Libro algoritmo kNN_k5

June 13, 2023

1 k vecinos cercanos (kNN)

1.1 Paquetes numpy y pandas

[2]:
```
import numpy as np
import pandas as pd
```

1.1.1 Para fines de estética en la salida, se desactivan las advertencias que pueda informar el intérprete Python

[3]:
```
import warnings
warnings.filterwarnings("ignore")
```

1.2 Importación método para creación del conjunto de entrenamiento desde paquete *sklearn*

[4]:
```
from sklearn.model_selection import train_test_split
```

1.3 Paquete sklearn que contiene los métodos para kNN

[5]:
```
# Método para k vecinos cercanos desde sklearn
from sklearn.neighbors import KNeighborsClassifier
```

1.4 Lectura de los datos desde el archivo *datosAB.txt*

[7]:
```
# Esta es la opción para Jupyter Lab/Notebook en desktop
datos = pd.read_table("datosAB.txt", sep='\t')
```

1.5 Creación de conjunto de datos

[8]:
```
# Conjunto de datos
X = datos.iloc[:,:-1]
y = datos.iloc[:,2]
```

1.6 Creación de subconjutos CP y CE

[9]:
```
# Se elige una semilla para la selección pseudo-aleatoria
semilla = 123456
```

[10]:
```
X_ce, X_cp, y_ce, y_cp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=semilla)
```

1.7 Creación y ajuste del clasificador con k=5

[11]:
```
# Entrenamiento y ajuste
clasificador = KNeighborsClassifier()
clasificador = clasificador.fit(X_ce, y_ce)
```

1.8 Predicción

[14]:
```
# Predicción del conjunto de prueba
y_pred = clasificador.predict(X_cp)
```

```
['naranja' 'naranja' 'rojo' 'naranja' 'naranja' 'naranja' 'rojo' 'rojo'
 'rojo' 'naranja' 'naranja']
```

[21]:
```
# Probabilidad de pertenencia a la clase rojo o naranja
y_prob = clasificador.predict_proba(X_cp)
```

[41]:
```
# Impresión de las probabilidades
from prettytable import PrettyTable
probs = PrettyTable()

probs.add_column("Predicción", y_pred)
probs.add_column("Prob Naranja", y_prob[:,0])
probs.add_column("Prob Rojo", y_prob[:,1])
print(probs)
```

```
+------------+--------------+-----------+
| Predicción | Prob Naranja | Prob Rojo |
+------------+--------------+-----------+
|  naranja   |     0.8      |    0.2    |
|  naranja   |     1.0      |    0.0    |
|    rojo    |     0.0      |    1.0    |
|  naranja   |     0.8      |    0.2    |
|  naranja   |     0.6      |    0.4    |
|  naranja   |     0.8      |    0.2    |
|    rojo    |     0.0      |    1.0    |
|    rojo    |     0.4      |    0.6    |
|    rojo    |     0.4      |    0.6    |
|  naranja   |     0.8      |    0.2    |
|  naranja   |     1.0      |    0.0    |
+------------+--------------+-----------+
```

