Nom et prénoms du binôme : LY Sovanleng et THUO Menghor
ENSIIE
Date : 10/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 ?

Réponse : Support Vector Machine. Le principe de cette approche repose sur deux propriétés

- Maximiser la distance de la « marge ».

- Transformer l'espace de représentation des donnés d'entrées en un espace de plus grande dimension où il existe une séparatrice linéaire.

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).

. Problème Primal:

$$\begin{cases} \min \frac{1}{2}\|w\|^2 \\ \forall i,\ y_i(w.x_i+b) \geq 1 \end{cases} \quad \text{pour } i=1 \text{ à } L \text{ avec } L < M$$

. Critère quadratique sous contrainte de Lagrange:

$$L_p = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{L} \alpha_i y_i (w.x_i+b)$$

où $\alpha_i \geq 0$ sont les multiplicateurs de Lagrange.

- Passage du problème primal au problème dual par recherche d'un minorant du critère de Lagrange $L_p(.)$ dès lors noté $L_d(.)$,

$$\text{Max}\{L_d(\alpha)\} \leq \text{Min}\{L_p(w,b,\alpha)\} \quad \text{(dualité faible)}$$

$$\text{Max}\{L_d(\alpha)\} = \text{Min}\{L_p(w,b,\alpha)\} \quad \text{(dualité forte)}$$

avec les conditions KKT

$$\frac{\partial}{\partial w} L_p = w - \sum_i \alpha_i y_i x_i = 0$$

$$w = \sum_i \alpha_i y_i x_i$$

$$\frac{\partial}{\partial b} L_p = -\sum_i \alpha_i y_i = 0$$

$$\sum_i \alpha_i y_i = 0$$

$$\Rightarrow L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j x_i . x_j$$

Dual Problème :

$$\begin{cases} \max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j x_i . x_j \\ \forall i,\ 0 \leq \alpha_i \leq C \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases}$$

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 ?
Réponse :

- Noyau Linéaire: $K(x, x') = x.x'$
- Noyau polynomial : $K(x, x') = (x.x')^d$
- Noyau gaussien ou RBF : $K(x, x') = e^{-||x-x'||^2/\sigma}$

La condition de Mercer est vérifiée une fonction noyau parce qu'il existe d'un devt en séries de convergence absolue, uniforme et semi-définie positive.

- Noyau polynomial d'ordre 1: $K(x, x') = x.x' + c$
  Ce noyau revient à un noyau linéaire avec un terme constant c. C'est équivalent à l'algorithme de régression logistique lorsqu'il est utilisé dans un contexte de classification linéaire.
- Noyau Sigmoïde: $K(x, x') = (k\,x.x' + \delta)$
  Ce noyau utilisé dans certains types de réseaux de neurones(perceptrons multicouches), transforme les produits scalaires via la fonction tangente hyperbolique, liant le SVM à des modèles de neurones artificiels.

La fonction de décision: $f(x) = sgn(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b)$

- α_i sont les multiplicateurs de Lagrange obtenus par la résolution du problème d'optimisation dual des SVM.
- y_i sont les étiquettes de classe des vecteurs de support x_i
- b est le biais.
- K est la fonction noyau

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***
Réponse :
Les donnés semblent être des enregistrements horodatés, qui incluent des mesures issues probablement d'un dispositif de télévigilance médicale. Chaque ligne correspond à un enregistrement à un moment donné.
   - Première colonne: Index utilisateur, souvent 0, qui peut représenter un état normal.
   - Deuxième colonne: date
   - Troisième colonne : heure
   - Quatrième colonne : batterie qui est souvent 0, qui indique normal et 1 est déchargée
   - Cinquième colonne: chute également souvent 0, qui indique normal et 1 est chute seuil accélératrice
   - Sixième colonne: posture, 0 debout et 1 caché
   - Septième colonne: button appelle, 0 pas d'appelle et 1 appelle
   - Huitième colonne: l'activité de patient
   - Neuvième colonne: 40 bpm < pouls < 255 bpm
     Les paramètres différentient les données normales de celles anormales (situations de détresses) sont activités et pouls.

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

Exemple de données extrait du fichier en question :

Indice de l'utilisateur |Date |Heure| battery | chute | posture | button appelle| activity | pouls

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 person est couché, activités est 0, mesure de pouls < 40 bpm

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 ?

