7. Ensembles d'apprentissage et de test en divisant les données d'apprentissage et de test

1 Données d'apprentissage, de test et d'évaluation

Vous avez vos données prêtes et vous êtes impatient de commencer l'entraînement du classificateur ? Mais attention : Lorsque votre classifieur sera terminé, vous aurez besoin de données de test pour l'évaluer. Si vous évaluez votre classificateur avec les données utilisées pour l'apprentissage, vous pouvez obtenir des résultats étonnamment bons. Ce que nous voulons réellement tester, c'est la performance de la classification sur des données inconnues.

Pour ce faire, nous devons diviser nos données en deux parties :

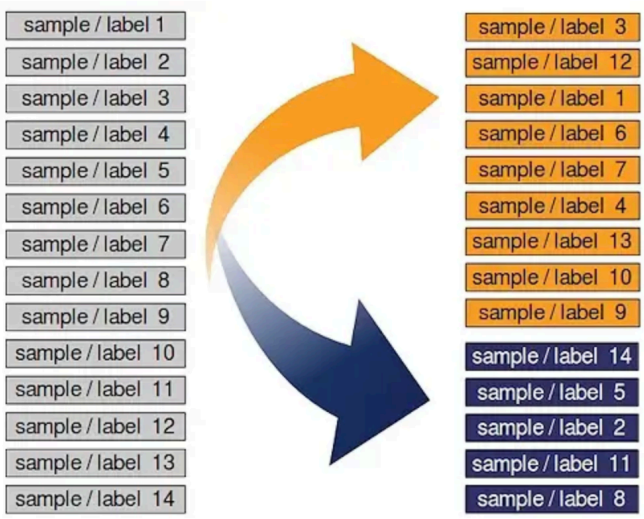
1. Un ensemble d'apprentissage avec lequel l'algorithme d'apprentissage adapte ou apprend le modèle.
2. Un ensemble de test pour évaluer la performance de généralisation du modèle.

Si l'on considère le fonctionnement normal de l'apprentissage automatique, l'idée d'une séparation entre les données d'apprentissage et les données de test est logique. Les systèmes existants s'entraînent sur des données existantes et si de nouvelles données (provenant de clients, de capteurs ou d'autres sources) arrivent, le classificateur entraîné doit prédire ou classer ces nouvelles données. Nous pouvons simuler cela pendant la formation avec un ensemble de données d'apprentissage et de test - les données de test sont une simulation des "futures données" qui entreront dans le système pendant la production.

Dans ce chapitre sur l'apprentissage automatique, nous allons apprendre à effectuer le fractionnement avec Python.

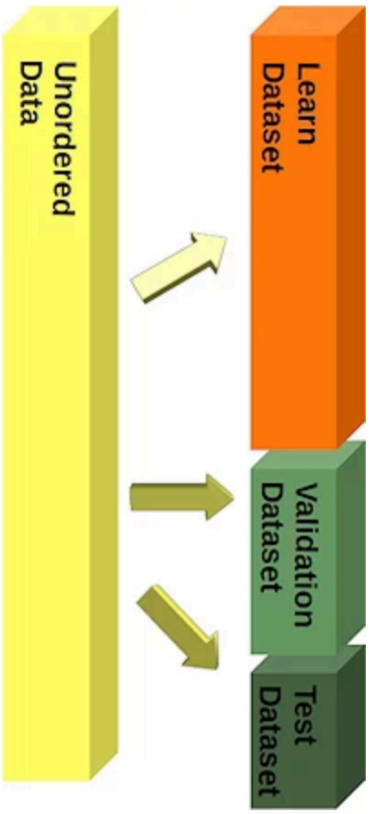
Nous verrons également qu'il n'est pas nécessaire de le faire manuellement, car la fonction train_test_split du module model_selection peut le faire pour nous.

Si l'ensemble de données est trié par étiquette, nous devrons le mélanger avant de le diviser.



