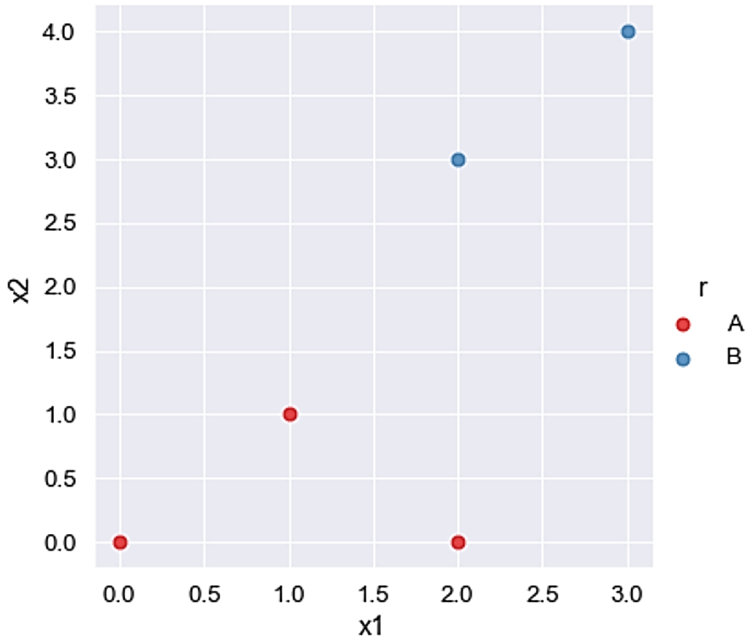
Exemple d'implémentation de SVM en Python avec Scikit-learn

Cette exemple provient du livre : (Lee, Wei-Meng. *Python Machine Learning*. Wiley, 2019)

Un exemple simple permet de mieux comprendre les principes de base des machines à vecteurs de support et leur fonctionnement. Imaginons que nous ayons deux étiquettes : rouge et bleu, et que nos données aient deux caractéristiques : x et y. Nous voulons un classificateur qui, étant donné une paire de coordonnées (x,y), indique si elle est rouge ou bleue. Nous représentons nos données d'entraînement déjà étiquetées sur un plan :

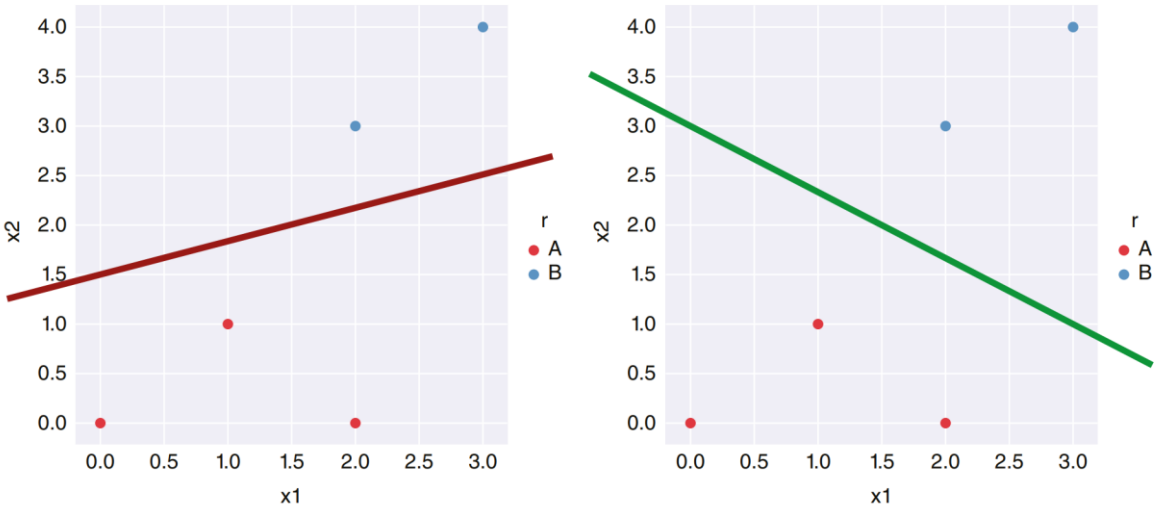


Un ensemble de points qui peuvent être séparés à l'aide d'un SVM.

Une machine à vecteurs de support prend ces points de données et produit une ligne qui sépare le mieux les étiquettes. Cette ligne est **la frontière de décision** : tout ce qui se trouve d'un côté de cette ligne sera classé comme bleu, et tout ce qui se trouve de l'autre côté comme rouge.