N.B. : Quelques consignes pour vous aider à démarrer avec les outils de Scikit-learn :
- vous pouvez consulter : https://scikit-learn.org/stable/modules/svm.html
- 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.

Réponse :

Code:

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

#------- 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]]
#--- 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.]]

# fonction pour apprentissage
def appren_svm(X, y, kernel_type):
    svm_mod = svm.SVC(kernel=kernel_type)
    svm_fit = svm_mod.fit(X,y)
    # get support vectors
    SV=svm_fit.support_vectors_
    print(SV)
    # get indices of support vectors
    Index_SV=svm_fit.support_
    print(Index_SV)
    # get number of support vectors for each class
    Number_SV=svm_fit.n_support_
    print(Number_SV)
    return svm_fit

# fonction pour prediction (Test)
def predict_svm(X_test, Lab_reels, model):
    Lab_pred=model.predict(X_test)
    delta_Lab=Lab_pred - Lab_reels
    print('Erreurs :',delta_Lab)
    Nb_erreurs=sum(abs(delta_Lab))
    Taux_erreurs=Nb_erreurs/len(Lab_reels)
    Taux_Reco=(1-Taux_erreurs)*100
    print('Taux_Reco en % :',Taux_Reco)
    print("Accuracy:",metrics.accuracy_score(Lab_reels, Lab_pred))
    return Lab_pred

# Noyau Linéaire
print('Noyau Linéaire\n')
```

```
#------- Jeu de données n°1 --------
print('Jeu de données n°1: \n')
# ------ Apprentissage ------
svm_lin1 = appren_svm(X1, y1, 'linear')
# ------ Prediction ------
lin_pred1 = predict_svm(X_test1, Lab_reels1, svm_lin1)
print('\n')

#------- Jeu de données n°2 --------
print('Jeu de données n°2: \n')
# ------ Apprentissage ------
svm_lin2 = appren_svm(X2, y2, 'linear')
# ------ Prediction ------
lin_pred2 = predict_svm(X_test2, Lab_reels2, svm_lin2)
print('\n')

#------- Jeu de données n°3 --------
print('Jeu de données n°3: \n')
# ------ Apprentissage ------
svm_lin3 = appren_svm(X3, y3, 'linear')
# ------ Prediction ------
lin_pred3 = predict_svm(X_test3, Lab_reels3, svm_lin3)

# Noyau RBF (Gaussien)
print('Noyau RBF (Gaussien)\n')
#------- Jeu de données n°1 --------
print('Jeu de données n°1: \n')
# ------ Apprentissage ------
svm_rbf1 = appren_svm(X1, y1, 'rbf')
# ------ Prediction ------
rbf_pred1 = predict_svm(X_test1, Lab_reels1, svm_rbf1)
print('\n')

#------- Jeu de données n°2 --------
print('Jeu de données n°2: \n')
# ------ Apprentissage ------
svm_rbf2 = appren_svm(X2, y2, 'rbf')
# ------ Prediction ------
rbf_pred2 = predict_svm(X_test2, Lab_reels2, svm_rbf2)
print('\n')

#------- Jeu de données n°3 --------
print('Jeu de données n°3: \n')
# ------ Apprentissage ------
svm_rbf3 = appren_svm(X3, y3, 'rbf')
# ------ Prediction ------
rbf_pred3 = predict_svm(X_test3, Lab_reels3, svm_rbf3)

# Noyau Polynomial
print('Noyau Polynomial\n')
#------- Jeu de données n°1 --------
print('Jeu de données n°1: \n')
# ------ Apprentissage ------
svm_poly1 = appren_svm(X1, y1, 'poly')
```

```
# ------ Prediction ------
poly_pred1 = predict_svm(X_test1, Lab_reels1, svm_poly1)
print('\n')

#------- Jeu de données n°2 --------
print('Jeu de données n°2: \n')
# ------ Apprentissage ------
svm_poly2 = appren_svm(X2, y2, 'poly')
# ------ Prediction ------
poly_pred2 = predict_svm(X_test2, Lab_reels2, svm_poly2)
print('\n')