Nous avons séparé l'ensemble de données en un ensemble de données d'apprentissage (ou de formation) et un ensemble de données de test. La meilleure pratique consiste à le diviser en un jeu de données d'apprentissage, de test et d'évaluation.

Nous allons entraîner notre modèle (classificateur) étape par étape et chaque fois le résultat doit être testé. Si nous n'avons qu'un jeu de données de test. Les résultats des tests pourraient se retrouver dans le modèle. Nous utiliserons donc un jeu de données d'évaluation pour la phase d'apprentissage complète. Lorsque notre classificateur est terminé, nous le vérifions avec l'ensemble de données de test, qu'il n'a pas "vu" auparavant !



Dans la suite de cette présentation, nous n'utiliserons que les divisions en ensembles de données d'apprentissage et de test.

Exemple de fractionnement : Jeu de données Iris

Nous allons démontrer les sujets abordés précédemment à l'aide du jeu de données Iris.

Les 150 ensembles de données de l'ensemble de données Iris sont triés, c'est-à-dire que les 50 premières données correspondent à la première classe de fleurs (0 = Setosa), les 50 suivantes à la deuxième classe de fleurs (1 = Versicolor) et les données restantes correspondent à la dernière classe (2 = Virginica).

Si nous devions diviser nos données dans le rapport 2/3 (ensemble d'apprentissage) et 1/3 (ensemble de test), l'ensemble d'apprentissage contiendrait toutes les fleurs des deux premières classes et l'ensemble de test toutes les fleurs de la troisième classe de fleurs. Le classificateur ne pourrait apprendre que deux classes et la troisième classe serait complètement inconnue. Il est donc urgent de mélanger les données.

En supposant que tous les échantillons sont indépendants les uns des autres, nous voulons mélanger l'ensemble de données de façon aléatoire avant de le diviser comme indiqué ci-dessus.

Dans ce qui suit, nous divisons les données manuellement :

Entrée [1]:
```
import numpy as np
from sklearn.datasets import load_iris
iris = load_iris()
```

L'examen des étiquettes de iris.target nous montre que les données sont triées.

Entrée [2]:
```
iris.target
```

Out[2]:
```
array([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, 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, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

La première chose à faire est de réorganiser les données de manière à ce qu'elles ne soient plus triées. Pour cela, nous allons utiliser la fonction de permutation du sous-module aléatoire de Numpy :

Entrée [3]:
```
n_test_samples = 12
learnset_data = iris.data[indices[:-n_test_samples]]
learnset_labels = iris.target[indices[:-n_test_samples]]
testset_data = iris.data[indices[-n_test_samples:]]
testset_labels = iris.target[indices[-n_test_samples:]]
print(learnset_data[:4], learnset_labels[:4])
print(testset_data[:4], testset_labels[:4])
```

```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
/var/folders/gp/hcjxzfyd3yj3yq_dyvblymph0000gn/T/ipykernel_65188/2
924322260.py in <module>
      1 n_test_samples = 12
----> 2 learnset_data = iris.data[indices[:-n_test_samples]]
      3 learnset_labels = iris.target[indices[:-n_test_samples]]
      4 testset_data = iris.data[indices[-n_test_samples:]]
      5 testset_labels = iris.target[indices[-n_test_samples:]]

