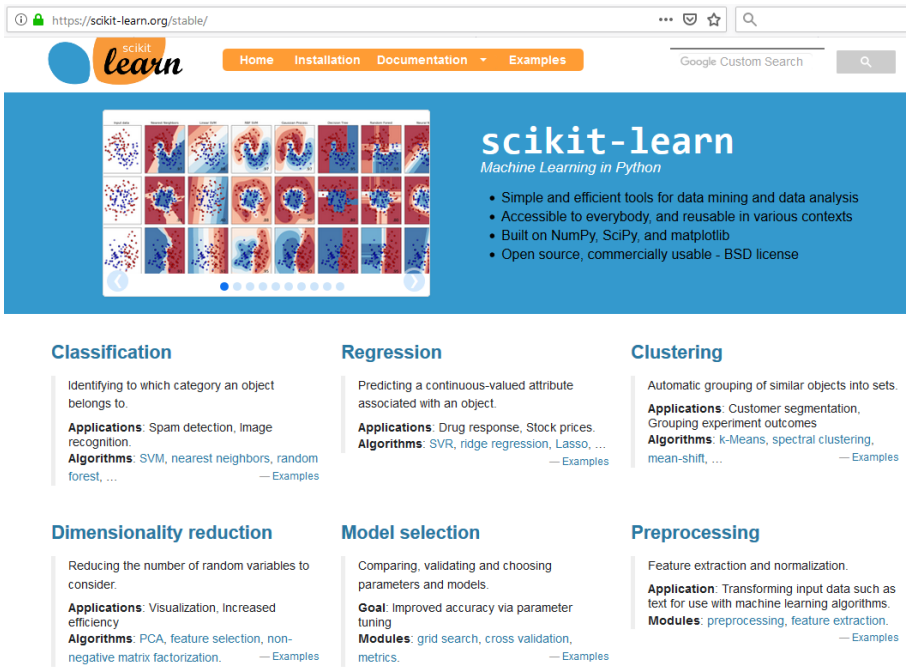
Machine learning con scikit-learn



Introducción

Como punto de partida, mostramos la página web oficial de esta herramienta de *machine learning* en Python.

A partir de un conjunto de n ejemplares de datos, se plantea el problema de predecir el comportamiento de un dato desconocido, típicamente formado por varias características. Además de la regresión, hay dos categorías fundamentales de técnicas para lograrlo: la clasificación supervisada y la clasificación no supervisada o *clustering*. La página oficial mostrada proporciona además otras herramientas relacionadas con el aprendizaje de máquina.

En la clasificación supervisada partimos de un juego de observaciones X cuya categoría conocemos, y tratamos de predecir la categoría de una variable nueva, y, usualmente llamada “objetivo”.

En la casificación no supervisada o clustering, también conocemos un juego de observaciones de partida X, pero su categoría es desconocida para nosotros. El objetivo ahora es formar grupos de ejemplares de categorías similares.

Seguidamente usamos dos juegos de datos. El primero contiene información sobre unas flores llamadas iris, y lo adoptamos para ejemplificar la clasificación supervisada, y concretamente, el algoritmo de los k vecinos más próximos, y la clasificación no supervisada, y concretamente el algoritmo de las k-medias. El segundo juego de datos ofrece información sobre imágenes de dígitos, y nos servirá para ejemplificar el modelo de las máquinas de soporte vectorial, que es otro modelo de clasificación supervisada.

El juego de datos "iris"

En este juego de datos se identifican tres tipos de iris (Setosa, Versicolour, and Virginica) a partir de las longitudes y anchuras de los pétalos y sépalos. El juego de datos se guarda en un array de numpi, de 150×4, donde cada fila es un ejemplar y las columnas los parámetros las longitudes y anchuras de pétalos y sépalos.

