau Polynomiale --------")
polynomial_SVM(X3, y3, X_test3, Lab_reels3)
print("------- Noyau RBF --------")
gaussian_SVM(X3, y3, X_test3, Lab_reels3)
```

Result for Dataset 3:
################# Jeu de données n°3 #################

------- Noyau Linéaire -------
Support vector:
[[13. 41. 86.]
[0. 58. 85.]
[15. 37. 80.]
[5. 38. 84.]]
Index SV: [4 8 9 14]
Number of support vectors for each class: [1 3]
Erreurs Kernel Linéaire : [0 0 0 0 0 0 0 0]
Taux_Reco Noyau Linéaire en % : 100.0
Accuracy of linear kernel : 1.0

- **With three dimensional dataset, for linear kernel, there are 4 support vectors in total out of 16:**
  - **One of them is from normal class (0): [13, 41, 86]**
  - **Three of them are from abnormal class (1): [0, 58, 85], [15, 37, 80], [5, 38, 84]**
- **So, the decision boundary between the classes is more influenced by the data points from class 1 (abnormal).**
  - **The prediction using linear kernel is 100% accurate.**

------- Noyau Polynomiale --------
Support vector:
[[3. 70. 89.]
[13. 41. 86.]

[0. 58. 85.]
[15. 37. 80.]]
Index SV: [3 4 8 9]
Number of support vectors for each class: [2 2]
Erreurs Kernel Polynomiale : [0 1 0 0 0 0 0 0]
Taux_Reco Noyau Polynomiale en % : 87.5
Accuracy of polynomial kernel : 0.875

- **For polynomial kernel, the same number of support vectors were chosen, 4 out of 16, two are from class 0 and the other two are from class 1.**
- **However, the accuracy is only at 87.5% rather than 100%, which shows that the linear kernel who mostly chose support vectors from class 1 performs better than the equally chosen from polynomial so class 1 is more influent.**

------- Noyau RBF --------
Support vector:
[[15. 56. 92.]
[15. 58. 93.]
[14. 61. 90.]
[3. 70. 89.]
[13. 41. 86.]
[11. 65. 92.]
[3. 70. 90.]
[0. 58. 85.]
[15. 37. 80.]
[4. 38. 75.]
[2. 42. 84.]
[2. 40. 85.]
[5. 38. 84.]]
Index SV: [0 1 2 3 4 6 7 8 9 10 11 12 14]
Number of support vectors for each class: [7 6]
Erreurs Kernel RBF : [0 1 0 0 0 0 0 0]
Taux_Reco Noyau RBF en % : 87.5
Accuracy of RBF kernel : 0.875

