13 - La classe de Perceptron de sklearn

Introduction

Dans le chapitre précédent, nous avions implémenté une classe Perceptron simple en utilisant du Python pur. Le module sklearn contient une classe Perceptron. Nous avons vu qu'un perceptron est un algorithme permettant de résoudre des problèmes de classifieurs binaires. Cela signifie qu'un Perceptron est un classificateur binaire, qui peut décider si une entrée appartient ou non à l'une ou l'autre classe. Par exemple, "spam" ou "ham". Pour ce faire, nous combinons linéairement les poids avec le vecteur de caractéristiques, c'est-à-dire l'entrée.

Il est étonnant que l'algorithme du perceptron ait été inventé en 1958 par Frank Rosenblatt. L'algorithme a été mis en œuvre dans un matériel personnalisé, appelé "perceptron Mark 1". Ce matériel a été conçu pour la reconnaissance d'images.

L'invention a été extrêmement surestimée : En 1958, le New York Times écrivait après une conférence de presse avec Rosenblatt : "New Navy Device Learns By Doing ; Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser".

Ce qui semblait initialement très prometteur s'est rapidement révélé incapable de tenir ses promesses. Ces perceptrons n'ont pas pu être entraînés à reconnaître de nombreuses classes de modèles.

Exemple : Classe Perceptron

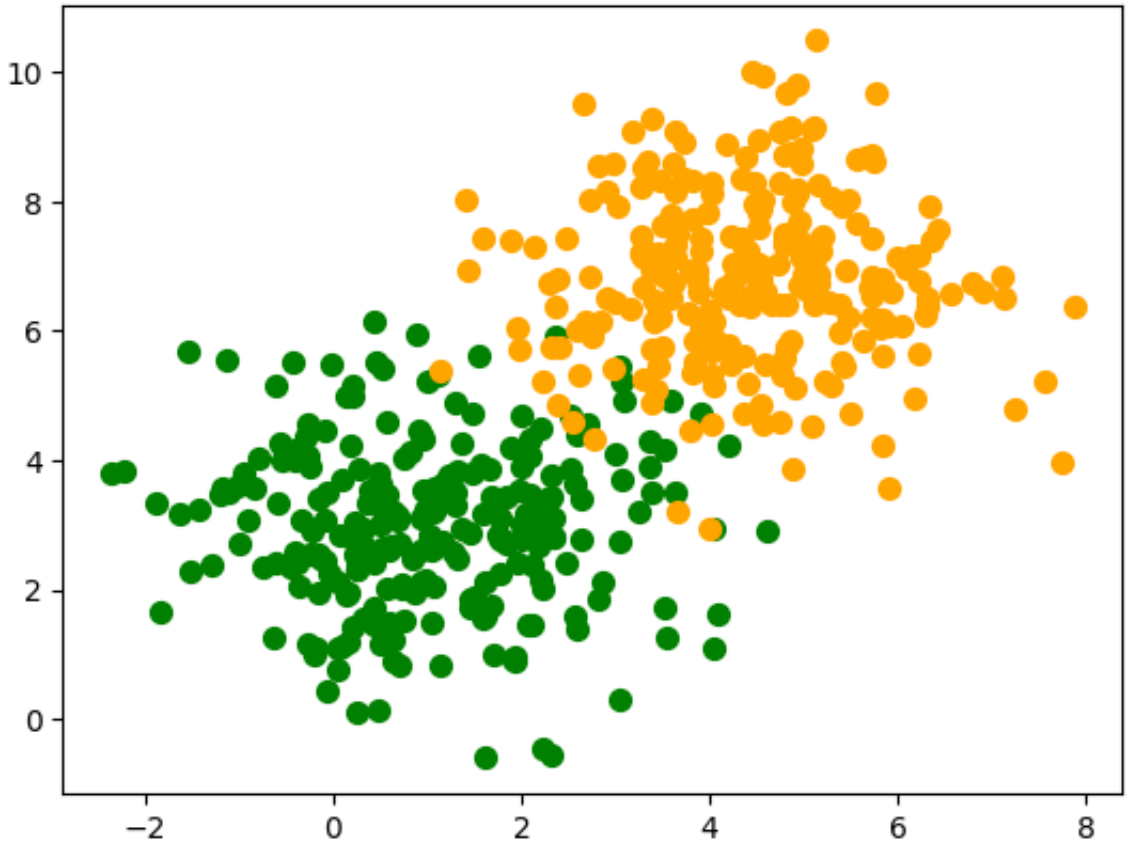
Nous allons créer avec l'aide de `make_blobs` un jeu de tests binaires :