#------- Jeu de données n°3 --------
print('Jeu de données n°3: \n')
# ------ Apprentissage ------
svm_poly3 = appren_svm(X3, y3, 'poly')
# ------ Prediction ------
poly_pred3 = predict_svm(X_test3, Lab_reels3, svm_poly3)
```

Résultats :

a) Noyau Linéaire

Jeu de données n°1:

```
[[15. 42.]
 [ 3. 70.]
 [ 4. 38.]]
[0 3 6]
[2 1]
Erreurs : [0 0 0 0 0]
Taux_Reco en % : 100.0
Accuracy: 1.0
```

Jeu de données n°2:

```
[[13. 40.]
 [14. 43.]
 [15. 37.]
 [ 2. 59.]
 [13. 33.]]
[ 4  5  9 12 13]
[2 3]
Erreurs : [ 0  1  0  0 -1 -1  0  1]
Taux_Reco en % : 50.0
Accuracy: 0.5
```

Jeu de données n°3:

```
[[13. 41. 86.]
 [ 0. 58. 85.]
 [15. 37. 80.]
 [ 5. 38. 84.]]
[ 4  8  9 14]
```

[1 3]
Erreurs : [0 0 0 0 0 0 0 0]
Taux_Reco en % : 100.0
Accuracy: 1.0

b) Noyau RBF (Gaussien)

Jeu de données n°1:

[[15. 42.]
 [15. 45.]
 [14. 61.]
 [3. 70.]
 [0. 30.]
 [15. 10.]
 [4. 38.]
 [2. 42.]]
[0 1 2 3 4 5 6 7]
[4 4]
Erreurs : [0 0 0 0 1]
Taux_Reco en % : 80.0
Accuracy: 0.8

Jeu de données n°2:

[[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.]]
[0 1 2 3 4 5 7 8 9 10 11 12 13 14]
[7 7]
Erreurs : [0 1 0 0 0 0 0 0]
Taux_Reco en % : 87.5
Accuracy: 0.875

Jeu de données n°3:

[[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.]]
[0 1 2 3 4 6 7 8 9 10 11 12 14]
[7 6]
Erreurs : [0 1 0 0 0 0 0 0]
Taux_Reco en % : 87.5
Accuracy: 0.875

c) Noyau Polynomial

Jeu de données n°1:

[[15. 42.]
[4. 38.]
[2. 42.]]
[0 6 7]
[1 2]
Erreurs : [0 0 0 0 0]
Taux_Reco en % : 100.0
Accuracy: 1.0

Jeu de données n°2:

[[15. 41.]
[13. 40.]
[14. 43.]
[3. 70.]
[15. 37.]
[2. 59.]
[13. 33.]
[5. 38.]]
[1 4 5 7 9 12 13 14]
[4 4]
Erreurs : [0 1 0 0 -1 -1 0 0]
Taux_Reco en % : 62.5
Accuracy: 0.625

Jeu de données n°3:

[[3. 70. 89.]
[13. 41. 86.]
[0. 58. 85.]
[15. 37. 80.]]
[3 4 8 9]
[2 2]
Erreurs : [0 1 0 0 0 0 0 0]
Taux_Reco en % : 87.5
Accuracy: 0.875

Summary

Le tableau ci-dessous montre le taux de reconnaissance de chaque donnée en utilisant différents noyaux.

Noyau\Données	Jeu de données n°1	Jeu de données n°2	Jeu de données n°3
Noyau Linéaire	100%	50%	100%
Noyau RBF(Gaussien)	80%	87.5%	87.5%
Noyau Polynomial	100%	62.5%	87.5%

Pour le jeu de données n°1, tous les types de noyau ont un taux de reconnaissance élevé (100 % pour le noyau linéaire et le noyau polynomial), ce qui signifie que les SVM ont parfaitement séparé les différentes classes.

Pour le jeu de données n°2, le taux de reconnaissance est inférieur à 100% pour tous les types de noyau. Cela s'explique par le fait que ce jeu de données est plus complexe que le précédent, avec plus d'exemples que le jeu de données n°1.

Le choix du noyau impacte significativement la performance du modèle SVM. Le noyau linéaire est efficace lorsque les données sont linéairement séparables ; nous observons que les résultats sont les plus précis avec les données 1 et données 3. Le noyau RBF est plus adapté pour les données présentant des distributions complexes, ce qui est également démontré par les hautes performances sur les données 2 et données 3. Le noyau polynomial peut améliorer la séparation dans certains cas, mais peut aussi mener à l' overfitting, mais on remarque sur les données et données 3 donnent les plus accurate.

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 ?
Réponse :

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

Répondre :

Code :

