TP - SVM

April 7, 2023

Pattern Recognition and Biometrics

TP SVM

Library

[1]:
```
from pylab import *
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

Test Préliminaire

[3]:
```
# 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)
```

```
[1] [1] [0]
[[0. 0.]
 [1. 1.]]
[0 1]
[1 1]
```

Données

[4]:
```
########### 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=(Mut,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=(Mut,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=(Mut,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.]]
```

JEU DE DONNES 1

Noyau Linéaire

[5]:
```
# 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))
```

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

Noyau Polynomial

[6]:
```
# 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='poly')
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))
```

```
[[15. 42.]
 [ 4. 38.]
 [ 2. 42.]]
[0 6 7]
[1 2]
Erreurs Kernel Linéaire : [0 0 0 0 0]
Taux_Reco Noyau Linéaire en % : 100.0
Accuracy linear kernel : 1.0
```

Noyau RBF

[7]:
```
# 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='rbf')
clf.fit(X, y)

# ou bien :
```

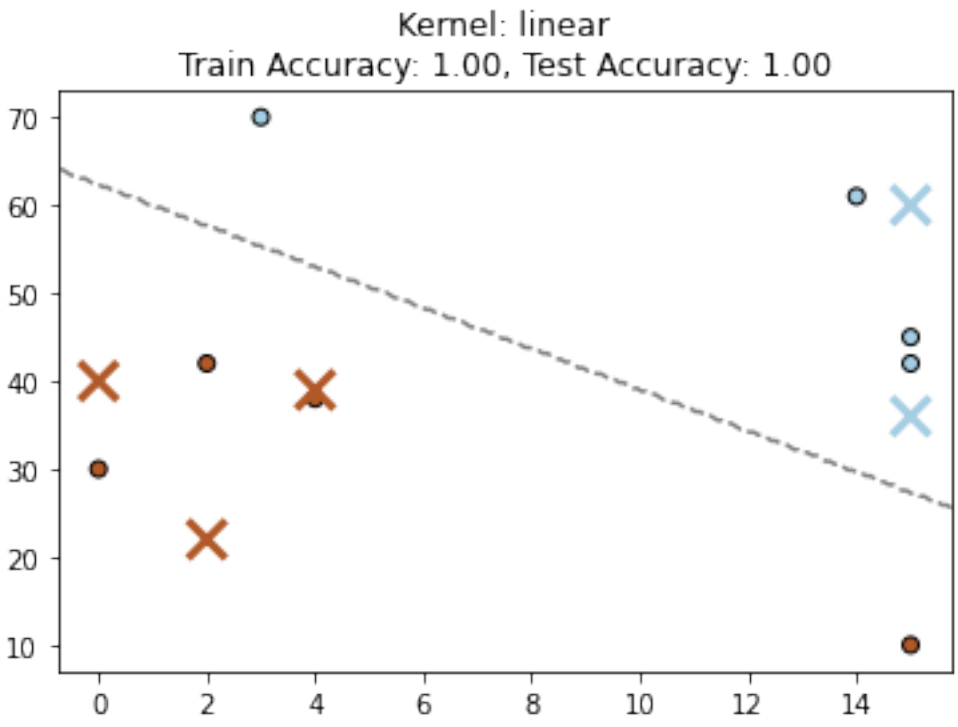
```
#linear_svc = svm.SVC(kernel='linear')
#svm_lin=linear_svc.fit(X,y)

# 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))
```

```
[[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 Kernel Linéaire : [0 0 0 0 1]
Taux_Reco Noyau Linéaire en % : 80.0
Accuracy linear kernel : 0.8
```

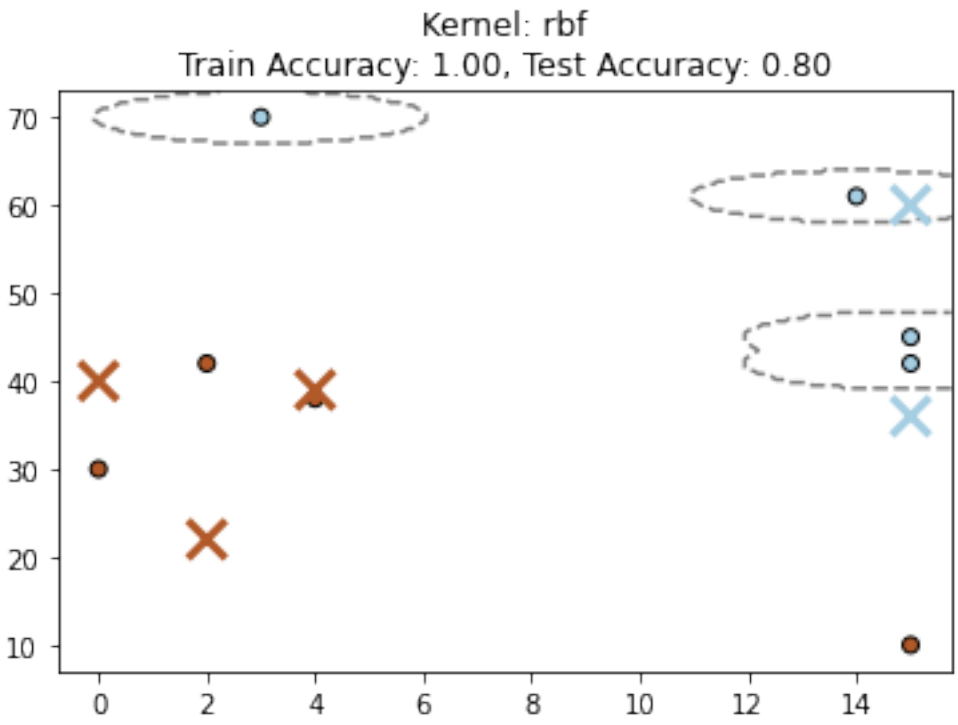
[20]:
```
# Données d'apprentissage
X_train = np.array([[15, 42], [15, 45], [14, 61], [3, 70], [0, 30], [15, 10],␣
 ↪[4, 38], [2, 42]])
```