- **Here, for RBG kernel, there are 13 support vectors which is a lot more than linear and polynomial kernel.**
- **7 of those are from class 0 and the other 6 are from class 1.**
- **However, the despite the great number of support vectors, it did not outperform linear kernel.**
- **In conclusion, class 1 influences the decision making, since the linear kernel who chose most of support vectors from class 1 outperform the other two kernels so it is not necessary to use other kernels rather than the linear one.**

N.B. : Quelques consignes pour vous aider à démarrer avec les outils de Scikit-learn :
- vous pouvez consulter :
- voir quelques exemples simples de simulation pour démarrer (cf. aussi annexe) : https://scikit-learn.org/stable/modules/svm.html#classification
- des éléments sur les paramètres des noyaux classiques (kernel) : https://scikit-learn.org/stable/modules/svm.html#svm-kernels

Pour la ***question facultative n°6, s***i vous voulez visualiser l'hyperplan séparateur à vaste marge (SVM) aller dans : https://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#sphx-glr-auto-examples-svm-plot-separating-hyperplane-py

Vous êtes également libres de reprogrammer les éléments donnés en annexe sous une forme plus compacte et conviviale.

6) ***(Question Facultative)*** : en s'aidant de la bibliothèque de Scikit-Learns, trouver un moyen de représenter l'hyperplan séparateur à vaste marge avec ses vecteurs supports et ajouter les lignes de code à la fin du programme actuel pour cette représentation graphique finale. Que remarquer entre les 3 types de noyaux ? répartition et nombre de vecteurs supports autour de l'hyperplan séparateur ?

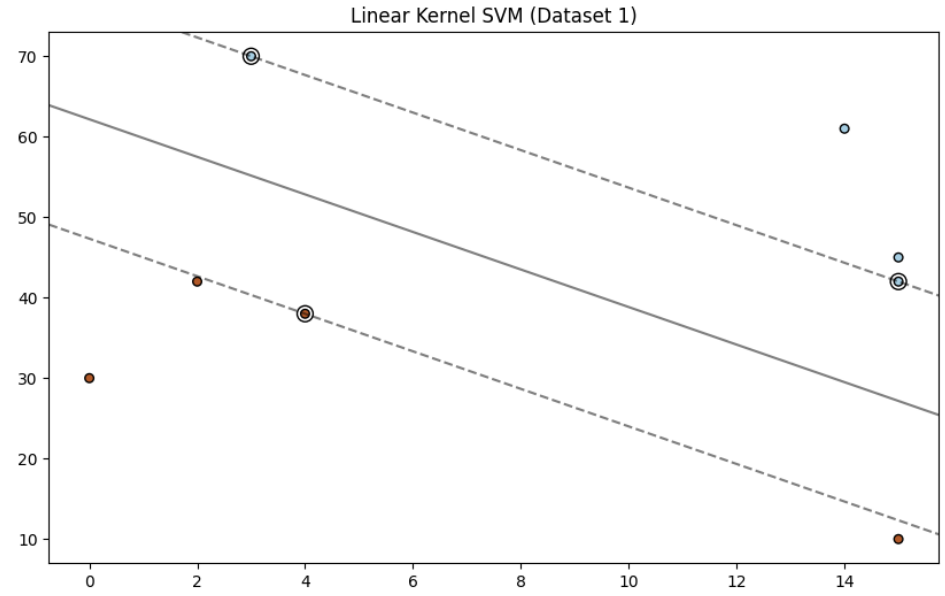
N.B. : Voir par exemple : https://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#sphx-glr-auto-examples-svm-plot-separating-hyperplane-py

Code :

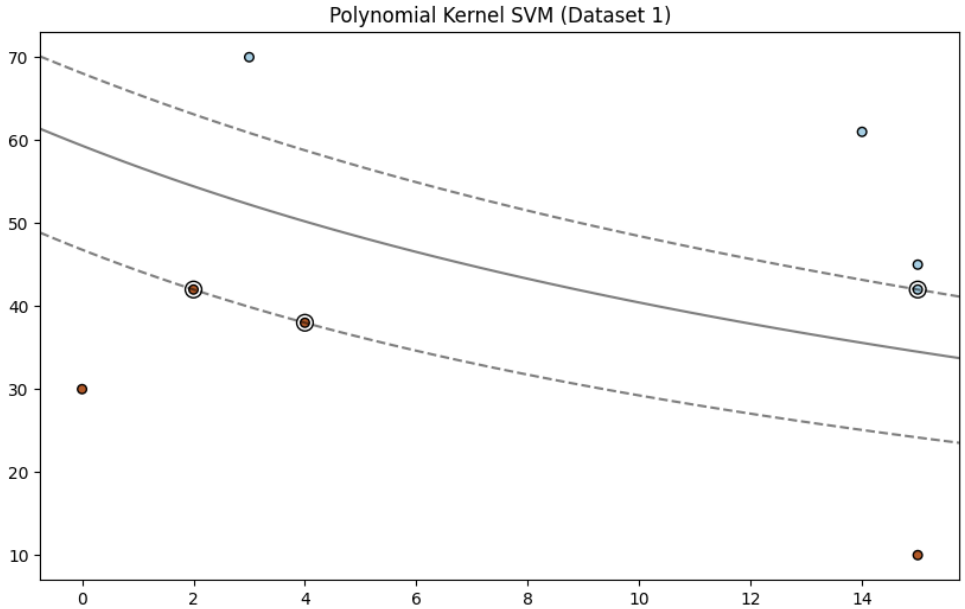
```
def plot_svm_decision_boundary(clf, X, y, title="SVM Decision Boundary"):
    # Set up the plot
    plt.figure(figsize=(10, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired, edgecolors='k')