Séparation maximale

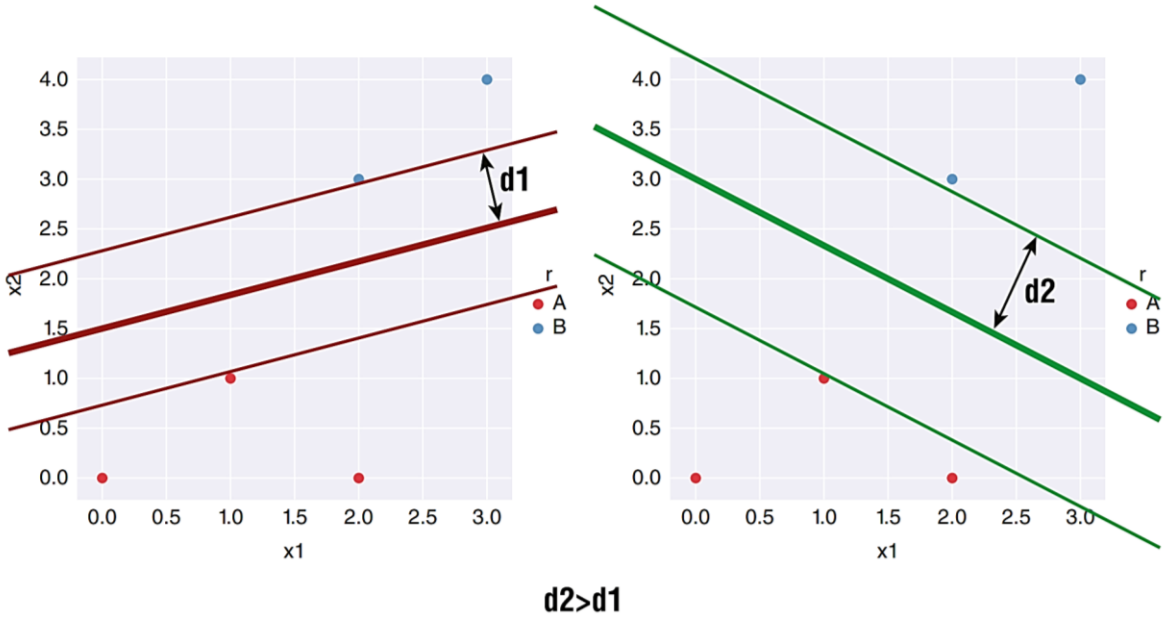
la figure suivante montre deux lignes possibles séparant les deux groupes de points.



Deux manières possibles de diviser les points en deux classes

Bien que les deux lignes séparent les points en deux groupes distincts, pour le SVM, la bonne ligne est celle qui a les marges les plus larges. (avec chaque marge touchant au moins un point dans chaque classe).

Comme le montre la figure suivante. Dans cette figure, d1 et d2 représentent les largeurs des marges. L'objectif est de trouver la plus grande largeur possible pour la marge qui peut séparer les deux groupes. Ainsi, dans ce cas, d2 est la plus grande. La ligne choisie est donc celle de droite.



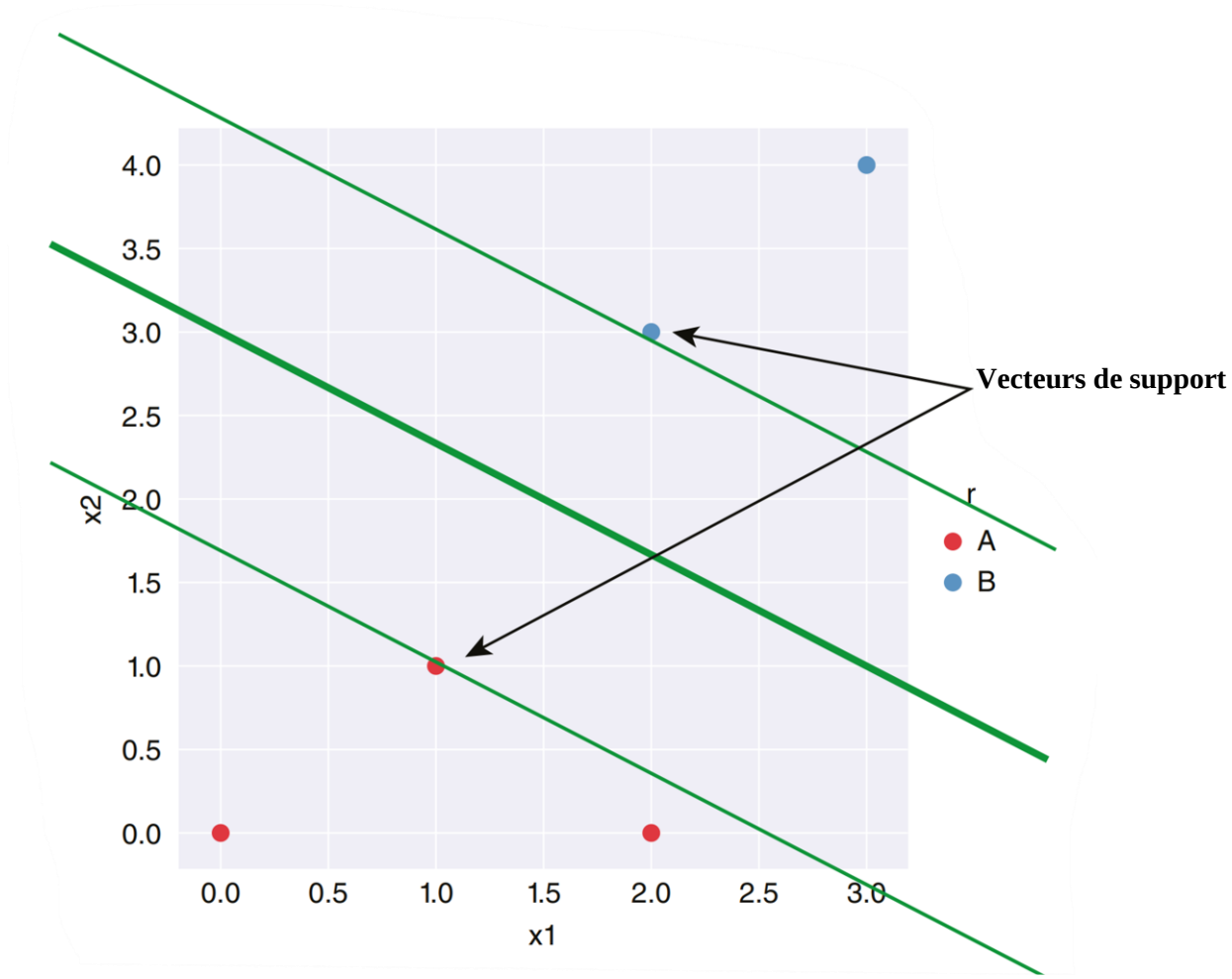
SVM cherche à séparer les deux classes avec la marge la plus large

Chacune des deux marges touche le ou les points les plus proches de chaque groupe de points, et le centre des deux marges est appelé l'hyperplan. L'hyperplan est la ligne qui sépare les deux groupes de points. Nous utilisons le terme "hyperplan" au lieu de "ligne" parce qu'en SVM, les dimensions sont en général supérieures à deux, et l'utilisation du mot "hyperplan" traduit plus précisément l'idée d'un plan dans un espace multidimensionnel.

Vecteurs de support

Un terme essentiel des SVM est celui de vecteurs de support. Les vecteurs de support sont les points qui se trouvent sur les deux marges. En reprenant l'exemple de la section précédente, la figure suivante montre les deux vecteurs de support situés sur les deux marges.

Dans ce cas, nous disons qu'il y a deux vecteurs de support, un pour chaque classe.



Les vecteurs de support sont des points qui se trouvent sur les marges

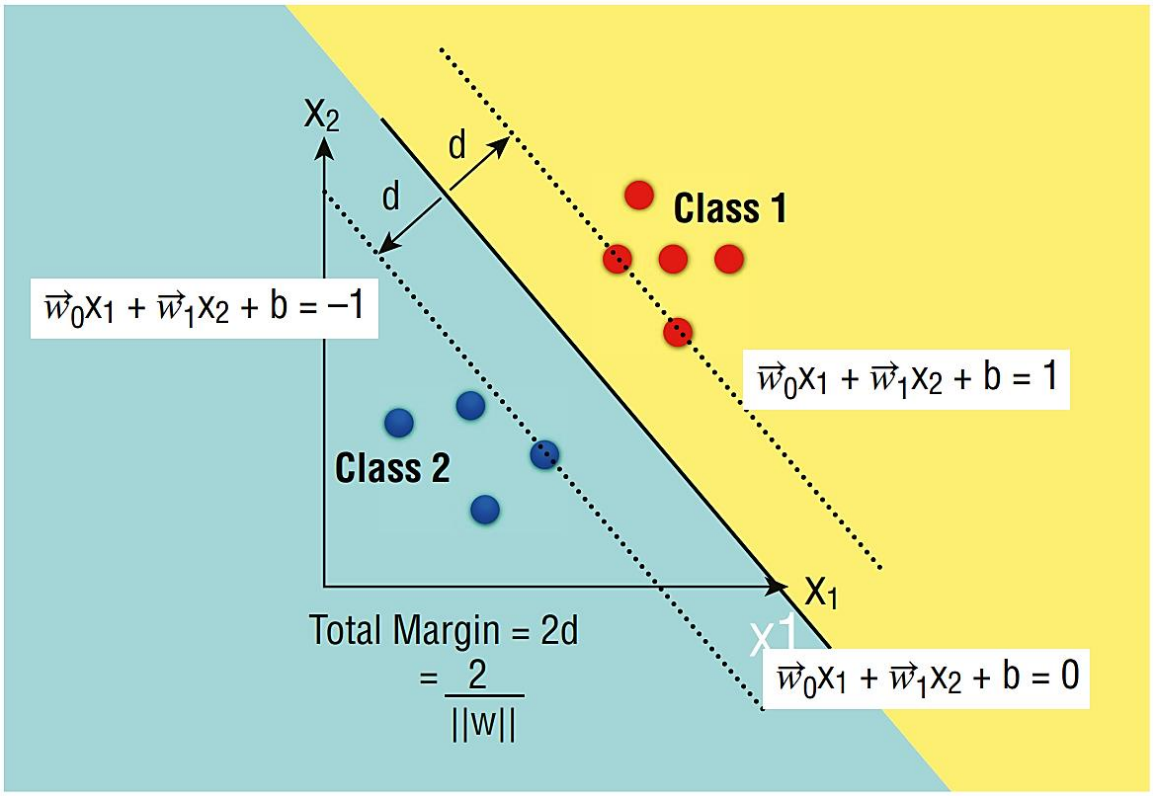
La formule de l'hyperplan

Avec la série de points, la question suivante serait de trouver la formule de l'hyperplan, ainsi que les deux marges. La figure suivante montre la formule permettant d'obtenir l'hyperplan.