1.9 Creación de los resultados estadísticos de la clasificación

1.9.1 Importación de método para la matriz de confusión desde paquete *sklearn*

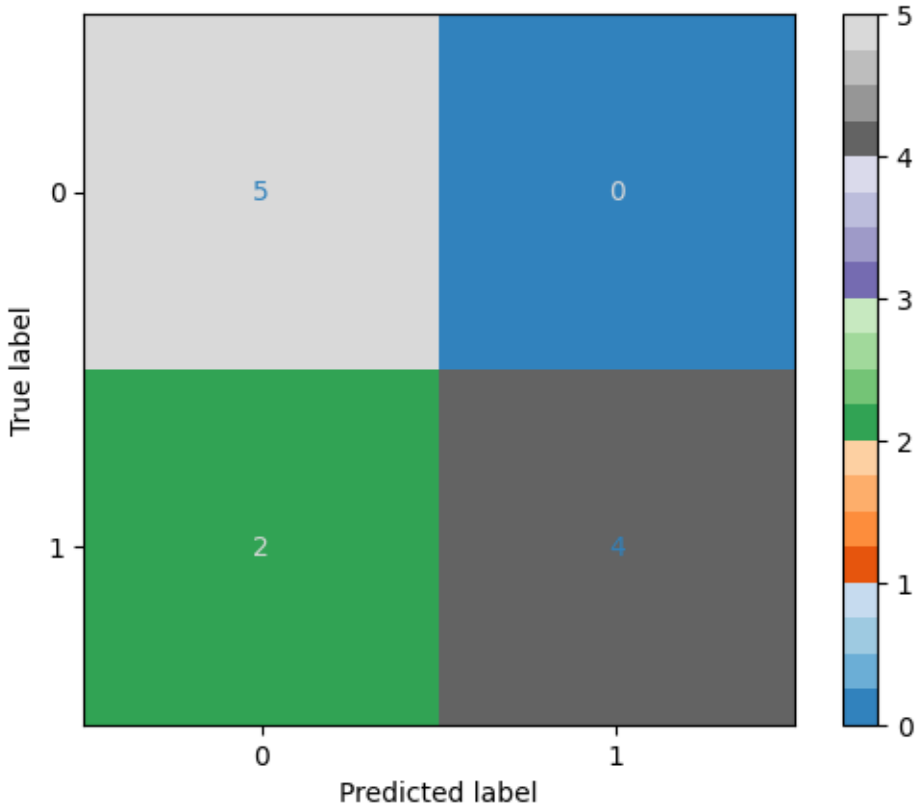
[42]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

1.9.2 Cálculo de la matriz de confusión

[45]:
```
mconf = confusion_matrix(y_cp, y_pred)
```

1.9.3 Impresión de la matriz de confusión

[54]:
```
mconfg = ConfusionMatrixDisplay(mconf).plot(cmap='tab20c')
```



1.9.4 Importación de método para la puntuación de precisión desde paquete *sklearn*

[55]:
```
from sklearn.metrics import accuracy_score
```

1.9.5 Cálculo de la puntuación de precisión

[56]:
```
cc = accuracy_score(y_cp, y_pred)
```

1.9.6 Impresión de la puntuación

[57]:
```
print(f'Accuracy Score = {cc}')
```

```
Accuracy Score = 0.8181818181818182
```

1.10 Importación de métodos para el gráfico

[58]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

1.11 Ajuste del etiquetado de la variable y

[59]:
```
# Importación del etiquetador
from sklearn.preprocessing import LabelEncoder
# Creación del etiquetador
labelencoder_y = LabelEncoder()
# Etiquetado y ajuste
y_ce = labelencoder_y.fit_transform(y_ce)
```

1.11.1 Nota: Es necesario realizar el ajuste de nuevo dado que cambió la variable y debido al proceso de etiquetado

[60]:
```
clasificador.fit(X_ce, y_ce)
```

[60]:
```
KNeighborsClassifier()
```

2 Se grafica todo el conjunto de datos empleando el clasificador DT para cada dato

[61]:
```
# Etiquetado y ajuste del conjunto de datos original
X_set, y_set = X, labelencoder_y.fit_transform(y)
```

2.1 Creación de la malla (plano cartesiano)

[62]:
```
X1, X2 = np.meshgrid(
    np.arange(start = X_set.iloc[:,0].min()-1, stop = X_set.iloc[:,0].max()+1,␣
 ↪step=0.1),
    np.arange(start = X_set.iloc[:,1].min()-1, stop = X_set.iloc[:,1].max()+1,␣
 ↪step=0.1)
)
```