```
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.inspection import DecisionBoundaryDisplay

# fonction pour représenter l'hyperplan séparateur à vaste marge en 2-dimension
def seperate_hyperlane_2d(X, y, clf):

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

    # plot the decision function
    ax = plt.gca()
    DecisionBoundaryDisplay.from_estimator(
```

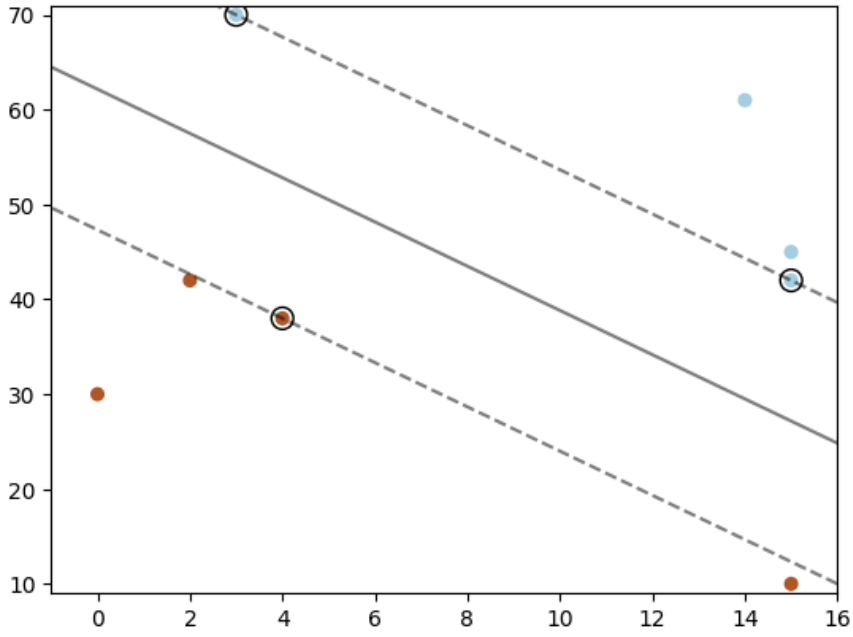
```
        clf,
        X,
        plot_method="contour",
        colors="k",
        levels=[-1, 0, 1],
        alpha=0.5,
        linestyles=["--", "-", "--"],
        ax=ax,
    )
    # plot support vectors
    ax.scatter(
        clf.support_vectors_[:, 0],
        clf.support_vectors_[:, 1],
        s=100,
        linewidth=1,
        facecolors="none",
        edgecolors="k",
    )
    plt.show()

# fonction pour représenter l'hyperplan séparateur en 3-dimension avec kernel linéaire
from mpl_toolkits.mplot3d import Axes3D
def seperate_hyperlane_3d(X, y, clf):
    # Extract the coefficients of the hyperplane equation
    w = clf.coef_[0]
    b = clf.intercept_

    X = np.asarray(X)
    # Create a 3D plot of the data points and the hyperplane
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)

    # Define the range of the plot axes
    xx, yy = np.meshgrid(np.linspace(0, 100, 20), np.linspace(0, 100, 20))
    zz = (-w[0]*xx - w[1]*yy - b) / w[2]

    # Plot the hyperplane
    ax.plot_surface(xx, yy, zz, alpha=0.2)

    # Set the labels for the plot axes
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('X3')

    # Show the plot
    plt.show()
## SVM avec le noyau Linéaire
# Jeu de données n°1
seperate_hyperlane_2d(X1, y1, svm_lin1)

# Jeu de données n°2
seperate_hyperlane_2d(X2, y2, svm_lin2)
```

```
# Jeu de données n°3
seperate_hyperlane_3d(X3, y3, svm_lin3)