    # Get axis limits
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # Create grid to evaluate model
    xx = np.linspace(xlim[0], xlim[1], 100)
    yy = np.linspace(ylim[0], ylim[1], 100)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = clf.decision_function(xy).reshape(XX.shape)

    # Plot decision boundary and margins
    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
    # Highlight support vectors with a circle around them
    ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
               linewidth=1, facecolors='none', edgecolors='k')

    plt.title(title)
    plt.show()

X1= np.array(X1)
y1= np.array(y1)

clf_linear = svm.SVC(kernel='linear')
clf_linear.fit(X1, y1)
plot_svm_decision_boundary(clf_linear, X1, y1, title="Linear Kernel SVM (Data 1)")

# Train a SVM classifier with a polynomial kernel
clf_poly = svm.SVC(kernel='poly', degree=3)
clf_poly.fit(X1, y1)
plot_svm_decision_boundary(clf_poly, X1, y1, title="Polynomial Kernel SVM (Data 1)")

# Train a SVM classifier with an RBF kernel
clf_rbf = svm.SVC(kernel='rbf')
clf_rbf.fit(X1, y1)
plot_svm_decision_boundary(clf_rbf, X1, y1, title="Gaussian Kernel SVM (Data 1)")

X2= np.array(X2)
```

```
y2= np.array(y2)

clf_linear = svm.SVC(kernel='linear')
clf_linear.fit(X2, y2)
plot_svm_decision_boundary(clf_linear, X2, y2, title="Linear Kernel SVM (Data 2)")

# Train a SVM classifier with a polynomial kernel
clf_poly = svm.SVC(kernel='poly', degree=3)
clf_poly.fit(X2, y2)
plot_svm_decision_boundary(clf_poly, X2, y2, title="Polynomial Kernel SVM (Data 2)")

# Train a SVM classifier with an RBF kernel
clf_rbf = svm.SVC(kernel='rbf')
clf_rbf.fit(X2, y2)
plot_svm_decision_boundary(clf_rbf, X2, y2, title="Gaussian Kernel SVM (Data 2)")

# define function to boundary for 3d
def plot_svm_3d(X, y, svc):
    from mpl_toolkits.mplot3d import Axes3D
    plt.figure()
    X = np.asarray(X)
    ax = plt.axes(projection='3d')
    # ax.scatter(X[:,0], X[:,1], X[:,2], c=y, cmap=plt.cm.Paired, edgecolors='k')
    ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, s=30, cmap=plt.cm.Paired, edgecolors='k')
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('X3')

    zz = lambda xx,yy: (-svc.intercept_[0]-svc.coef_[0][0]*xx-svc.coef_[0][1]*yy) / svc.coef_[0][2]
    tmpx = np.linspace(0, 70, 10)
    tmpy = np.linspace(0, 70, 10)
    xx,yy = np.meshgrid(tmpx,tmpy)
    ax.plot_surface(xx, yy, zz(xx,yy), alpha = 0.5, cmap = 'Purples')
    for ii in range(0,90,1):
        ax.view_init(elev=15, azim=ii)
    plt.title("Linear kernel on dataset 3 (3D)")
    plt.show()