La información completa oficial sobre este juego de datos puede encontrarse en la siguiente URL:

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

In [1]:

```
from sklearn import datasets

iris = datasets.load_iris()
iris_X = iris.data
iris_y = iris.target

print(iris_X)
print(iris_y)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

In [2]:

```
# Veamos todos los métodos de la clase iris:

print(dir(iris))

# Y los nombres de los valores objetivo:

print(iris.target_names)
```

```
['DESCR', 'data', 'feature_names', 'filename', 'target', 'target_names']
['setosa' 'versicolor' 'virginica']
```

Para ver gráficamente de qué estamos hablando, mostramos gráficos adecuados. El código que genera estos gráficos se ha tomado de la url que ya hemos mencionado:

https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

In [3]:

```
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(1, figsize=(8, 6))
plt.clf()

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

# To getter a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(2, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```

```
<Figure size 800x600 with 1 Axes>
```

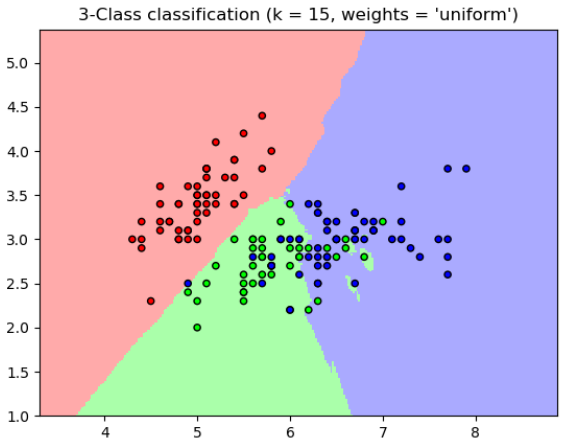
```
<Figure size 800x600 with 1 Axes>
```

El primer gráfico se ha generado tomando las dos primeras características. En el segundo, se ha aplicado un algoritmo de reducción de la dimensionalidad mediante el método de las componentes principales, PCA.

Clasificador de los vecinos más próximos

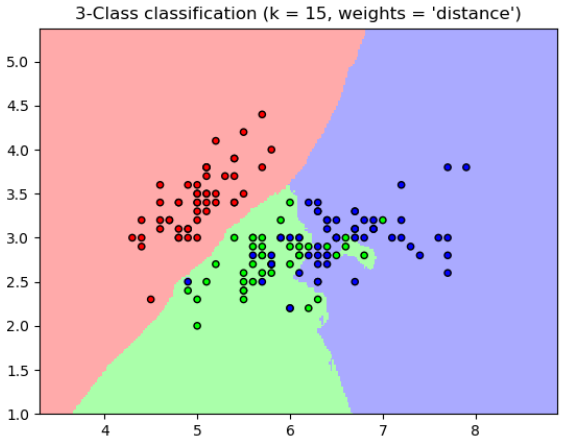
([https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html (https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html)](https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html))

Se trata del clasificador supervisado más sencillo imaginable: se parte de una colección de datos de entrenamiento, cuya categoría se conoce, y el algoritmo las almacena inicialmente. Tras este paso preliminar, el algoritmo toma una observación nueva, cuya categoría no conoce, y decide a qué categoría pertenece examinando los k ejemplares más cercanos.



Actualmente, es la técnica de clasificación supervisada que se usa con más frecuencia.

Una variante alternativa adopta, para la clasificación, los ejemplares más cercanos considerando un radio r prefijado, en lugar de los k vecinos más cercanos. Esta variante es usada más frecuentemente cuando los ejemplares no se distribuyen uniformemente.



Veamos el algoritmo en acción. Para ello, se han tomado todos los ejemplares salvo 10, elegidos aleatoriamente, y con éstos se ajusta el clasificador.

Luego se toman los otros 10 ejemplares, y se les aplica el algoritmo para predecir su especie.

Como los ejemplares usados para predecir están en realidad clasificados, es posible comparar el resultado del algoritmo con la información registrada.