## SVM avec le noyau RBF
# Jeu de données n°1
seperate_hyperlane_2d(X1, y1, svm_rbf1)

# Jeu de données n°2
seperate_hyperlane_2d(X2, y2, svm_rbf2)


## SVM avec le noyau polynomial
# Jeu de données n°1
seperate_hyperlane_2d(X1, y1, svm_poly1)

# Jeu de données n°2
seperate_hyperlane_2d(X2, y2, svm_poly2)
```

Résultats:

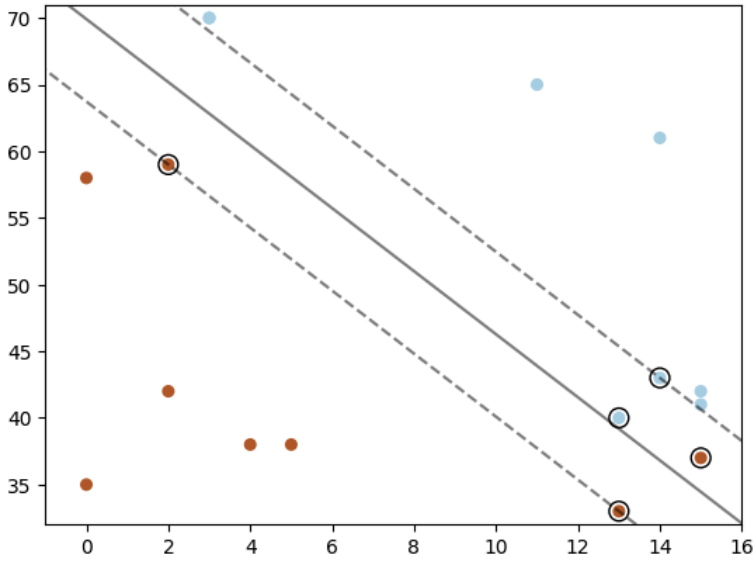
a) Noyau Linéaire :

Jeu de données n°1 : (2-dimension)



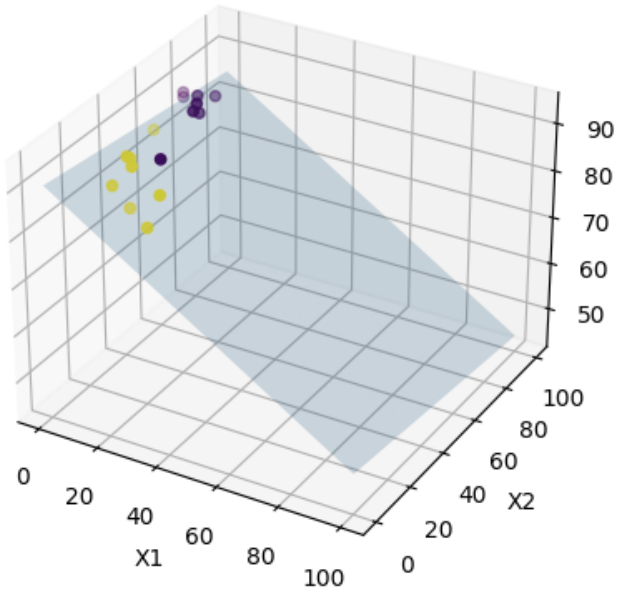
Sur le graphique ci-dessus, on peut voir qu'on peut tracer une ligne (hyperplan) pour séparer correctement les données en 2 classes donc le modèle SVM à noyau linéaire a très bien fonctionné avec ce premier jeu de données. On peut aussi voir qu'il n'y a pas de point dans la marge et pas de point de mauvais classement. La précision de ce modèle sur l'ensemble de test est de 100 %. Ainsi, le SVM avec noyau linéaire est très bien pour des données linéairement séparables avec moins d'exemples.

Jeu de données n°2 : (2-dimension)



Sur le graphique ci-dessus, le jeu de données n°2 avec plus d'exemples que la première n'est pas des données linéairement séparables, donc on ne peut pas trouver la ligne (hyperplan) pour séparer correctement ces données. Par conséquent, le noyau linéaire n'est pas très bon pour ce jeu de données., on peut voir aussi qu'il y a deux points dans la marge et un point de mauvais classement. Le nombre de vecteurs support augmente et la précision de ce modèle sur le jeu de données de test diminue à 50 %. Ainsi, on peut voir que lors de l'apprentissage sur un jeu de données non linéairement séparables avec le noyau linéaire, des points dans la marge et des points de mauvaise classification commencent à apparaître. Ensuite, le modèle SVM avec le noyau linéaire aura une précision réduite (50% dans ce cas).