Comme vous pouvez le voir sur la figure, la formule de l'hyperplan (g) est donnée comme suit :

$$g(x) = \vec{W}_0 x_1 + \vec{W}_1 x_2 + b$$

où $\mathbf{x_1}$ et $\mathbf{x_2}$ sont les entrées, $\overrightarrow{W_0}$ et $\overrightarrow{W_1}$ sont les vecteurs de poids, et **b** est le biais.



La formule pour l'hyperplan et les deux marges qui l'accompagnent

Si la valeur de g est ≥1, alors le point spécifié est dans la classe 1, et si la valeur de g est ≤ − 1, alors le point spécifié est dans la classe 2. Comme il a été mentionné, le but du SVM est de trouver les marges les plus larges qui divisent les classes, et la marge totale (2d) est définie par : 2 / ‖w‖

où ‖w‖ est le vecteur de poids normalisé ($\overrightarrow{W_0}$ et $\overrightarrow{W_1}$). En utilisant l'ensemble d'apprentissage, l'objectif est de minimiser la valeur de ‖w‖ afin d'obtenir une séparabilité maximale entre les classes. Une fois que cela est fait, vous serez en mesure d'obtenir les valeurs de $\overrightarrow{W_0}$, $\overrightarrow{W_1}$ et b.

La recherche des marges est un problème d'*optimisation sous contraintes*, qui peut être résolu à l'aide de la technique des *multiplicateurs de Lagrange*. Dans un premier temps, nous allons utiliser la bibliothèque Scikit-learn pour trouver la marge en fonction de l'ensemble de données.

L'utilisation de Scikit-learn pour les SVM

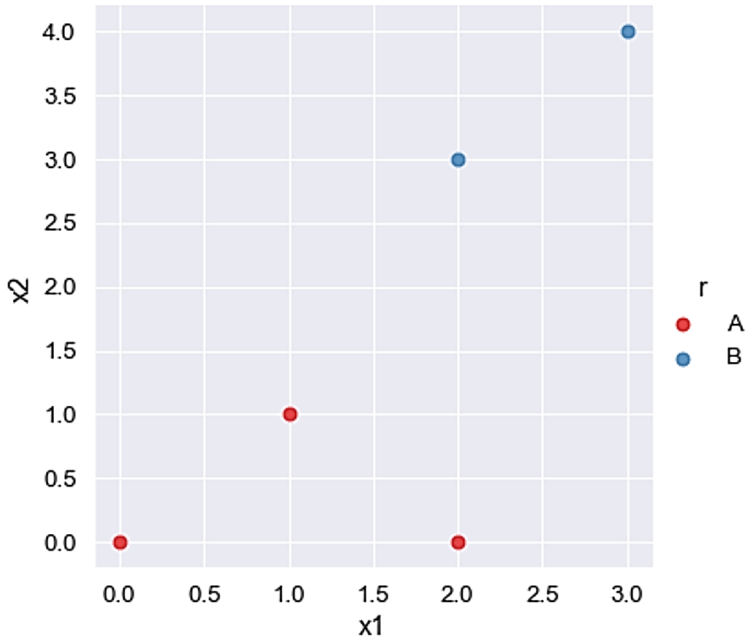
Nous allons maintenant étudier un exemple pour voir comment fonctionne le SVM et comment le mettre en œuvre en utilisant Scikit-learn. Pour cet exemple, nous avons un fichier nommé svm.csv contenant les données suivantes les données suivantes :

```
x1,x2,r
0,0,A
1,1,A
2,3,B
2,0,A
3,4,B
```

La première chose que nous allons faire est de tracer les points en utilisant Seaborn :

```
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns; sns.set(font_scale=1.2)
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('svm.csv')
sns.lmplot('x1', 'x2', data=data, hue='r', palette='Set1', fit_reg=False,
scatter_kws={"s": 50});
```



Traçage des points à l'aide de Seaborn.