```
y_train = np.array([0, 0, 0, 0, 1, 1, 1, 1])

# Données de test
X_test = np.array([[15., 60.], [2., 22.], [4, 39], [0, 40], [15, 36]])
y_test = np.array([0, 1, 1, 1, 0])

# Définition des modèles avec différents noyaux
models = [
    svm.SVC(kernel='linear'),
    svm.SVC(kernel='rbf', gamma=1),
    svm.SVC(kernel='poly', degree=3)
]

# Entrainement et prédiction avec chaque modèle
for model in models:
    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    train_accuracy = accuracy_score(y_train, y_pred_train)
    test_accuracy = accuracy_score(y_test, y_pred_test)

    # Tracé de la frontière de décision
    plt.figure()
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.Paired,␣
 ↪edgecolors='k')
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.Paired,␣
 ↪marker='x', s=200, linewidths=3, zorder=10)
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 200), np.
 ↪linspace(ylim[0], ylim[1], 200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],␣
 ↪colors='k', alpha=0.5)
    plt.title(f"Kernel: {model.kernel}\nTrain Accuracy: {train_accuracy:.2f},␣
 ↪Test Accuracy: {test_accuracy:.2f}")
    plt.show()
```

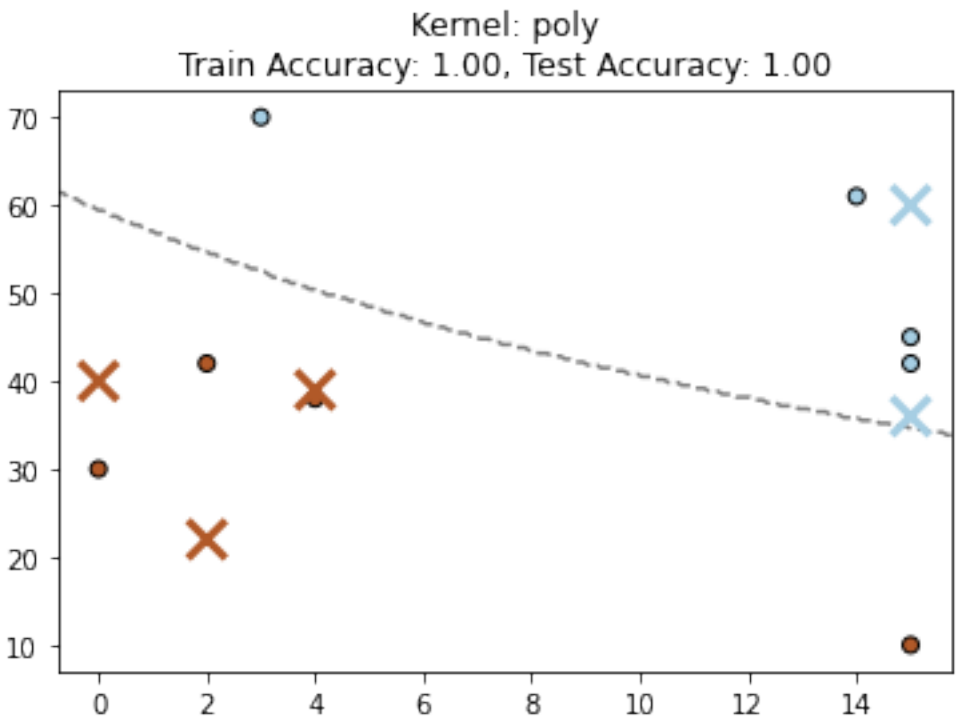
```
C:\Users\pierr\AppData\Local\Temp/ipykernel_7336/221401921.py:34: UserWarning:
No contour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],
colors='k', alpha=0.5)
```




```
C:\Users\pierr\AppData\Local\Temp/ipykernel_7336/221401921.py:34: UserWarning:
No contour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],
colors='k', alpha=0.5)
```




```
C:\Users\pierr\AppData\Local\Temp/ipykernel_7336/221401921.py:34: UserWarning:
No contour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],
colors='k', alpha=0.5)
```



JEU DE DONNES 2

Noyau Linéaire

[9]:
```
# 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=X2
y=y2

Lab_reels=Lab_reels2
X_test=X_test2

#####################################
# 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))
```

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

Noyau Polynomial

[10]:
```
# 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=X2
y=y2
```

```
Lab_reels=Lab_reels2
X_test=X_test2

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

# ------ Apprentissage ------
clf = svm.SVC(kernel='poly')
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))
```

```
[[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 Kernel Linéaire : [ 0  1  0  0 -1 -1  0  0]
Taux_Reco Noyau Linéaire en % : 62.5
Accuracy linear kernel : 0.625
```

Noyau RBF

[11]:
```
# 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=X2
y=y2