Jeu de données n°3 : (3-dimension)

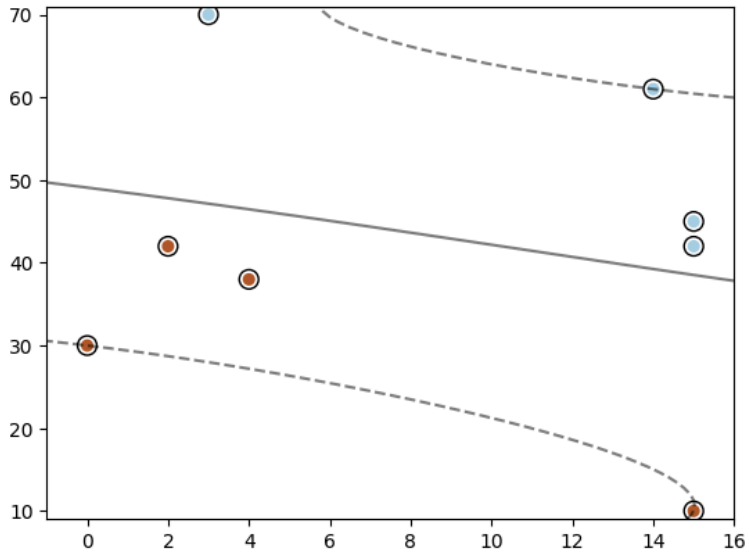


Quand la dimension augmentait dans le troisième ensemble de données, sur le graphique, on peut trouver l'hyperplan pour séparer correctement ces données.

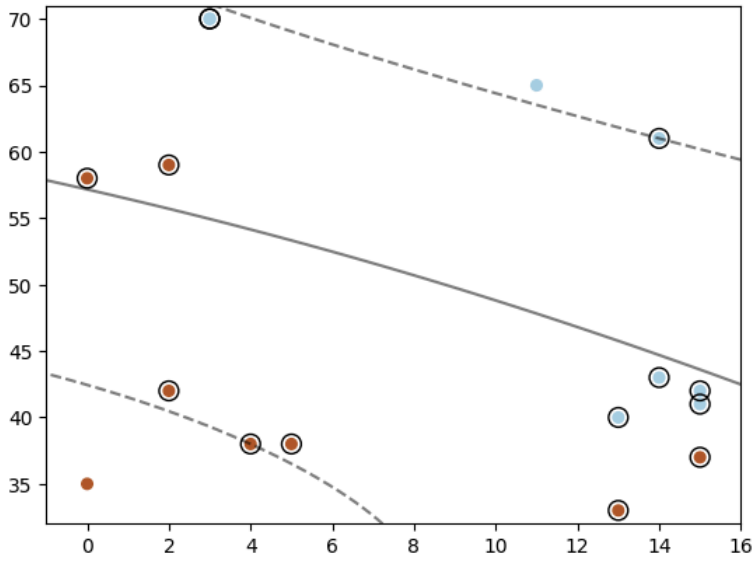
En conclusion, le noyau linéaire est bon pour les données séparables linéairement et le nombre de vecteurs supports autour de l'hyperplan séparateur est inférieur aux noyau RBF et noyau polynomial. Il est efficace en termes de calcul et peut gérer de grands jeux de données avec une grande dimension. Cependant, le noyau linéaire peut ne pas bien fonctionner avec des données non linéairement séparables, car il ne peut trouver qu'un hyperplan linéaire optimal pour classer les données. En outre, la marge du noyau linéaire est petite, de sorte qu'il pourrait ne pas être robuste au bruit dans les données. Pour cette raison, le résultat de la prédiction du noyau linéaire pourrait être instable pour différents ensembles de données.

b) Noyau RBF

Jeu de données n°1



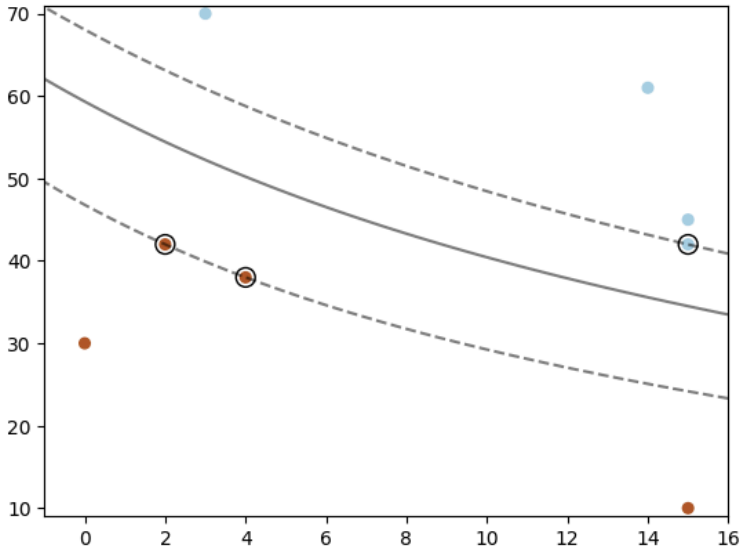
Jeu de données n°2



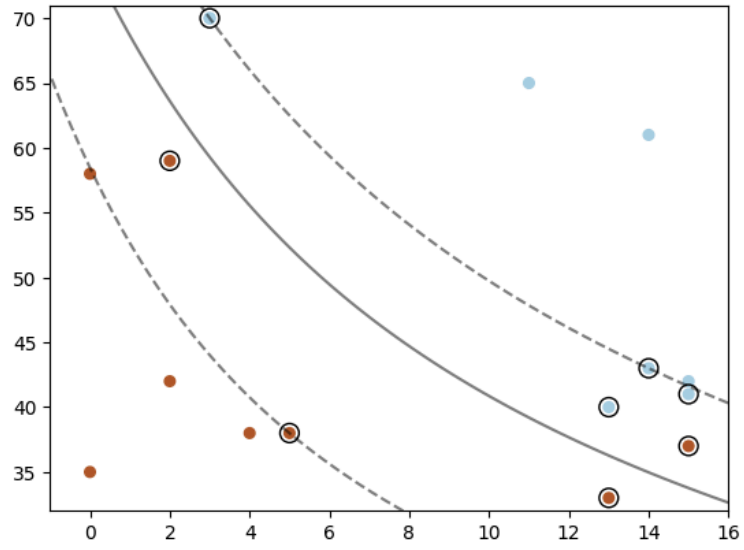
Sur les graphiques ci-dessus, on peut voir que le noyau RBF crée un hyperplan optimal non linéaire pour séparer les données et le nombre de vecteurs supports autour de l'hyperplan séparateur est supérieur au noyau linéaire. Ainsi, le noyau RBF peut classer bien des données séparables