À l'aide des points de données que nous avons chargés précédemment, utilisons maintenant la classe SVC du module svm de Scikitlearn pour nous aider à obtenir la valeur des différentes variables que nous devons calculer. L'extrait de code suivant utilise le noyau linéaire pour résoudre le problème. Le noyau linéaire suppose que l'ensemble de données peut être séparé linéairement.

```
from sklearn import svm
#---Converting the Columns as Matrices---
points = data[['x1','x2']].values
result = data['r']
clf = svm.SVC(kernel = 'linear')
clf.fit(points, result)
print('Vector of weights (w) = ',clf.coef_[0])
print('b = ',clf.intercept_[0])
print('Indices of support vectors = ', clf.support_)
print('Support vectors = ', clf.support_vectors_)
print('Number of support vectors for each class = ', clf.n_support_)
print('Coefficients of the support vector in the decision function = ',
np.abs(clf.dual_coef_))
```

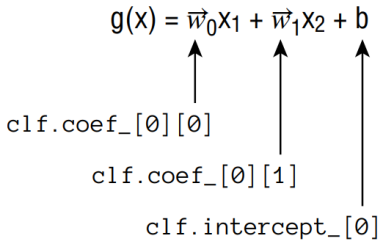
L'extrait de code précédent donne le résultat suivant :

```
Vector of weights (w) = [0.4 0.8]
b = -2.2
Indices of support vectors = [1 2]
Support vectors = [[1. 1.]
[2. 3.]]
Number of support vectors for each class = [1 1]
Coefficients of the support vector in the decision function = [[0.4 0.4]]
```

Comme on peut le voir, le vecteur de poids a été trouvé comme étant [0.4 0.8], ce qui signifie que $\overrightarrow{W_0}$ est maintenant 0.4 et $\overrightarrow{W_1}$ est maintenant 0,8. La valeur de b est -2,2, et il y a deux vecteurs de support. L'indice des vecteurs de support est de 1 et 2, ce qui signifie que les points sont ceux en gras :

```
   x1  x2  r
0  0   0   A
1  1   1   A
2  2   3   B
3  2   0   A
4  3   4   B
```

La figure suivante montre la relation entre les différentes variables de la formule et les variables de l'extrait de code.



La relation entre les variables de la formule et les variables de l'extrait de code.

Traçage de l'hyperplan et des marges

Une fois les valeurs des variables obtenues, il est temps de tracer l'hyperplan et les deux marges qui l'accompagnent. Vous souvenez-vous de la formule pour l'hyperplan ? Elle est la suivante :

$$g(x) = \overrightarrow{W_0}x_1 + \overrightarrow{W_1}x_2 + b$$

Pour tracer l'hyperplan, mettez $\overrightarrow{W_0}x_1 + \overrightarrow{W_1}x_2 + b$ à 0, comme suit :

$$\overrightarrow{W_0}x_1 + \overrightarrow{W_1}x_2 + b = 0$$

Afin de tracer l'hyperplan (qui est une ligne droite dans ce cas), nous avons besoin de deux points : un sur l'axe des x et un sur l'axe des y.

En utilisant la formule précédente, lorsque $x_1 = 0$, nous pouvons résoudre x_2 comme suit :

$$\overrightarrow{W_0}(0) + \overrightarrow{W_1}x_2 + b = 0$$

$$\overrightarrow{W_1}x_2 = -b$$

$$x_2 = -\frac{b}{\overrightarrow{W_1}}$$

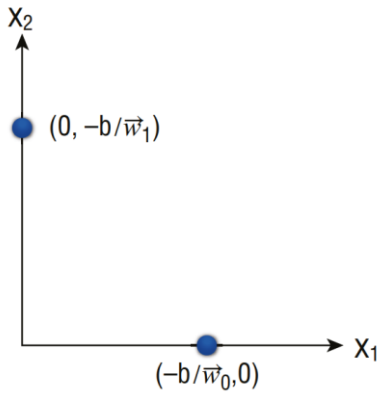
Lorsque $x_2 = 0$, nous pouvons résoudre x_1 comme suit :

$$\overrightarrow{W_0}x_1 + \overrightarrow{W_1}(0) + b = 0$$

$$\overrightarrow{W_0}x_1 = -b$$

$$x_1 = -\frac{b}{\overrightarrow{W_0}}$$

Le point $(0, -b/\overrightarrow{W_1})$ est l'ordonnée à l'origine (*y-intercept*) de la droite. La figure suivante montre les deux points sur les deux axes.



Les deux intercepts de l'hyperplan

Une fois que les points sur chaque axe sont trouvés, vous pouvez maintenant calculer *la pente* (*Slope*) de la façon suivante comme suit :

$$\text{Pente} = (-b/\overrightarrow{W_1}) \,/(b/\overrightarrow{W_0})$$

$$\text{Pente} = -(\overrightarrow{W_0}/\overrightarrow{W_1})$$

Après avoir trouvé la pente et l'ordonnée à l'origine de la ligne, vous pouvez maintenant tracer l'hyperplan. C'est ce que fait l'extrait de code suivant :

```
#---w is the vector of weights---
w = clf.coef_[0]

#---find the slope of the hyperplane---
slope = -w[0] / w[1]

b = clf.intercept_[0]

#---find the coordinates for the hyperplane---
xx = np.linspace(0, 4)
yy = slope * xx - (b / w[1])

#---plot the margins---
s = clf.support_vectors_[0] #---first support vector---
yy_down = slope * xx + (s[1] - slope * s[0])

s = clf.support_vectors_[-1] #---last support vector---
yy_up = slope * xx + (s[1] - slope * s[0])

#---plot the points---
sns.lmplot('x1', 'x2', data=data, hue='r', palette='Set1',
fit_reg=False, scatter_kws={"s": 70})

#---plot the hyperplane---
plt.plot(xx, yy, linewidth=2, color='green');

#---plot the 2 margins---
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
```

Un extrait de code qui affiche l'hyperplan et les deux marges.