In [4]:

```
# Separamos los datos de iris en datos de entrenamiento y de test
# Una permutación (de los índices) aleatoria,
# para elegir los datos de entrenamiento aleatoriamente:

import numpy as np
np.random.seed(0)
indices = np.random.permutation(len(iris_X))

iris_X_train = iris_X[indices[:-10]]
iris_y_train = iris_y[indices[:-10]]
iris_X_test = iris_X[indices[-10:]]
iris_y_test = iris_y[indices[-10:]]

# Creamos y ajustamos un clasificador de nearest-neighbor:

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(iris_X_train, iris_y_train)

# Y predecimos:

knn.predict(iris_X_test)
```

Out[4]:

```
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
```

In [5]:

```
iris_y_test
```

Out[5]:

```
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

La maldición de la dimensionalidad

Para que un estimador sea efectivo, la distancia entre puntos vecinos debe ser menor que algún valor d, dependiendo del problema, y ese valor típico requiere un número de ejemplares de entrenamiento mayor según se requiera un valor menor de d. En una dimensión, un valor promedio del número de puntos es $n \simeq 1/d$. Pero en dimensión p, $n \simeq 1/d^p$. Por tanto, el número de puntos de entrenamiento crece exponencialmente con respecto a la dimensión.

Clasificación no supervisada: clustering

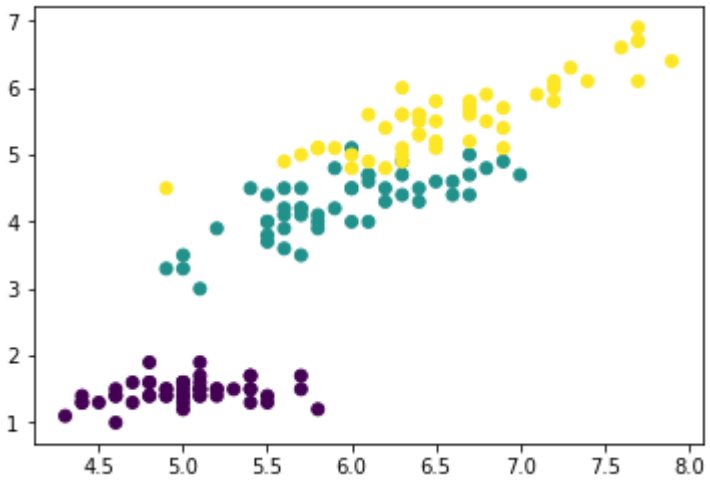
(https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03 (https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03))

In [6]:

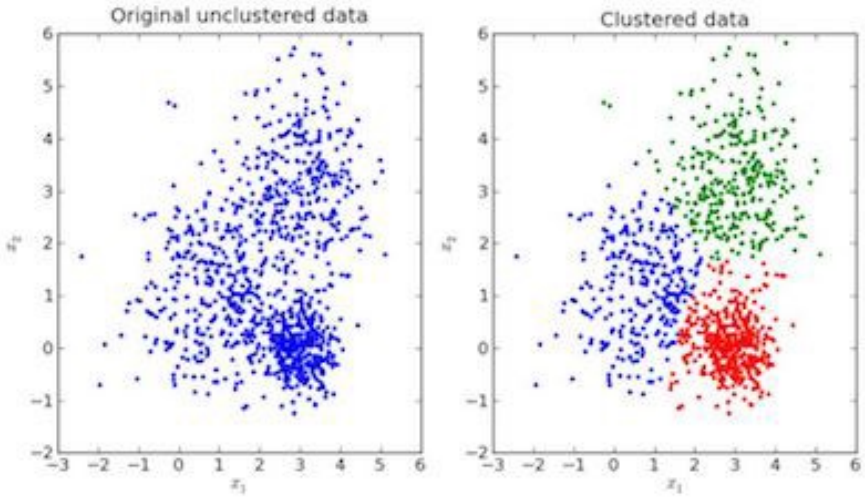
```
import matplotlib.pyplot as plt

# Dataset Slicing
x_axis = iris_X[:, 0]  # Sepal Length
y_axis = iris_X[:, 2]  # Sepal Width

# Plotting
plt.scatter(x_axis, y_axis, c=iris_y)
plt.show()
```



El objetivo del clustering es formar grupos (clusters) de ejemplares con características similares en cada grupo. A manera de ejemplo:



K-medias

Es uno de los múltiples algoritmos de clustering. El número inicial de grupos se decide inicialmente. En nuestro ejemplo es 3. El parámetro “n_clusters” captura ese dato. Luego, se parte de tres datos elegidos aleatoriamente y se asignan a cada uno de los clusters. Los demás datos se asignan a cada cluster inicial según su proximidad a los "centroides", que van siendo recalculados iterativamente. El centroide de cada cluster es una colección de características que representa a cada grupo.

El modelo KMeans está implementado en la librería sklearn:

In [7]:

