Implementación de KNeighborsClassifier en scikit-learn

Un paso más hacia lo que sería machine learning es el método *k* -nn de clasificación supervisada. En este caso, a la hora de clasificar un nuevo elemento, buscamos en el conjunto de datos de que disponemos al punto, o *k* -puntos más cercanos, y le asignamos su categoría.

Esta técnica de clasificación tiene un fundamento geométrico,puesto que su funcionamiento se basa en determinar la clase de una instancia simplemente observando la clase de las k instancias que están más próximas a ella en un espacio multidimensional,con tantas dimensiones como atributos tienen las instancias. Este procedimiento requiere de la elección de una medida de distancia ,que es la que se encarga de decidir cuáles son los vecinos más cercanos.Algunas medidas de distancia típicas son la euclídea,,manhattan,coseno..

KneighborsClassifier

```
from sklearn.neighbors import KneighborsClassifier

clf = KneighborsClassifier(n_neighbors=3)
```

Ahora, ajustamos el clasificador usando el conjunto de entrenamiento. Para KNeighborsClassifier esto significa almacenar el conjunto de datos, para que podamos calcular vecinos durante la predicción.

Realizar predicciones

```
clf.fit(X_train, y_train)
```

Para hacer predicciones sobre los datos de prueba, llamamos al método predict.
Para cada punto de datos en el conjunto de pruebas, este calcula sus vecinos más cercanos en el conjunto de entrenamiento y encuentra la clase más común entre estos

Para medir o evaluar el porcentaje de aciertos de los datos de entrenamiento,tenemos la **función score** que mide el accuracy que el porcentaje de aciertos sobre el total de clasificaciones

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KneighborsClassifier

# Importamos el dataset
iris = pd.read_csv('iris.csv')

# Tomamos el ancho y longitud del pétalo
X = iris[['PetalLenght', 'PetalWidth']]

# Para crear luego los gráficos vamos a necesitar categorizar los nombres
# y asignarles un número a cada variedad de planta; en este caso tres.
y = iris['Name'].astype('category')
y.cat.categories = [0, 1, 2]

#Definimos un número de vecinos relativamente grande, k = 15
n_neighbors = 15

# Creamos los colormap
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

for weights in ['uniform', 'distance']:
# Creamos una instancia de Neighbors Classifier y hacemos un fit a partir de los
# datos.
# Los pesos (weights) determinarán en qué proporción participa cada punto en la
# asignación del espacio. De manera uniforme o proporcional a la distancia.
clf = KNeighborsClassifier(n_neighbors, weights=weights)
clf.fit(X, y)

# Creamos una gráfica con las zonas asignadas a cada categoría según el modelo
# k-nearest neighborgs. Para ello empleamos el meshgrid de Numpy.
# A cada punto del grid o malla le asignamos una categoría según el modelo knn.
# La función c_() de Numpy, concatena columnas.
h = .02
x_min, x_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
y_min, y_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Ponemos el resultado en un gráfico.
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

```
# Representamos también los datos de entrenamiento.
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = %i, weights = '%s')"
% (n_neighbors, weights))
plt.xlabel('Petal Width')
plt.ylabel('Petal Length')
plt.savefig('iris-knn-{}'.format(weights))
```

Como podemos ver, en este caso, las líneas que separan las tres categorías de planta Iris ya no son verticales. El modelo k Nearest Neighborgs que hemos empleado ha considerado que la separación de categorías (para selección de ordenadas y abscisas) es más bien tirando a horizontal. Visualmente también podemos apreciar que éste modelo incluye más puntos dentro de la categoría correcta que el modelo simple que hemos estudiado en primer lugar. Si recurrimos a la función score(), obtendremos una puntuación del 96% en el caso de weights='uniform', y del 98.6% si optamos por weights='distancia'. Cabe recordar que en este caso tampoco se ha recurrido a un cross validation para puntuar el modelo, por lo que esta puntuación puede ser algo optimista.