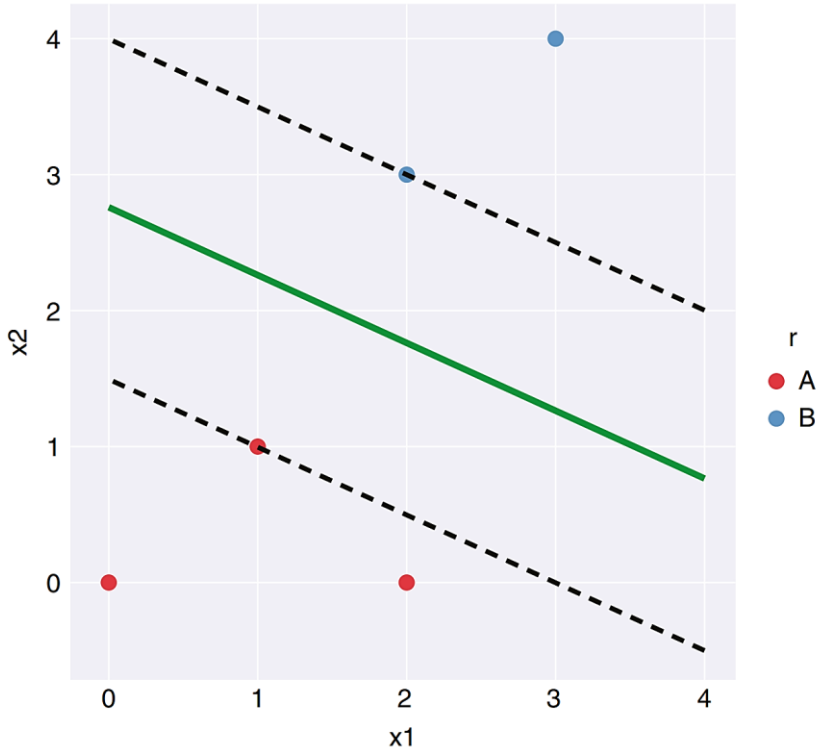
Faire des prédictions

Rappelez-vous que le but du SVM est de séparer les points en deux classes ou plus, afin de pouvoir l'utiliser pour prédire les classes des futurs points. Après avoir entraîné votre modèle à l'aide de SVM, vous pouvez maintenant effectuer des prédictions à l'aide de ce modèle.

L'extrait de code suivant utilise le modèle que vous avez entraîné pour effectuer des quelques prédictions:

```
print(clf.predict([[3,3]])[0]) # 'B'
print(clf.predict([[4,0]])[0]) # 'A'
print(clf.predict([[2,2]])[0]) # 'B'
print(clf.predict([[1,2]])[0]) # 'A'
```

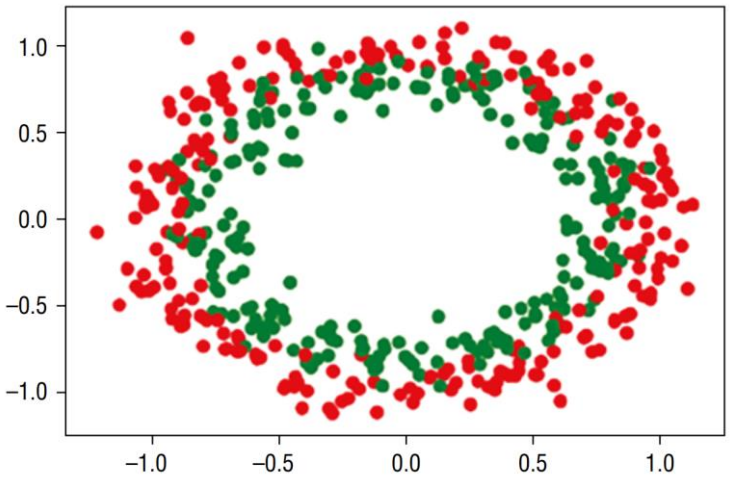
Comparez les points avec le graphe de la figure suivante et voyez si cela a du sens.



L'hyperplan et les deux marges

L'astuce du noyau (Kernel Trick)

Parfois, les points d'un ensemble de données ne sont pas toujours linéairement séparables. Considérons les points illustrés dans la figure suivante.



Un diagramme en nuage de points de deux groupes de points distribués de façon circulaire.

Vous pouvez constater qu'il n'est pas possible de tracer une ligne droite pour séparer les deux ensembles de points. Cependant, avec quelques manipulations, vous pouvez rendre cet ensemble de points linéairement séparables. Cette technique est connue sous le nom de "kernel trick" ou l'astuce du noyau. C'est une technique d'apprentissage automatique qui transforme les données vers un espace de dimension supérieure de sorte que, après la transformation, la marge de séparation entre les classes de données soit claire.