hautement non linéaires que le noyau linéaire et peut fournir une limite de décision plus flexible que le noyau linéaire. Cela peut être vu à partir de l'ensemble de données 2, la précision lors de la prédiction sur l'ensemble de données de test est de 87,5 % (plus élevée que le noyau linéaire avec 50%). La plus grande marge dans le noyau RBF contribue également à augmenter la capacité de généralisation du modèle et à le rendre plus robuste au bruit dans les données. Donc, le noyau RBF renvoie des résultats de prédiction beaucoup plus stables sur les 3 jeux de données de test (80 %, 87,5 %, 87,5 %) que le noyau linéaire et le noyau polynomial. Cependant, cela peut être coûteux en calcul et choisir la bonne valeur pour le paramètre gamma peut être difficile.

c) Noyau Polynomial

Jeu de données n°1



Jeu de données n°2



Sur les graphiques ci-dessus, on peut voir que le noyau polynomial crée un hyperplan optimal non linéaire pour séparer les données et la marge est modérément grande. Le nombre de vecteurs supports autour de l'hyperplan séparateur est aussi supérieur au noyau linéaire. Semblable au noyau RBF, le noyau polynomial peut également classer bien des données séparables modérément non linéaires et peut fournir une limite de décision plus flexible que le noyau linéaire. Cependant, choisir le bon degré du polynôme peut être difficile, et un "overfitting" peut se produire si le degré est trop élevé. Le résultat du noyau polynomial est plus stable que le noyau linéaire mais moins stable que le noyau RBF pour prédire différentes données.

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))
```