NameError: name 'indices' is not defined
```

Fractionnement avec Sklearn

Même s'il n'était pas difficile de diviser manuellement les données en un ensemble d'apprentissage (train) et un ensemble d'évaluation (test), nous n'avons pas besoin d'effectuer la division manuellement comme indiqué ci-dessus. Comme cela est souvent nécessaire dans l'apprentissage automatique, scikit-learn dispose d'une fonction prédéfinie pour diviser les données en ensembles de formation et de test.

Nous allons en faire la démonstration ci-dessous. Nous utiliserons 80 % des données pour la formation et 20 % pour le test. Nous aurions tout aussi bien pu prendre 70 % et 30 %, car il n'y a pas de règle absolue. Le plus important est que vous évaluez votre système de manière équitable sur la base de données qu'il n'a pas vues pendant l'exercice ! En outre, il doit y avoir suffisamment de données dans les deux ensembles de données.

Entrée [5]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
data, labels = iris.data, iris.target

res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=42)
train_data, test_data, train_labels, test_labels = res

n = 7
print(f"The first {n} data sets:")
print(test_data[:7])
print(f"The corresponding {n} labels:")
print(test_labels[:7])
```

```
The first 7 data sets:
[[6.1 2.8 4.7 1.2]
 [5.7 3.8 1.7 0.3]
 [7.7 2.6 6.9 2.3]
 [6.  2.9 4.5 1.5]
 [6.8 2.8 4.8 1.4]
 [5.4 3.4 1.5 0.4]
 [5.6 2.9 3.6 1.3]]
The corresponding 7 labels:
[1 0 2 1 1 0 1]
```

Echantillon aléatoire stratifié

En particulier avec des quantités de données relativement faibles, il est préférable de stratifier la division. La stratification signifie que nous conservons la proportion de classe originale de l'ensemble de données dans les ensembles de test et de formation. Nous calculons les proportions de classe de la division précédente en pourcentage en utilisant le code suivant. Pour calculer le nombre d'occurrences de chaque classe, nous utilisons la fonction numpy `bincount` . Elle compte le nombre d'occurrences de chaque valeur dans le tableau d'entiers non-négatifs passé en argument.

Entrée [6]:
```
import numpy as np
print('All:', np.bincount(labels) / float(len(labels)) * 100.0)
print('Training:', np.bincount(train_labels) / float(len(train_labe
print('Test:', np.bincount(test_labels) / float(len(test_labels)) *
```

```
All: [33.33333333 33.33333333 33.33333333]
Training: [33.33333333 34.16666667 32.5       ]
Test: [33.33333333 30.         36.66666667]
```

Pour stratifier la division, nous pouvons passer le tableau d'étiquettes comme argument supplémentaire à la fonction train_test_split :

Entrée [7]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
data, labels = iris.data, iris.target

res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=42,
                       stratify=labels)
train_data, test_data, train_labels, test_labels = res

print('All:', np.bincount(labels) / float(len(labels)) * 100.0)
print('Training:', np.bincount(train_labels) / float(len(train_labe
print('Test:', np.bincount(test_labels) / float(len(test_labels)) *
```

```
All: [33.33333333 33.33333333 33.33333333]
Training: [33.33333333 33.33333333 33.33333333]
Test: [33.33333333 33.33333333 33.33333333]
```

C'était un exemple stupide pour tester l'échantillon aléatoire stratifié, car l'ensemble des données Iris a les mêmes proportions, c'est-à-dire que chaque classe a 50 éléments.

Nous allons maintenant travailler avec le fichier `strange_flowers.txt` du répertoire `data` . Ce jeu de données est créé dans le chapitre Générer des jeux de données en Python. Les classes de ce jeu de données ont des nombres d'éléments différents. Tout d'abord, nous chargeons les données :

Entrée [8]:
```
content = np.loadtxt("data/strange_flowers.txt", delimiter=" ")
data = content[:, :-1]    # cut of the target column
labels = content[:, -1]
labels.dtype
labels.shape
```

Out[8]:
```
(795,)
```

Entrée [9]:
```
res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=42,
                       stratify=labels)
train_data, test_data, train_labels, test_labels = res

# np.bincount expects non negative integers:
print('All:', np.bincount(labels.astype(int))  / float(len(labels))
print('Training:', np.bincount(train_labels.astype(int)) / float(le
print('Test:', np.bincount(test_labels.astype(int)) / float(len(tes
```

```
All: [ 0.         23.89937107 25.78616352 28.93081761 21.3836478 ]
Training: [ 0.         23.89937107 25.78616352 28.93081761 21.3836
478 ]
Test: [ 0.         23.89937107 25.78616352 28.93081761 21.3836478
]
```

Entrée [ ]: