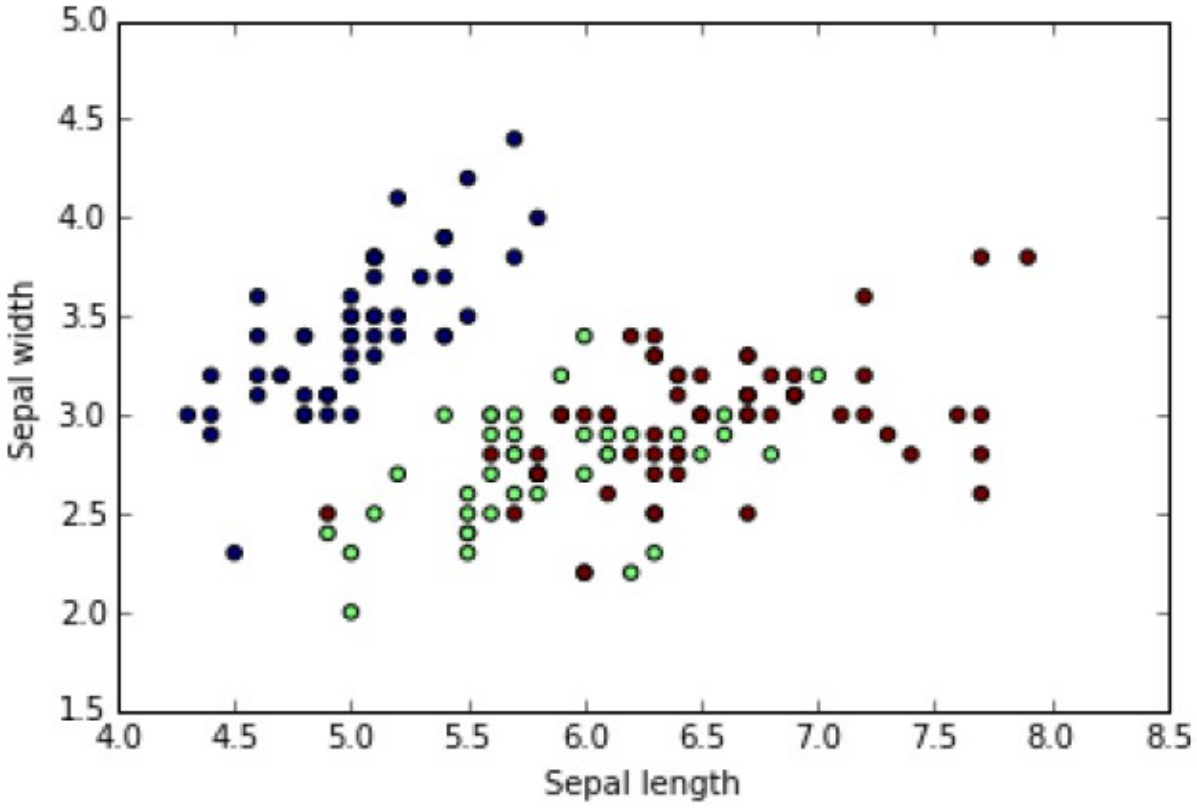
Cargando conjuntos de datos con scikit-learn

Scikit-learn proporciona un montón de métodos para cargar y buscar conjuntos de datos populares, así como generar datos artificiales. Todos estos se pueden encontrar en el paquete **sklearn.datasets**

scikit-learn incorpora algunos pequeños juegos de datos , que proporcionan a los científicos de datos un conjunto de datos para experimentar un nuevo algoritmo y evaluar la corrección de su código antes de aplicarlo a un mundo real de datos de tamaño. Vamos a cargar y renderizar el dataset de iris.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

iris = datasets.load_iris()

# sólo tomamos las 2 primeras features
X = iris.data[:, :2]
Y = iris.target

# Pintamos los puntos de entrenam
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
```

En la figura vemos que mostramos las características correspondientes al ancho y largo del sépalo de las flores



Conjunto de Datos populares

Scikit-learn proporciona cargadores que automáticamente descargan, analizan los archivos de metadatos de varios conjuntos de datos de tamaño real populares:

```
from sklearn.datasets import fetch_20newsgroup

cats = ['alt.atheism', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=cats)
list(newsgroups_train.target_names) # ['alt.atheism', 'sci.space']
newsgroups_train.filenames.shape # (1073,)
newsgroups_train.target.shape # (1073,)
```

MIdata.org repository

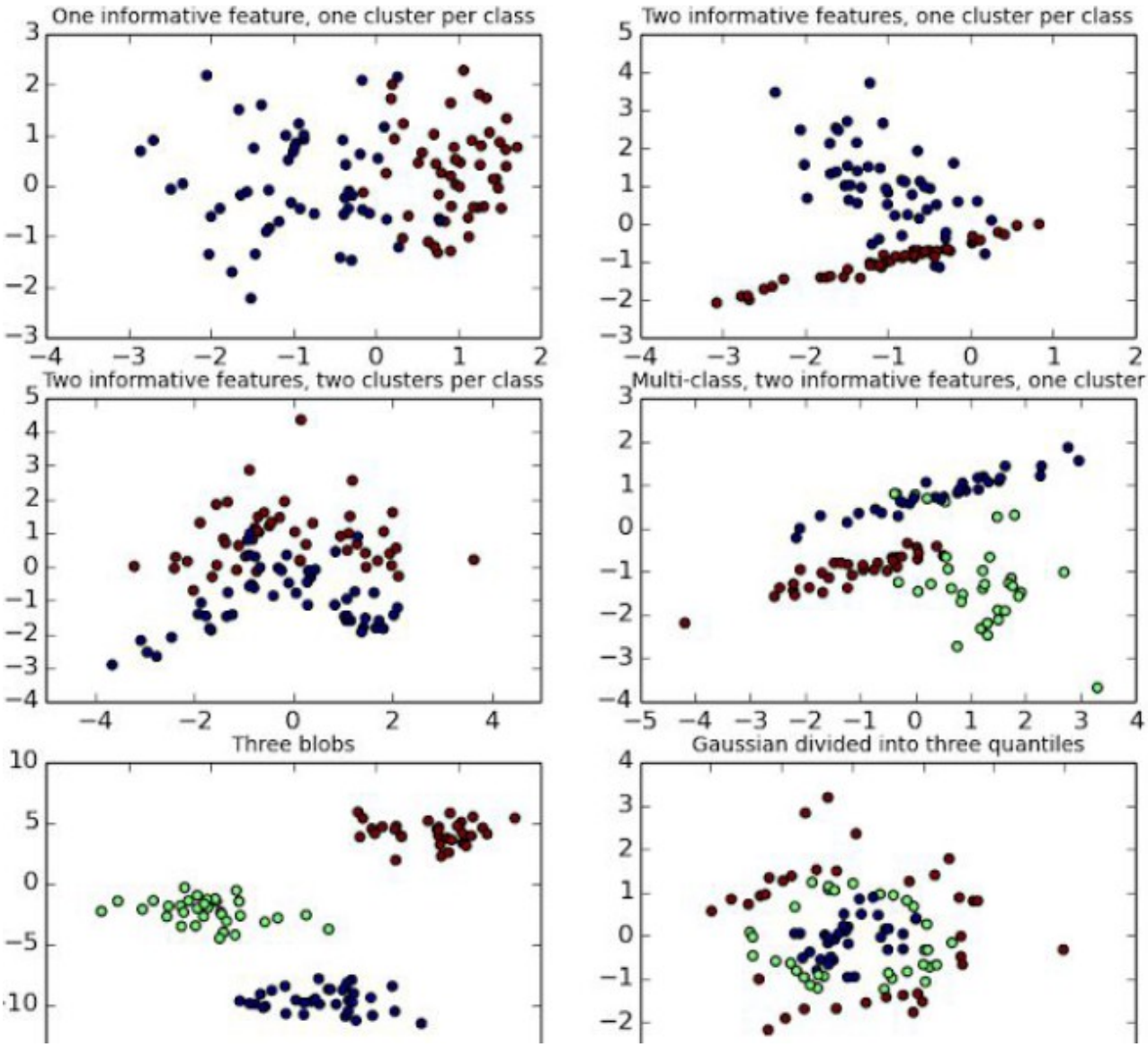
El paquete *sklearn.datasets* puede *descargar directamente conjuntos de* datos desde el repositorio usando la función **fetch_mldata** . Por ejemplo, para descargar la base de datos de reconocimiento de dígitos MNIST, que contiene un total de 70000 ejemplos de dígitos manuscritos de tamaño 28 × 28 píxeles, etiquetados de 0 a 9:

```
from sklearn.datasets import fetch_mldata

mnist = fetch_mldata('MNIST original', data_home=some_path)
mnist.data.shape # (70000, 784)
mnist.target.shape # (70000,)
np.unique(mnist.target) # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Conjuntos de datos generados de forma aleatoria

A veces los conjuntos de datos reales no son suficientes y necesitamos datos que sigan patrones específicos que no pueden lograrse a través de conjuntos de datos reales. **Scikit-learn incluye varios generadores de muestras aleatorias que pueden usarse para construir conjuntos de datos artificiales de tamaño y complejidad controlados** .Esto incluye datos de etiqueta única y múltiple, regresión, clasificaciones, agrupación y más. Vamos a crear varios conjuntos de datos para un problema de clasificación:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.datasets import make_blobs
from sklearn.datasets import make_gaussian_quantiles
plt.figure(figsize=(8, 8))

plt.subplots_adjust(bottom=.05, top=.9, left=.05, right=.95)
plt.subplot(321)
plt.title("One informative feature, one cluster per class", fontsize='small')
X1, Y1 = make_classification(n_features=2, n_redundant=0, n_informative=1,
n_clusters_per_class=1)
plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1)
plt.subplot(322)

plt.title("Two informative features, one cluster per class",
fontsize='small')
X1, Y1 = make_classification(n_features=2, n_redundant=0, n_informative=2,
n_clusters_per_class=1)
plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1)
plt.subplot(323)
plt.title("Two informative features, two clusters per class",
fontsize='small')
X2, Y2 = make_classification(n_features=2, n_redundant=0, n_informative=2)
plt.scatter(X2[:, 0], X2[:, 1], marker='o', c=Y2)

plt.subplot(324)
plt.title("Multi-class, two informative features, one cluster",
fontsize='small')
X1, Y1 = make_classification(n_features=2, n_redundant=0, n_informative=2,
n_clusters_per_class=1, n_classes=3)
plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1)

plt.subplot(325)
plt.title("Three blobs", fontsize='small')
X1, Y1 = make_blobs(n_features=2, centers=3)
plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1)
plt.subplot(326)
plt.title("Gaussian divided into three quantiles", fontsize='small')
X1, Y1 = make_gaussian_quantiles(n_features=2, n_classes=3)

plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1)
plt.show()
```


One informative feature, one cluster per class
Two informative features, one cluster per class
Two informative features, two clusters per class
Multi-class, two informative features, one cluster
Three blobs
Gaussian divided into three quantiles