L'ajout d'une troisième dimension

Pour ce faire, nous pouvons ajouter une troisième dimension, disons l'axe z, et définir z comme étant :

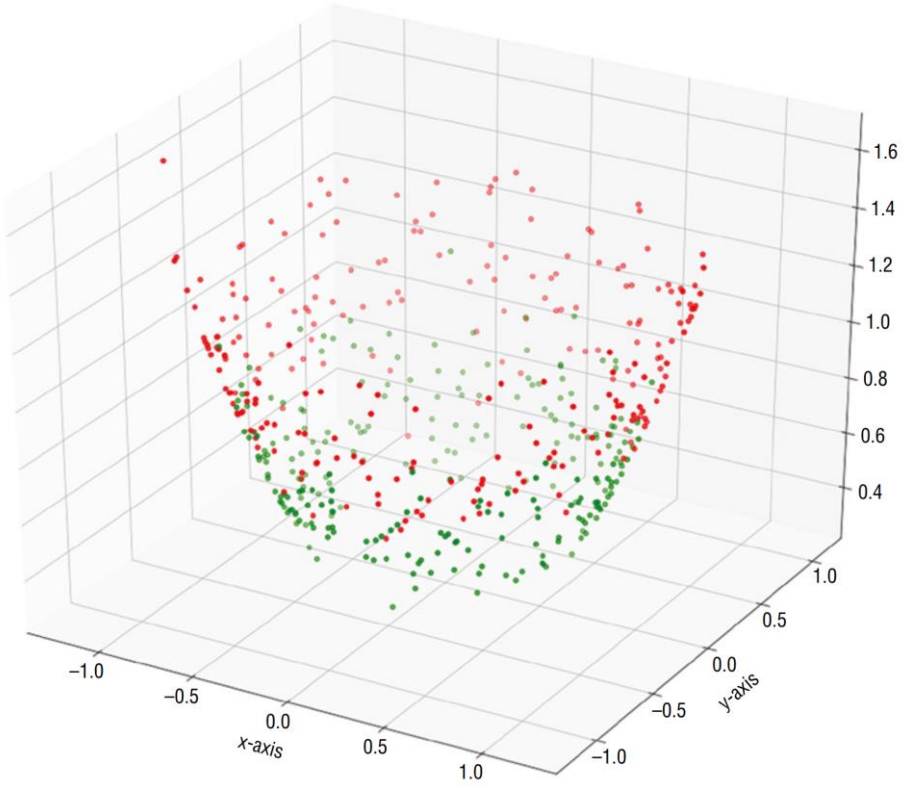
$Z = x^2 + y^2$

Une fois que nous avons tracé les points à l'aide d'un graphique 3D, les points sont maintenant linéairement séparables. Il est difficile de visualiser ce phénomène à moins de tracer les points. C'est ce que fait l'extrait de code suivant :

```
%matplotlib inline

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_circles

#---X is features and c is the class labels---
X, c = make_circles(n_samples=500, noise=0.09)

rgb = np.array(['r', 'g'])
plt.scatter(X[:, 0], X[:, 1], color=rgb[c])
plt.show()

fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
z = X[:,0]**2 + X[:,1]**2
ax.scatter(X[:, 0], X[:, 1], z, color=rgb[c])
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```

Nous commençons par créer deux ensembles de points aléatoires (500 points au total) répartis de façon circulaire en utilisant la fonction *make_circles*(). Nous les traçons ensuite sur un graphique 2D (comme celui de la figure précédente). Nous ajoutons ensuite un troisième axe, l'axe z, et nous traçons le graphique en 3D (voir la figure suivante).

***Remarque* :** *Si vous exécutez le code précédent dans un Terminal (supprimez simplement l'instruction %matplotlib en haut de l'extrait de code ou remplacez-la par : %matplotlib notebook) en utilisant la commande python, vous serez en mesure de faire pivoter et d'interagir avec le graphique.*



Affichage des points en 3 dimension

Traçage de l'hyperplan 3D

Avec les points tracés dans un graphique 3D, entraînons maintenant le modèle en utilisant la troisième dimension:

```
#---combine X (x-axis,y-axis) and z into single ndarray---
features = np.concatenate((X,z.reshape(-1,1)), axis=1)