clf_linear = svm.SVC(kernel='linear')
clf_linear.fit(X3, y3)
plot_svm_3d(X3, y3, clf_linear)
```

Result:
- Dataset 1

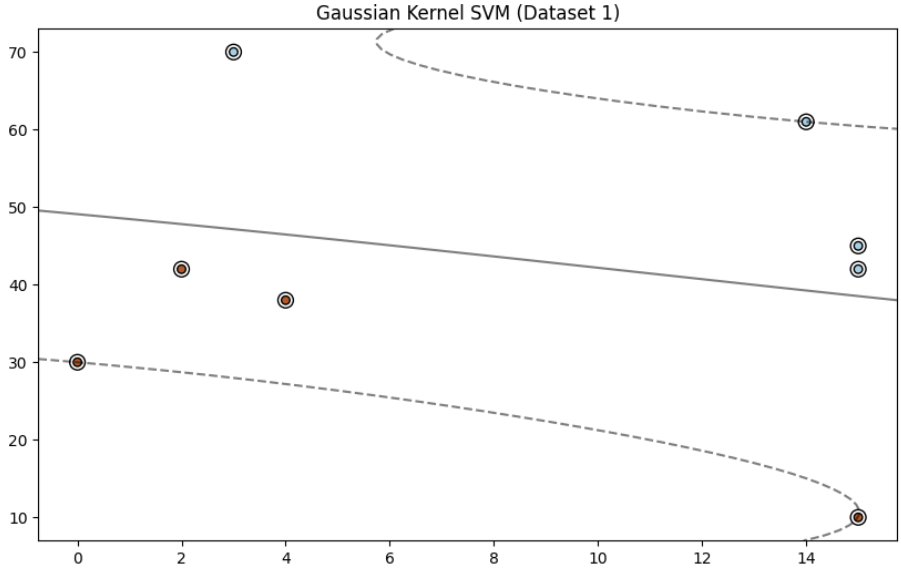


Linear Kernel SVM: Based on the graph above, a support vector classifier was fit to small data set (dataset 1). The hyperplane is shown as a solid line and the margins are those in dot lines. Points in blue represent normal class (0) and red are abnormal class (1). The support vectors are marked with a circle around them and are exactly on the margin lines, there are two support vectors for class 0 and one support vector for class 1. These points are the closest to the decision boundary, clearly defining the margins. We can see that the linear kernel can effectively separate the data into two classes, 4 observations on each side were correctly classified. However, it is worth to note that it might not be able to do as good (misclassify) with larger dataset if the data possesses a non-linear pattern.



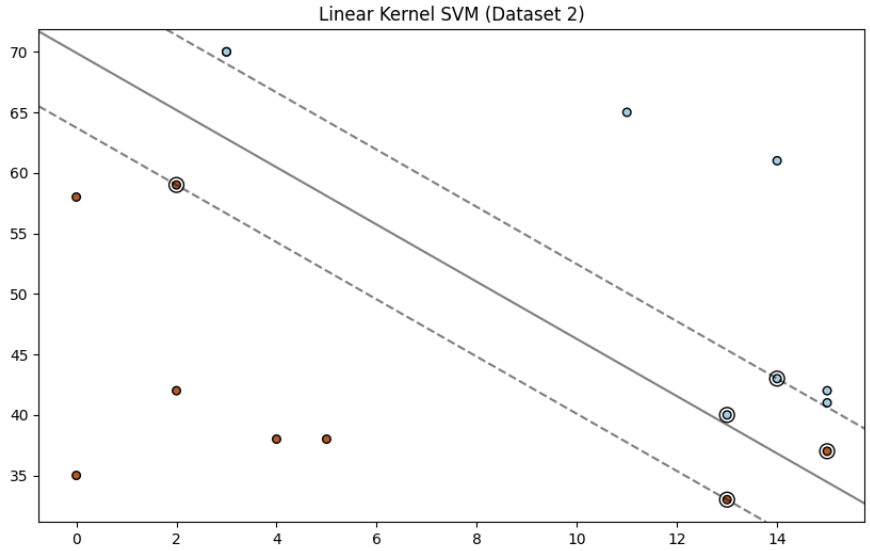
Polynomial Kernel SVM: The polynomial kernel introduces curves to the decision boundary, allowing a more flexible fit than the linear kernel. This can wrap more closely around one class, optimizing the margin. Similar to the linear model, support vectors are those marked in circle, but conversely, there are two support vectors of class 1 (red) and one support vector of class 0 (blue). This kernel also did a very good separation between classes the way linear kernel

did. Polynomial kernel encapsulates groups more tightly compared to the linear kernel since the margin is smaller, so it makes the separation more obvious compared to linear one. It seems to balance complexity and fitting, which might be advantageous in cases of slight non-linearity in the data distribution.

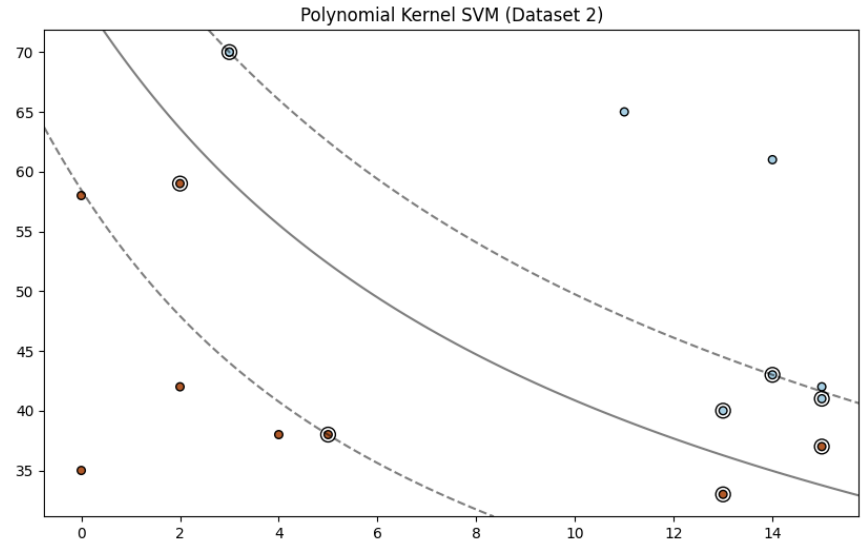


Gaussian (RBF) Kernel SVM: The RBF kernel allows for the most complex decision boundaries, which can flexibly wrap around data points. Based on the graph above, every data point is a support vector in which the decision boundary is overly complex and is fitting the training data too closely and many data points are within the margin (exceeded the decision boundary). Even though RBF kernel has high adaptability, potentially providing excellent fit for the training data but the risk is overfitting. In conclusion, RBF kernel did not do as good as linear and polynomial kernel for dataset 1.

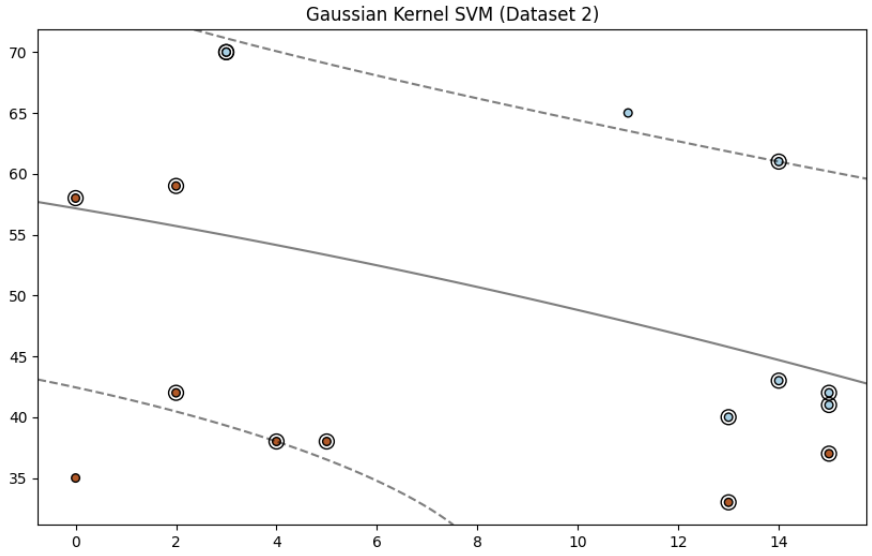