Lab_reels=Lab_reels2
X_test=X_test2

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

# ------ Apprentissage ------
clf = svm.SVC(kernel='rbf')
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
```

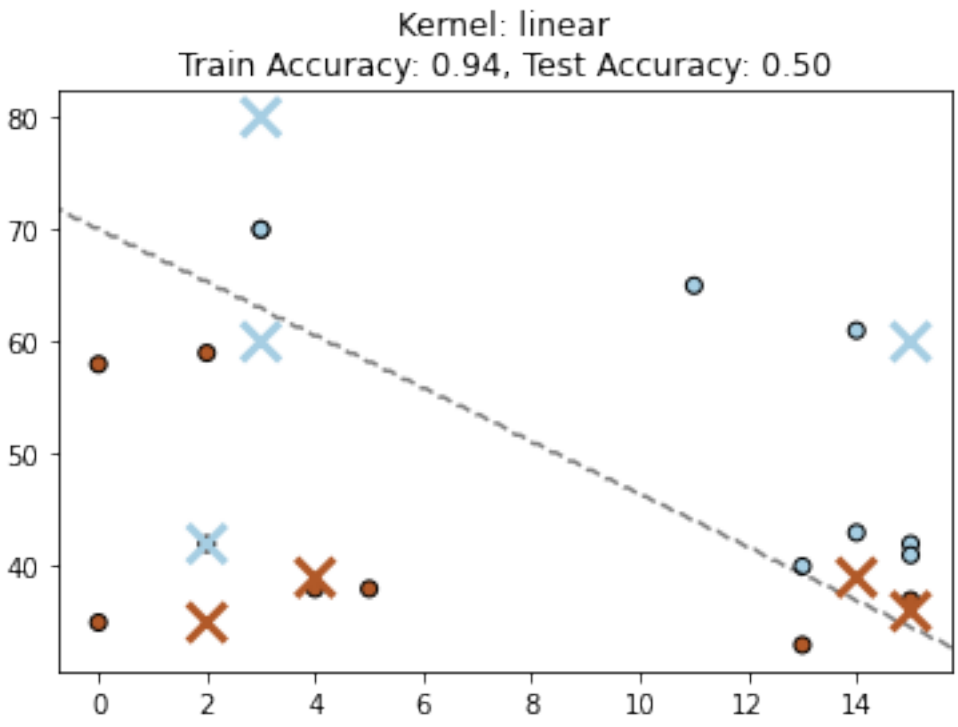
```
print('Erreurs Kernel Linéaire :',delta_Lab)
Nb_erreurs=sum(abs(delta_Lab))
Taux_erreurs=Nb_erreurs/size(Lab_reels)
Taux_Reco=(1-Taux_erreurs)*100
print('Taux_Reco Noyau Linéaire en % :',Taux_Reco)

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

```
[[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 Kernel Linéaire : [0 1 0 0 0 0 0 0]
Taux_Reco Noyau Linéaire en % : 87.5
Accuracy linear kernel : 0.875
```

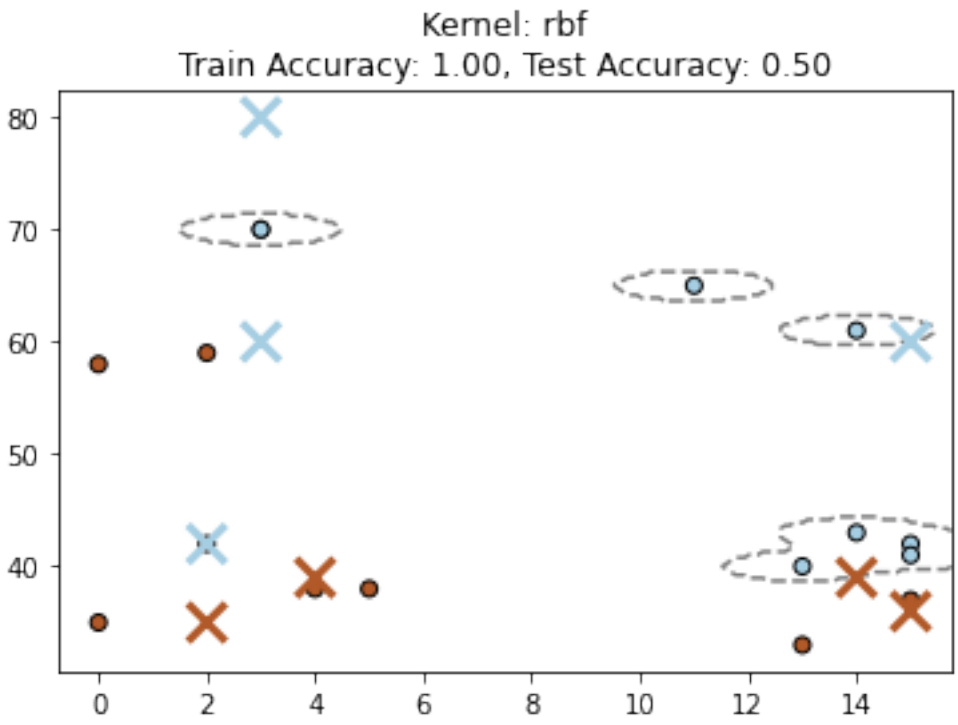
[12]:
```
# Données d'apprentissage
X_train = np.array([[15, 42], [15, 41], [14, 61], [3, 70], [13, 40], [14, 43],␣
 ↪[11, 65], [3, 70], [0, 58], [15, 37], [4, 38], [2, 42], [2, 59], [13, 33],␣
 ↪[5, 38], [0, 35]])
y_train = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1])

# Données de test
X_test = np.array([[15, 60], [2, 42], [4, 39], [2, 35], [15, 36], [14, 39], [3,␣
 ↪80], [3, 60]])
y_test = np.array([0, 0, 1, 1, 1, 1, 0, 0])

# Définition des modèles avec différents noyaux
models = [
    svm.SVC(kernel='linear'),
    svm.SVC(kernel='rbf', gamma=1),
    svm.SVC(kernel='poly', degree=3)
]
```

```
# Entrainement et prédiction avec chaque modèle
for model in models:
    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    train_accuracy = accuracy_score(y_train, y_pred_train)
    test_accuracy = accuracy_score(y_test, y_pred_test)

    # Tracé de la frontière de décision
    plt.figure()
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.Paired,␣
 ↪edgecolors='k')
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.Paired,␣
 ↪marker='x', s=200, linewidths=3, zorder=10)
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 200), np.
 ↪linspace(ylim[0], ylim[1], 200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],␣
 ↪colors='k', alpha=0.5)
    plt.title(f"Kernel: {model.kernel}\nTrain Accuracy: {train_accuracy:.2f},␣
 ↪Test Accuracy: {test_accuracy:.2f}")
    plt.show()
```