#---use SVM for training---
from sklearn import svm


clf = svm.SVC(kernel = 'linear')
clf.fit(features, c)
```

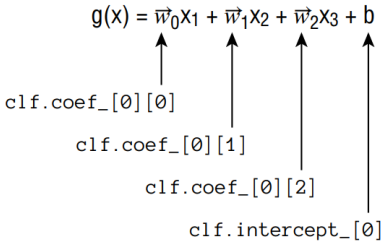
Tout d'abord, nous avons combiné les trois axes en un seul *ndarray* à l'aide de la fonction *np.concatenate()*. Nous avons ensuite entraîné le modèle comme d'habitude. Pour un ensemble de points linéairement séparables en deux dimensions, la formule de l'hyperplan est la suivante :

$$g(x) = \overrightarrow{W_0}x_1 + \overrightarrow{W_1}x_2 + b$$

Pour l'ensemble des points maintenant en trois dimensions, la formule devient maintenant la suivante :

$$g(x) = \overrightarrow{W_0}x_1 + \overrightarrow{W_1}x_2 + \overrightarrow{W_2}x_3 + b$$

En particulier, $\overrightarrow{W_2}$ est maintenant représenté par *clf.coef_*[0][2], comme le montre la figure suivante.



La formule pour l'hyperplan en 3D et ses variables correspondantes dans le extrait de code

L'étape suivante consiste à dessiner l'hyperplan en 3D. Pour ce faire, vous devez trouver la valeur de x_3, qui peut être dérivée, comme indiqué dans la figure suivante.

$$\vec{w}_0x_1 + \vec{w}_1x_2 + \vec{w}_2x_3 + b = 0$$
$$\vec{w}_2x_3 = -\vec{w}_0x_1 - \vec{w}_1x_2 - b$$
$$\frac{x_3 = -\vec{w}_0x_1 - \vec{w}_1x_2 - b}{\vec{w}_2}$$

Formule pour trouver l'hyperplan en 3D

Ceci peut être exprimé en code comme suit :

```
x3 = lambda x,y: (-clf.intercept_[0] - clf.coef_[0][0] * x - clf.coef_[0][1] * y) / clf.coef_[0][2]
```

Pour tracer l'hyperplan en 3D, utilisez la fonction *plot_surface*() :

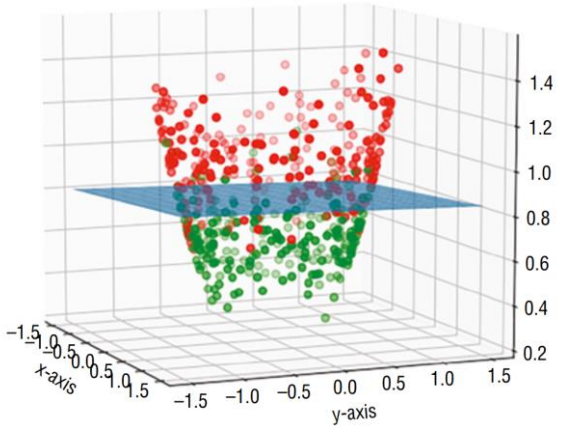
```
tmp = np.linspace(-1.5,1.5,100)
x,y = np.meshgrid(tmp,tmp)


ax.plot_surface(x, y, x3(x,y))
plt.show()
```

Le fragment de code complet est le suivant :

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_circles


#---X is features and c is the class labels---
X, c = make_circles(n_samples=500, noise=0.09)
z = X[:,0]**2 + X[:,1]**2


rgb = np.array(['r', 'g'])


fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], z, color=rgb[c])
plt.xlabel("x-axis")
plt.ylabel("y-axis")
# plt.show()


#---combine X (x-axis,y-axis) and z into single ndarray---
features = np.concatenate((X,z.reshape(-1,1)), axis=1)
```

```
#---use SVM for training---
from sklearn import svm


clf = svm.SVC(kernel = 'linear')
clf.fit(features, c)
x3 = lambda x,y: (-clf.intercept_[0] - clf.coef_[0][0] * x-clf.coef_[0][1]
* y) / clf.coef_[0][2]

tmp = np.linspace(-1.5,1.5,100)
x,y = np.meshgrid(tmp,tmp)


ax.plot_surface(x, y, x3(x,y))
plt.show()
```

La figure suivante présente l'hyperplan, ainsi que les points, tracés en 3D.

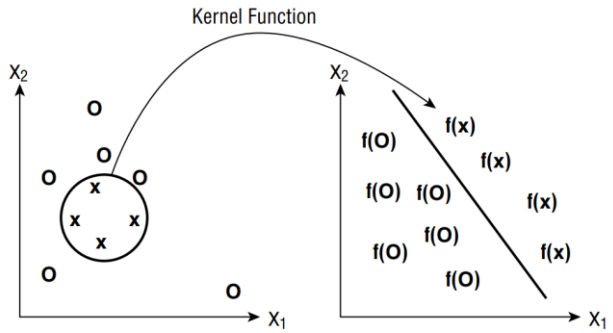


L'hyperplan en 3D qui coupe les deux ensembles de points.

Les types de noyaux

Jusqu'à présent, nous n'avons abordé qu'un seul type de SVM : le SVM linéaire (*linear SVM*). Comme son nom l'indique, le SVM linéaire utilise une ligne droite pour séparer les points. Dans la section précédente, vous avez également appris à utiliser des astuces de noyaux pour séparer deux ensembles de données distribuées de manière circulaire et vous avez ensuite utilisé un SVM linéaire pour les séparer.

Parfois, tous les points ne peuvent pas être séparés de façon linéaire, et ils ne peuvent pas non plus être séparés en utilisant les astuces du noyau que vous avez observées dans la section précédente. Pour ce type de données, vous devez "courber" les lignes pour les séparer. En apprentissage automatique, les noyaux sont des fonctions qui transforment vos données d'espaces non linéaires en espaces linéaires. (voir Figure suivante).



Une fonction noyau transforme les données des espaces non linéaires en espaces linéaires.