- Dataset 2:



Linear Kernel SVM: For larger dataset, visually, some data points near the margins suggest potential outliers or margin errors, which could affect the model's robustness and generalization to new data. The data points with circle are support vectors, three of them are of class 0 and two are of class 1. Moreover, there is one red data point of abnormal class (class 1) that falls into the side of hyperplane (of normal class - class 0) and two data points (one red and one blue) fell into the margin (wrong side). In addition, some blue points are close to the margin. So, with more examples it seems that linear kernel was not able to find a hyperplane that effectively separates the datapoints like it did for dataset 1 which leads to bad performance afterward that could be confirmed with the classification in the previous exercise (overfitting).

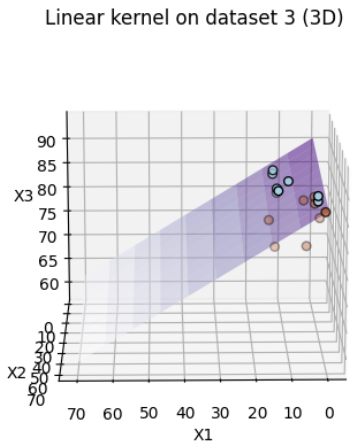


Polynomial Kernel SVM: According to the graph above, there are 8 support vectors in total, 4 each for both classes and they are those that would affect or influent the classification. However, we can see that there is one red point that locates on the other side of separation (wrong side of hyperplane). Polynomial kernel does not make a big difference from linear kernel in terms of separation; however, we can see that there is more space in terms of hyperplane separation which has bigger margins size.



Gaussian (RBF) Kernel SVM: The graph illustrates that this kernel allows the decision boundary to be flexible and adaptable to the data distribution. Also, RBF kernel is capable of maximizing the margin since the gaps of both sides are larger than two previous kernels. In terms of support vectors, one each for both classes are on the margin, the rest seem to be randomly distributed 2 red vectors and 4 blue vectors are on the wrong side of hyperplane respectively. However, due to the high flexibility of this kernel, only two observations are left but they are well classified and not too close to the margins as well.

- Dataset 3:



We can see that blue points and red points are well classified by the hyperplane (purple plane), so the linear kernel is good for linearly separable data and the number of support vectors around the separating hyperplane is lower than the RBF and polynomial kernels. It is efficient and can handle large datasets with high dimensionality. However, the linear kernel may not work well with non-linearly separable data, as it can only find an optimal linear hyperplane to classify the data. In addition, the margin of the linear kernel is small, so it may not be robust to noise in the data. For this reason, the result of the linear kernel prediction could be unstable for different datasets.