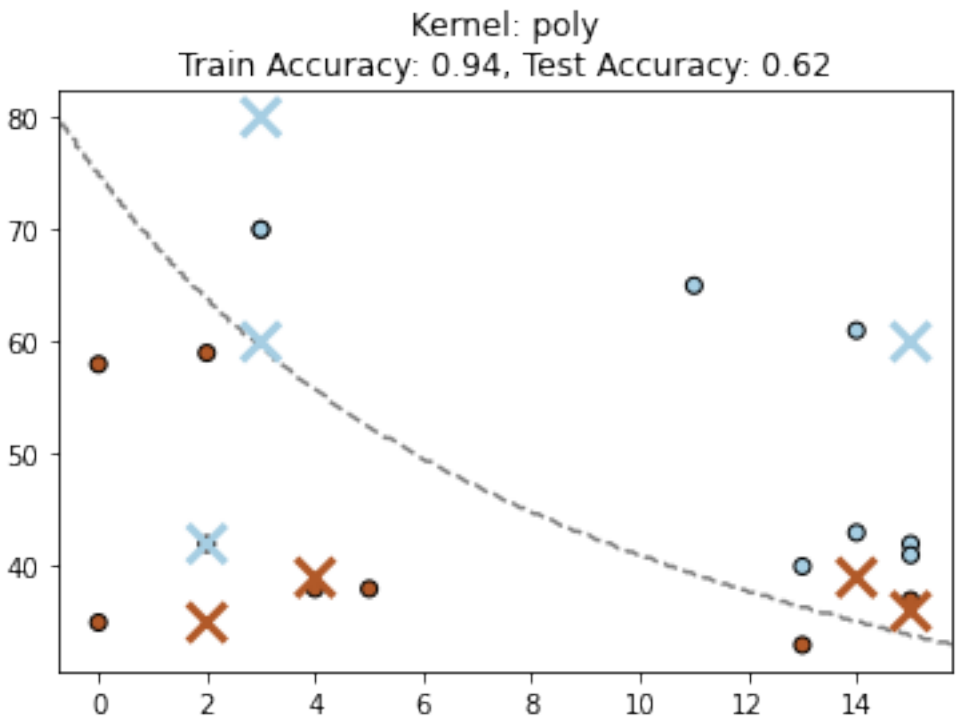
```
C:\Users\pierr\AppData\Local\Temp/ipykernel_7336/438834391.py:34: UserWarning:
No contour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],
colors='k', alpha=0.5)
```




```
C:\Users\pierr\AppData\Local\Temp/ipykernel_7336/438834391.py:34: UserWarning:
No contour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],
colors='k', alpha=0.5)
```




```
C:\Users\pierr\AppData\Local\Temp/ipykernel_7336/438834391.py:34: UserWarning:
No contour levels were found within the data range.
  plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'],
colors='k', alpha=0.5)
```



JEU DE DONNES 3

Noyau Linéaire

[13]:
```
# 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=X3
y=y3

Lab_reels=Lab_reels3
X_test=X_test3

#####################################
# 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))
```

```
[[13. 41. 86.]
 [ 0. 58. 85.]
 [15. 37. 80.]
 [ 5. 38. 84.]]
[ 4  8  9 14]
[1 3]
Erreurs Kernel Linéaire : [0 0 0 0 0 0 0 0]
Taux_Reco Noyau Linéaire en % : 100.0
Accuracy linear kernel : 1.0
```

Noyau Polynomial

[14]:
```
# 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=X3
y=y3

Lab_reels=Lab_reels3
```

```
X_test=X_test3

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

# ------ Apprentissage ------
clf = svm.SVC(kernel='poly')
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))
```

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

```
Accuracy linear kernel : 0.875
```

Noyau RBF

[15]:
```
# 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=X3
y=y3

Lab_reels=Lab_reels3
X_test=X_test3

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

# ------ Apprentissage ------
clf = svm.SVC(kernel='rbf')
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)
```

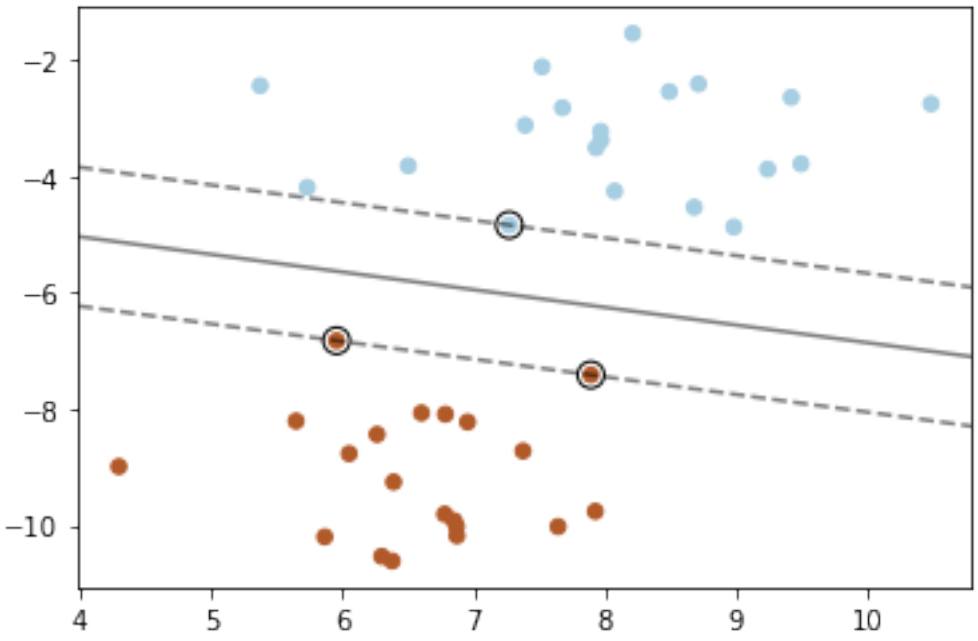
```
print("Accuracy linear kernel :",metrics.accuracy_score(Lab_reels,␣
 ↪Lab_pred_lin))
```

```
[[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 Kernel Linéaire : [0 1 0 0 0 0 0 0]
Taux_Reco Noyau Linéaire en % : 87.5
Accuracy linear kernel : 0.875
```

Visualisation

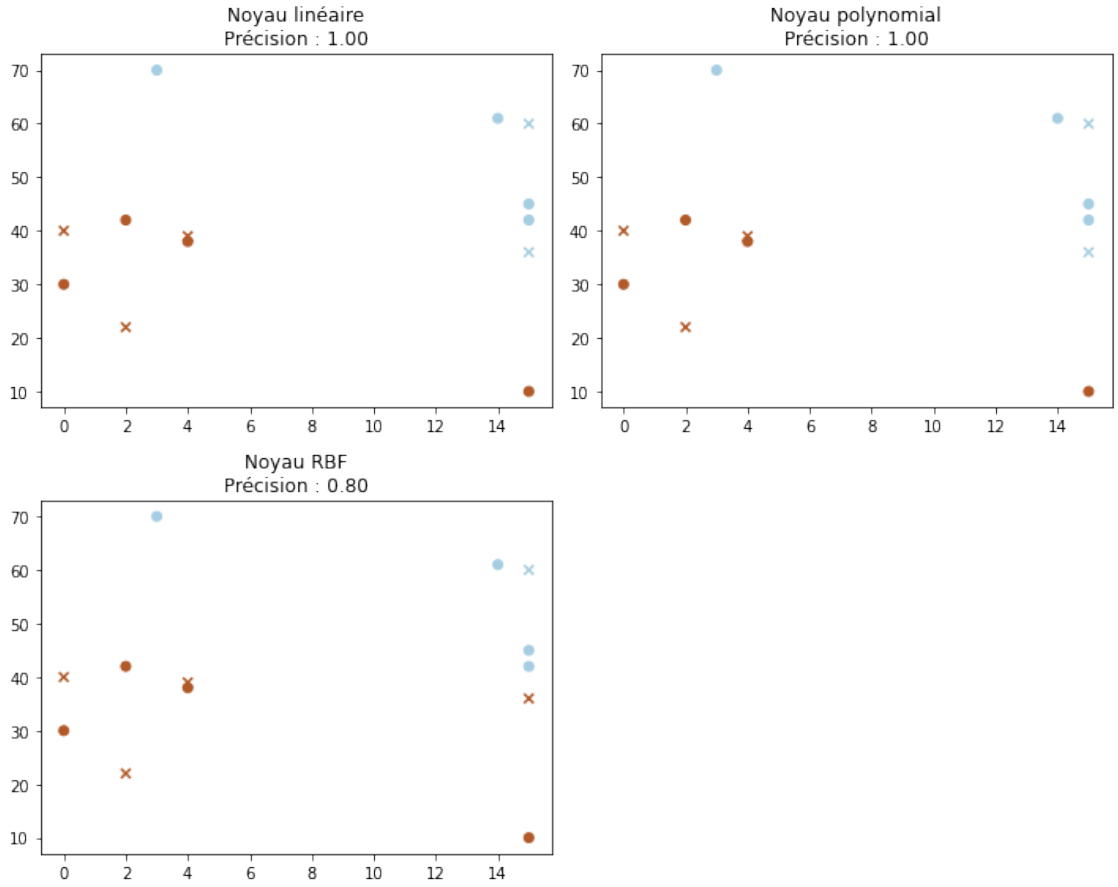
[16]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs

# we create 40 separable points
X, y = make_blobs(n_samples=40, centers=2, random_state=6)

# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel="linear", C=1000)
clf.fit(X, y)

# plot the data points
plt.scatter(X[:, 0], X[:, 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()
```



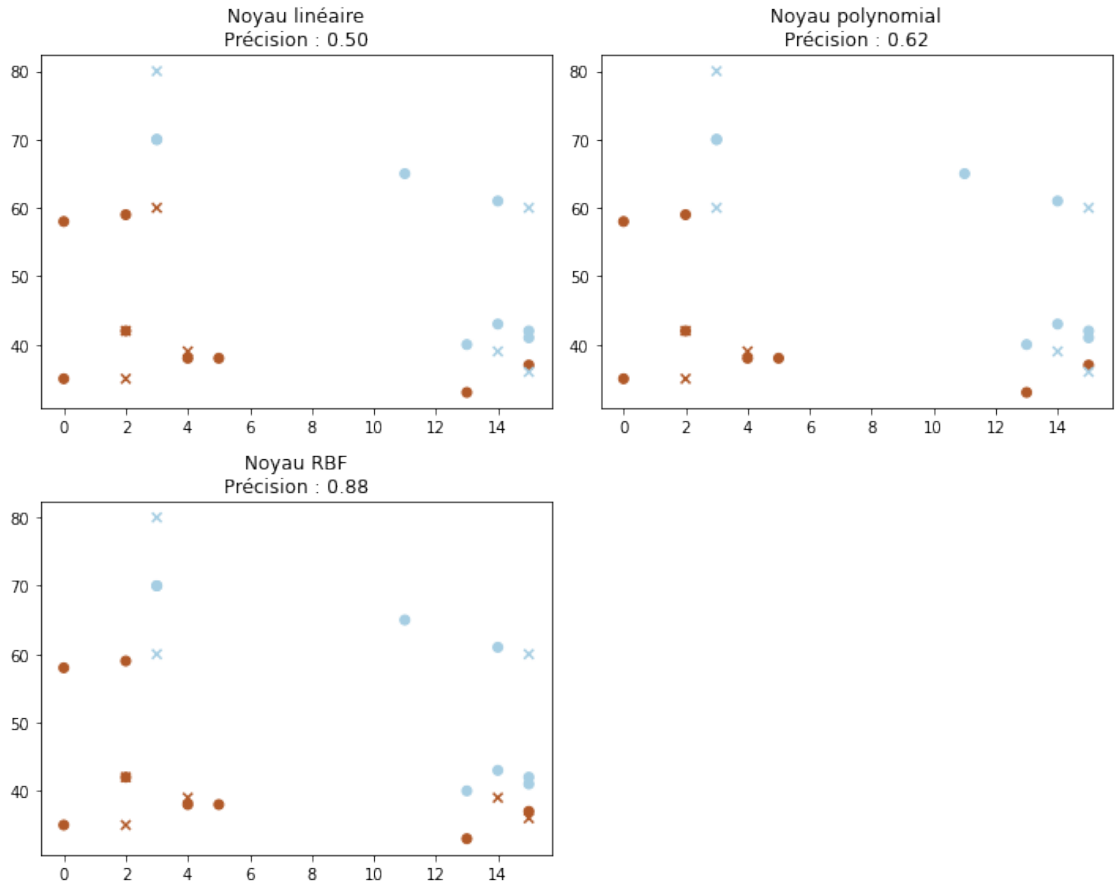
[17]:
```
# SVM avec noyau linéaire
clf_linear = svm.SVC(kernel="linear")
clf_linear.fit(X1, y1)
y_pred_linear = clf_linear.predict(X_test1)

# SVM avec noyau polynomial
clf_poly = svm.SVC(kernel="poly", degree=3)
clf_poly.fit(X1, y1)
y_pred_poly = clf_poly.predict(X_test1)

# SVM avec noyau RBF
clf_rbf = svm.SVC(kernel="rbf")
clf_rbf.fit(X1, y1)
y_pred_rbf = clf_rbf.predict(X_test1)

# Calcul de la précision de chaque modèle
acc_linear = accuracy_score(Lab_reels1, y_pred_linear)
acc_poly = accuracy_score(Lab_reels1, y_pred_poly)
acc_rbf = accuracy_score(Lab_reels1, y_pred_rbf)

# Affichage des résultats
plt.figure(figsize=(10, 8))

# Noyau linéaire
plt.subplot(2, 2, 1)
plt.scatter(np.array(X1)[:, 0], np.array(X1)[:, 1], c=y1, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test1)[:, 0], np.array(X_test1)[:, 1], c=y_pred_linear,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau linéaire\nPrécision : {acc_linear:.2f}")

# Noyau polynomial
plt.subplot(2, 2, 2)
plt.scatter(np.array(X1)[:, 0], np.array(X1)[:, 1], c=y1, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test1)[:, 0], np.array(X_test1)[:, 1], c=y_pred_poly,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau polynomial\nPrécision : {acc_poly:.2f}")

# Noyau RBF
plt.subplot(2, 2, 3)
plt.scatter(np.array(X1)[:, 0], np.array(X1)[:, 1], c=y1, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test1)[:, 0], np.array(X_test1)[:, 1], c=y_pred_rbf,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau RBF\nPrécision : {acc_rbf:.2f}")