```
# Importamos el algoritmo de las K-medias...
from sklearn.cluster import KMeans

# ... y lo ajustamos para repartir en 3 grupos
model = KMeans(n_clusters=3)

# Formamos el modelo con nuestros datos:
model.fit(iris_X)

# Predecimos un ejemplar nuevo cualquiera:
predicted_label = model.predict([[7.2, 3.5, 0.8, 1.6]])
print(predicted_label)

# Predecimos todos los ejemplares:
all_predictions = model.predict(iris_X)
print(all_predictions)
```

```
[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 2 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 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

El juego de datos "dígitos"

En este juego de datos consiste en imágenes de dígitos. La tarea es clasificar una imagen nueva según a qué dígito se parece más.

In [8]:

```
digits = datasets.load_digits()

print(digits.data)
print(digits.target)
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
[0 1 2 ... 8 9 8]
```

Cada juego de datos tiene una imagen formada por una matriz de 8×8:

In [9]:

```
digits.images[0]
```

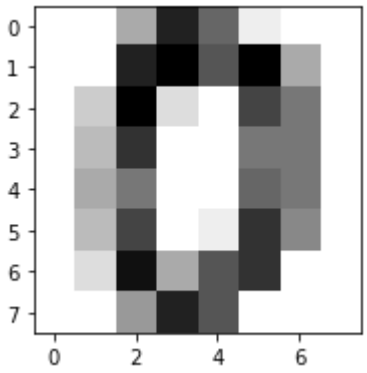
Out[9]:

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

In [10]:

```
# Veamos qué aspecto tiene esta imagen:
import matplotlib.pyplot as plt

# Mostramos la imagen del primer dígito:
plt.figure(1, figsize=(3, 3))
plt.imshow(digits.images[0], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```



Otro clasificador: máquinas de soporte vectorial

(https://scikit-learn.org/stable/tutorial/basic/tutorial.html (https://scikit-learn.org/stable/tutorial/basic/tutorial.html)

Las máquinas de soporte vectorial, máquinas de vectores de soporte o máquinas de vector soporte (Support Vector Machines, SVMs) son un conjunto de algoritmos de aprendizaje supervisado que construyen un modelo capaz de predecir si un punto nuevo (cuya categoría desconocemos) la categoría a la que pertenece un punto nuevo.

La clase sklearn.svm.SVC implementa uno la clasificación vectorial. Sus argumentos son los parámetros del modelo subyacente...

In [11]:

```
from sklearn import svm

# En este ejemplo ponemos los parámetros de la f. gamma manualmente:
clf = svm.SVC(gamma=0.001, C=100.)

# Elegimos seguidamente todos los ejemplares
# de entrenamiento menos el último:

clf.fit(digits.data[:-1], digits.target[:-1])

# El resultado es un array nuevo que contiene todos los ejemplares
# menos el último, de digits.data:
```

Out[11]:

```
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [12]:

```
# Ahora podemos predecir nuevos valores. En este caso, predecimos la última imagen de digit
# Esta predicción determina la imagen del conjunto de imágenes de entrenamiento que mejor e

clf.predict(digits.data[-1:])
```

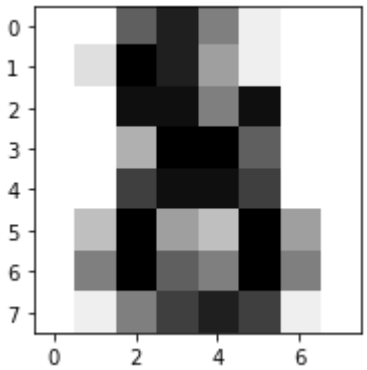
Out[12]:

```
array([8])
```

In [13]:

```
# Podemos verla así:

plt.figure(1, figsize=(3, 3))
plt.imshow(digits.images[-1], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```



Referencias

- https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html
- https://scikit-learn.org/stable/modules/neighbors.html#classification