Conclusion

Comparison Across Datasets
- Dataset 1: Both linear kernel and polynomial provided a good separation between the two groups, so the RBF kernel is not necessary even if it has high adaptability and as we saw that it could lead to many potential problems given that all points are support vectors. Plus, polynomial kernel seemed to better at maximizing the margin because there were some points that really close to the margin for linear kernel.

- Dataset 2: The linear kernel's simplicity resulted in potentially underfitting with more complex distribution of data. We can see there are some points fell into the margin and also on the wrong side. Whereas, for polynomial kernel, it might be offering better fitting since the margin are larger but still will not provide a big difference compared to linear one. Lastly, the RBF kernel had the largest margin. Many points fell into the wrong side; however, these accept misclassifications gave us a better prediction after all.

- Dataset 3: The complexity increases with the addition of a third dimension, but the linear kernel still did a great job in terms of class separation which can be confirmed by the graph and prediction above. From the prediction, RBF and Polynomial have an equal rate of classification but lower than linear, however, since they are more flexible which could outperform the linear one when data get larger.

- Notice: when the distribution of data starts becoming more complex, we saw that there are points that fell into the margin and some of them are on the wrong side as well. This is because of the misclassification (soft margin or slack variables) that we are willing to make, so we would also like to take into account the regularization technique to

optimize or to avoid misclassifying (too much). Linear kernel seemed to be a good choice for this practical session since when data is not complex since it is the simplest. Polynomial and RBF kernel always had a better margin than linear kernel for this problem.

Annexe : Programme SVM sous Python-Anaconda

Exemple pour le SVM à noyau Linéaire : réutiliser ce modèle de programme ensuite en le modifiant pour les deux autres noyaux RBF et Polynomial.