plt.tight_layout()
```

```
plt.show()
```



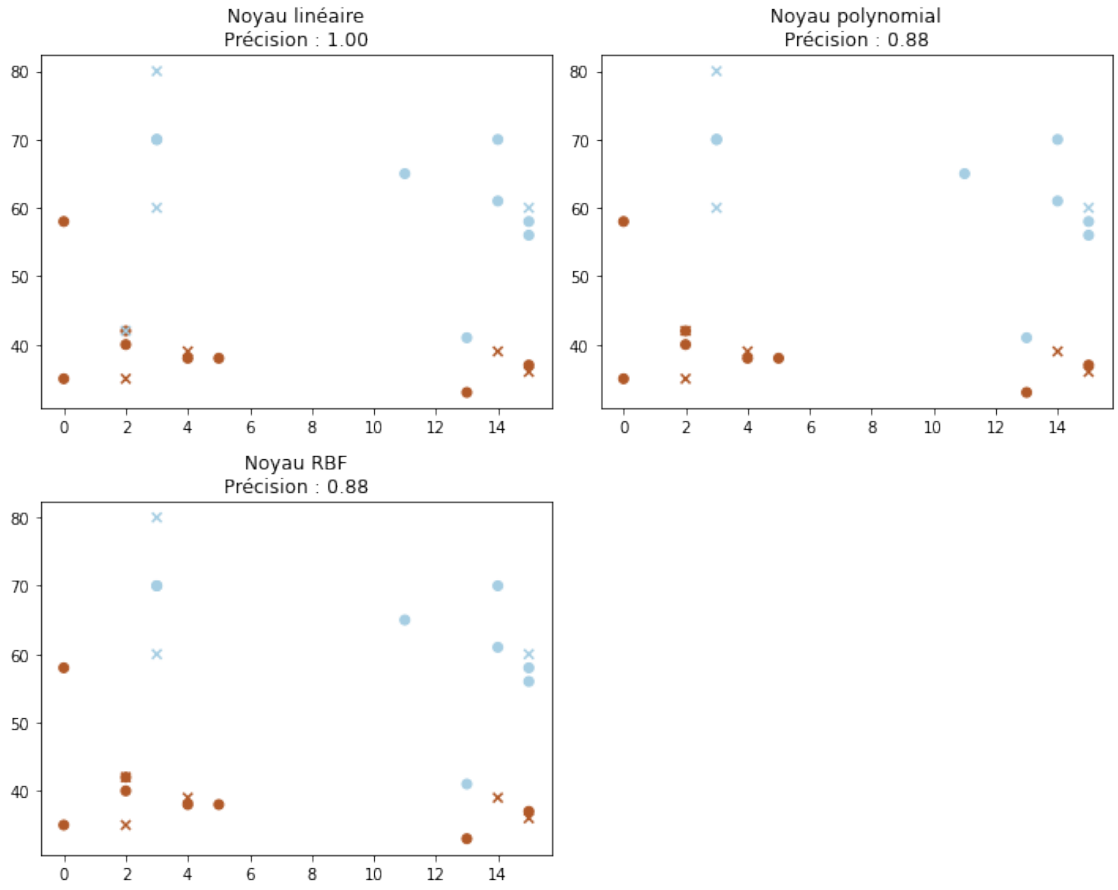
[18]:
```
# SVM avec noyau linéaire
clf_linear = svm.SVC(kernel="linear")
clf_linear.fit(X2, y2)
y_pred_linear = clf_linear.predict(X_test2)

# SVM avec noyau polynomial
clf_poly = svm.SVC(kernel="poly", degree=3)
clf_poly.fit(X2, y2)
y_pred_poly = clf_poly.predict(X_test2)

# SVM avec noyau RBF
clf_rbf = svm.SVC(kernel="rbf")
clf_rbf.fit(X2, y2)
y_pred_rbf = clf_rbf.predict(X_test2)

# Calcul de la précision de chaque modèle
acc_linear = accuracy_score(Lab_reels2, y_pred_linear)
```

```
acc_poly = accuracy_score(Lab_reels2, y_pred_poly)
acc_rbf = accuracy_score(Lab_reels2, y_pred_rbf)

# Affichage des résultats
plt.figure(figsize=(10, 8))

# Noyau linéaire
plt.subplot(2, 2, 1)
plt.scatter(np.array(X2)[:, 0], np.array(X2)[:, 1], c=y2, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test2)[:, 0], np.array(X_test2)[:, 1], c=y_pred_linear,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau linéaire\nPrécision : {acc_linear:.2f}")

# Noyau polynomial
plt.subplot(2, 2, 2)
plt.scatter(np.array(X2)[:, 0], np.array(X2)[:, 1], c=y2, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test2)[:, 0], np.array(X_test2)[:, 1], c=y_pred_poly,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau polynomial\nPrécision : {acc_poly:.2f}")

# Noyau RBF
plt.subplot(2, 2, 3)
plt.scatter(np.array(X2)[:, 0], np.array(X2)[:, 1], c=y2, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test2)[:, 0], np.array(X_test2)[:, 1], c=y_pred_rbf,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau RBF\nPrécision : {acc_rbf:.2f}")

plt.tight_layout()
plt.show()
```



[19]:
```
# SVM avec noyau linéaire
clf_linear = svm.SVC(kernel="linear")
clf_linear.fit(X3, y3)
y_pred_linear = clf_linear.predict(X_test3)

# SVM avec noyau polynomial
clf_poly = svm.SVC(kernel="poly", degree=3)
clf_poly.fit(X3, y3)
y_pred_poly = clf_poly.predict(X_test3)

# SVM avec noyau RBF
clf_rbf = svm.SVC(kernel="rbf")
clf_rbf.fit(X3, y3)
y_pred_rbf = clf_rbf.predict(X_test3)

# Calcul de la précision de chaque modèle
acc_linear = accuracy_score(Lab_reels3, y_pred_linear)
acc_poly = accuracy_score(Lab_reels3, y_pred_poly)
acc_rbf = accuracy_score(Lab_reels3, y_pred_rbf)
```

```
# Affichage des résultats
plt.figure(figsize=(10, 8))

# Noyau linéaire
plt.subplot(2, 2, 1)
plt.scatter(np.array(X3)[:, 0], np.array(X3)[:, 1], c=y3, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test3)[:, 0], np.array(X_test3)[:, 1], c=y_pred_linear,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau linéaire\nPrécision : {acc_linear:.2f}")