Entrée [1]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

n_samples = 500
data, labels = make_blobs(n_samples=n_samples,
                          centers=([1.1, 3], [4.5, 6.9]),
                          cluster_std=1.3,
                          random_state=0)


colours = ('green', 'orange')
fig, ax = plt.subplots()

for n_class in range(2):
    ax.scatter(data[labels==n_class][:, 0],
               data[labels==n_class][:, 1],
               c=colours[n_class],
               s=50,
               label=str(n_class))
```



Nous allons diviser notre jeu de tests en un jeu d'apprentissage et un jeu de tests :

Entrée [3]:
```
from sklearn.model_selection import train_test_split
datasets = train_test_split(data,
                            labels,
                            test_size=0.2)

train_data, test_data, train_labels, test_labels = datasets
```

Nous n'utiliserons pas la classe Perceptron de sklearn.linear_model :

Entrée [4]:
```
from sklearn.linear_model import Perceptron
p = Perceptron(random_state=42)
p.fit(train_data, train_labels)
```

Out[4]:
```
Perceptron(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Nous pouvons calculer des prédictions sur l'ensemble d'apprentissage et l'ensemble de tests et évaluer le score :

Entrée [5]:
```
from sklearn.metrics import accuracy_score

predictions_train = p.predict(train_data)
predictions_test = p.predict(test_data)
train_score = accuracy_score(predictions_train, train_labels)
print("score on train data: ", train_score)
test_score = accuracy_score(predictions_test, test_labels)
print("score on test data: ", test_score)
```

```
score on train data:  0.975
score on test data:  0.95
```

Entrée [6]:
```
p.score(train_data, train_labels)
```

Out[6]:
```
0.975
```

Classification des données d'iris avec le classificateur Perceptron

Nous voulons appliquer le classificateur Perceptron sur le jeu de données de l'iris, que nous avions déjà utilisé dans notre chapitre sur le k-plus proche voisin.

Chargement du jeu de données de l'iris :

Entrée [7]:
```
import numpy as np
from sklearn.datasets import load_iris


iris = load_iris()
```

Nous avons un problème : le classificateur Perceptron ne peut être utilisé que pour des problèmes de classification binaire, mais l'ensemble de données sur l'iris comprend trois classes différentes, à savoir `setosa` , `versicolor` , `virginica` , correspondant aux étiquettes 0, 1 et 2 :

Entrée [8]:
```
iris.target_names
```

Out[8]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Nous allons fusionner les classes 'versicolor' et 'virginica' en une seule classe. Cela signifie qu'il ne reste que deux classes. Nous pouvons donc différencier avec le classificateur entre

- Iris `setose` ;
- et non Iris `setosa` , ou en d'autres termes soit `viriginica` soit `versicolor` .

Nous accomplissons ceci avec la commande suivante :

Entrée [9]:
```
targets = (iris.target==0).astype(np.int8)
print(targets)
```

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

Nous divisons les données en un ensemble d'apprentissage et un ensemble de tests :

Entrée [10]:
```
from sklearn.model_selection import train_test_split
datasets = train_test_split(iris.data,
                            targets,
                            test_size=0.2)

train_data, test_data, train_labels, test_labels = datasets
```

Maintenant, nous créons une instance de Perceptron et ajustons les données d'entraînement :

Entrée [11]:
```
from sklearn.linear_model import Perceptron
p = Perceptron(random_state=42,
               max_iter=10,
               tol=0.001)
p.fit(train_data, train_labels)
```

Out[11]:
```
Perceptron(max_iter=10, random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Maintenant, nous sommes prêts pour les prédictions et nous allons examiner quelques valeurs X choisies au hasard :

Entrée [12]:
```
import random


sample = random.sample(range(len(train_data)), 10)
for i in sample:
    print(i, p.predict([train_data[i]]))
```

```
98 [0]
30 [1]
92 [1]
72 [1]
99 [0]
59 [0]
81 [0]
97 [1]
69 [1]
15 [1]
```

Entrée [13]:
```
from sklearn.metrics import classification_report

print(classification_report(p.predict(train_data), train_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        83
           1       1.00      1.00      1.00        37

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

Entrée [14]:
```
from sklearn.metrics import classification_report

print(classification_report(p.predict(test_data), test_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      1.00      1.00        13

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Entrée [ ]:
```

```