```
"""

from pylab import *
import numpy as np
from sklearn import svm
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn import metrics


# Tests préliminaires de prise en main des commandes principales pour exécuter une prédiction par l'algorithme SVM sous Python :
X = [[0, 0], [1, 1]]
y = [0, 1]

clf = svm.SVC(kernel='linear')
clf.fit(X, y)



Val_pred1=clf.predict([[2., 2.]])
Val_pred2=clf.predict([[2., 0.]])
Val_pred3=clf.predict([[0., 0.]])
print(Val_pred1,Val_pred2,Val_pred3)

# get support vectors
SV=clf.support_vectors_
print(SV)
# get indices of support vectors
Index_SV=clf.support_
print(Index_SV)
# get number of support vectors for each class
Number_SV=clf.n_support_
print(Number_SV)



########### Données d'Apprentissage et de Test indépendantes pour le SVM ##########

#------- Jeu de données n°1 --------
#--- Apprentissage à partir de 8 exemples de dimension N=2 Feature X=(Mvt,Pouls)
X1 = [[15, 42], [15, 45], [14, 61], [3, 70], [0, 30], [15, 10], [4, 38], [2, 42]]
y1 = [0, 0, 0, 0, 1, 1, 1, 1]
```

```
# Données de Test (prédiction)
Lab_reels1 = [0, 1, 1, 1, 0]
#X_test1=[[15., 60.], [2., 42.], [4, 39], [2, 35], [15, 41]]
X_test1=[[15., 60.], [2., 22.], [4, 39], [0, 40], [15, 36]]

#------- Jeu de données n°2 --------
#--- Apprentissage à partir de 16 exemples de dimension N=2 Feature X=(Mvt,Pouls)
X2 = [[15, 42], [15, 41], [14, 61], [3, 70], [13, 40], [14, 43], [11, 65], [3, 70], [0, 58], [15, 37], [4, 38], [2, 42], [2, 59], [13, 33], [5, 38], [0, 35]]
y2 = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

# Données de Test (prédiction)
Lab_reels2 = [0, 0, 1, 1, 1, 1, 0, 0]
X_test2=[[15., 60.], [2., 42.], [4, 39], [2, 35], [15, 36], [14, 39], [3, 80], [3, 60]]

#------- Jeu de données n°3 --------
#--- Apprentissage à partir de 16 exemples de dimension N=3 Feature X=(Mvt,Pouls, SpO2)
X3 = [[15, 56, 92], [15, 58, 93], [14, 61, 90], [3, 70, 89], [13, 41, 86], [14, 70, 91], [11, 65, 92], [3, 70, 90], [0, 58, 85], [15, 37, 80], [4, 38, 75], [2, 42, 84], [2, 40, 85], [13, 33, 74], [5, 38, 84], [0, 35, 80]]
y3 = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

# Données de Test (prédiction)
Lab_ reels3 = [0, 0, 1, 1, 1, 1, 0, 0]
X_test3=[[15., 60., 90.], [2., 42., 89.], [4., 39., 75.], [2., 35., 80.], [15., 36., 82.], [14., 39., 79.], [3., 80., 91.], [3., 60.,93.]]


# Changement des variables (X, y) d'entrée pour l'apprentissage (données, labels) et des données (X_test, Lab_reels) de Test avec leur « vérité terrain » ou classe réelle (comparée ensuite avec les labels prédits par la fonction de décision).

X=X1
y=y1

Lab_reels=Lab_reels1
X_test=X_test1

#######################################
# Noyau Linéaire


# ------ Apprentissage ------
clf = svm.SVC(kernel='linear')
clf.fit(X, y)

# ou bien :
#linear_svc = svm.SVC(kernel='linear')
#svm_lin=linear_svc.fit(X,y)


# get support vectors
SV=clf.support_vectors_
print(SV)
# get indices of support vectors
Index_SV=clf.support_
print(Index_SV)
```

```
# get number of support vectors for each class
Number_SV=clf.n_support_
print(Number_SV)



# ------ Prediction  (Test) -----

Lab_pred_lin=clf.predict(X_test)

# ou bien :
#Lab_pred_lin=svm_lin.predict(X_test)

delta_Lab=Lab_pred_lin - Lab_reels
print('Erreurs Kernel Linéaire :',delta_Lab)
Nb_erreurs=sum(abs(delta_Lab))
Taux_erreurs=Nb_erreurs/size(Lab_reels)
Taux_Reco=(1-Taux_erreurs)*100
print('Taux_Reco Noyau Linéaire en % :',Taux_Reco)

print("Accuracy linear kernel :",metrics.accuracy_score(Lab_reels, Lab_pred_lin))



#===➔ Eléments de programme pour le tracé des hyperplan séparateurs pour données de dim=2 (question 6 facultative)


Y=array(X)


plt.scatter(Y[:, 0], Y[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```