# Noyau polynomial
plt.subplot(2, 2, 2)
plt.scatter(np.array(X3)[:, 0], np.array(X3)[:, 1], c=y3, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test3)[:, 0], np.array(X_test3)[:, 1], c=y_pred_poly,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau polynomial\nPrécision : {acc_poly:.2f}")

# Noyau RBF
plt.subplot(2, 2, 3)
plt.scatter(np.array(X3)[:, 0], np.array(X3)[:, 1], c=y3, cmap=plt.cm.Paired)
plt.scatter(np.array(X_test3)[:, 0], np.array(X_test3)[:, 1], c=y_pred_rbf,␣
 ↪marker='x', cmap=plt.cm.Paired)
plt.title(f"Noyau RBF\nPrécision : {acc_rbf:.2f}")

plt.tight_layout()
plt.show()
```



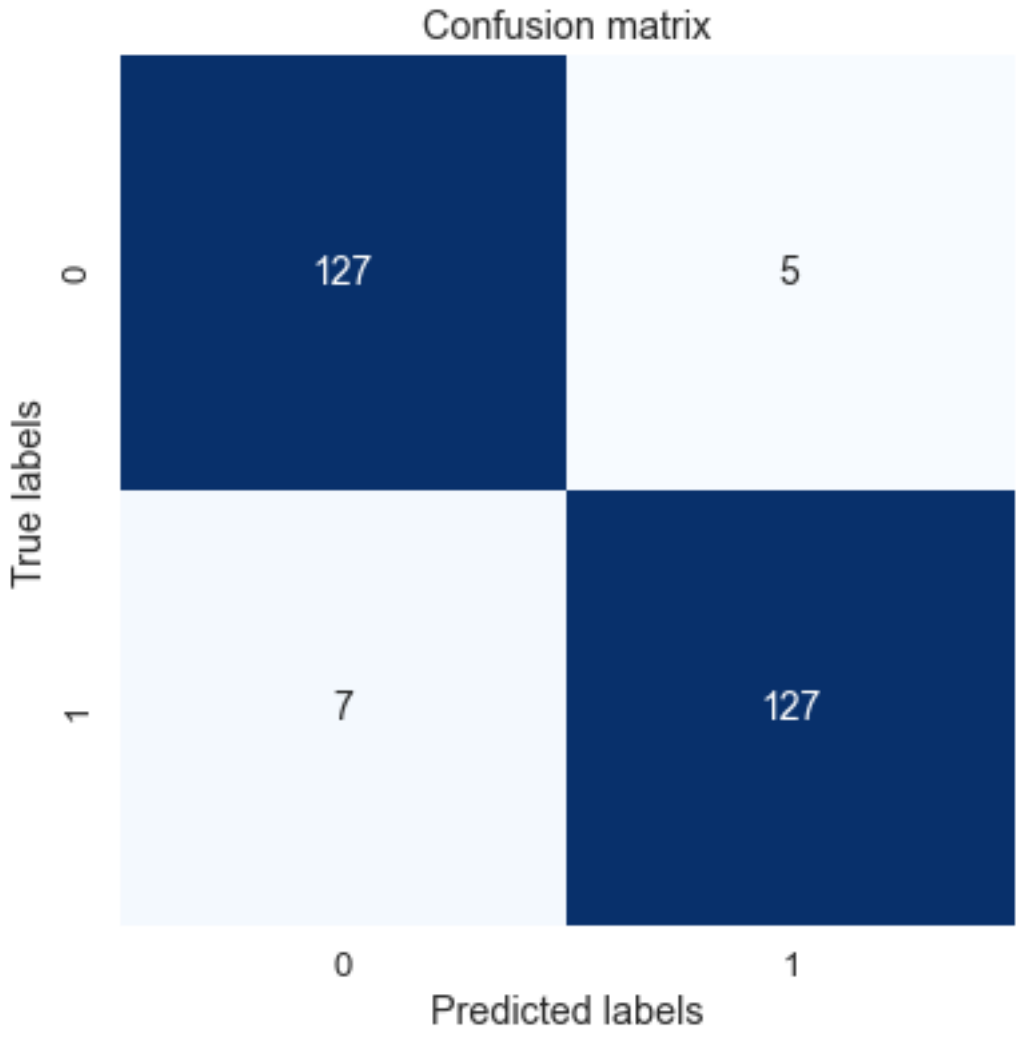
Data Exploitation

[38]:
```
data = pd.read_csv("Donnees_Televigilance_09-10-08.txt", sep=" ", header=None)
data = data.drop([0, 1, 2], axis=1)
display(data)
```

```
     3  4  5  6   7   8
0    0  0  0  0  15  42
1    0  0  0  0  15  41
2    0  0  0  0  15  61
3    0  0  0  0  15  82
4    0  0  0  0  15  81
..  .. .. .. ..  ..  ..
881  0  0  0  0  15  61
882  0  0  0  0  15  78
883  0  0  0  0  15  86
884  0  0  0  0  15  76
885  0  0  0  0  15  81

[886 rows x 6 columns]
```

[56]:
```
X = data[['Appui', 'Appel', 'Colonne6', 'Nuit', 'Pouls']]
Y = data['Cible']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)

model = svm.SVC(kernel='linear')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import confusion_matrix
import seaborn as sns

sns.set(font_scale=1.2)

cm = metrics.confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,␣
 ↪annot_kws={"fontsize":14})
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion matrix')
plt.show()
```

```
Accuracy: 0.9548872180451128
```


Confusion matrix
127
5
7
127
0
1
True labels
0
1
Predicted labels