2.2 Creación del gráfico

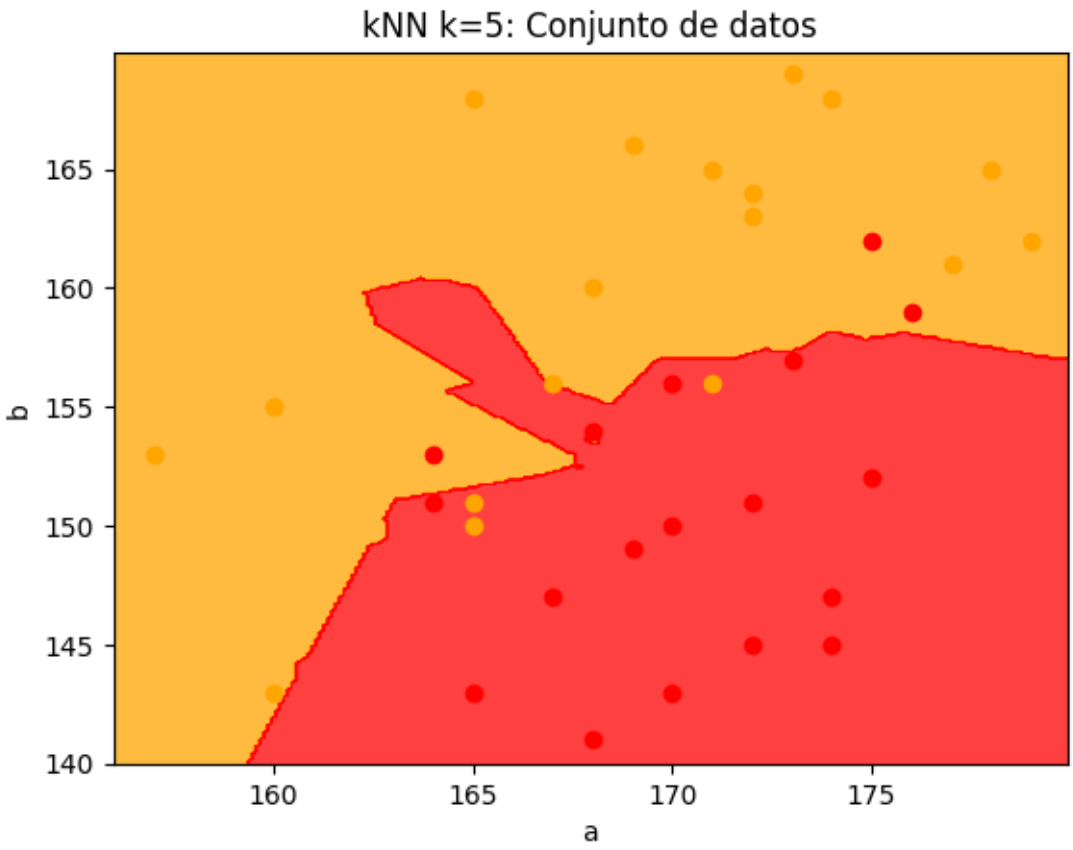
[63]:
```
# Al construir la malla, se colorea la región de naranja o rojo
# de acuerdo al clasificador DT obtenido
plt.contourf(X1, X2,
    clasificador.predict(
        np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap(('orange', 'red'))
)

# Se establecen los límites de los ejes x,y en el gráfico
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# Se grafica cada dato en el plano cartesiano, la clase de cada dato determina␣
 ↪el color.
# Debido al proceso de etiquetado, 'n' fue sustituido por 0 y 'r' sustituido␣
 ↪por 1
# 0 -> Naranja
# 1 -> Rojo
j=0
for i in y_set:
    if i==0:
        color = "orange"
    else:
        color = "red"
    plt.scatter(
        X_set.iloc[j,0],
        X_set.iloc[j,1],
        c = color,
        label = i
    )
    j=j+1

# Etiqueta del gráfico y sus ejes
plt.title('kNN k=5: Conjunto de datos')
plt.xlabel('a')
plt.ylabel('b')

# Creación del gráfico
plt.show()
```



3 Clasificar nuevos datos con kNN

3.1 Se clasifica un dato con el clasificador construido con kNN

dato = (160, 145)

[64]:
```
# Predicción del dato = (160, 145)
x = clasificador.predict([[160, 145]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
naranja
```

3.2 Se clasifica otro dato con el clasificador construido con kNN

dato = (160, 165)

[65]:
```
# Predicción del dato = (160, 165)
x = clasificador.predict([[160, 165]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
naranja
```

3.3 Ahora, a manera de prueba, se clasifica el promedio de los datos

[66]:
```
# X_set es un DataFrame de pandas
X_set.mean(0)
```

[66]:
```
a    169.694444
b    155.000000
dtype: float64
```

[67]:
```
# Predicción del dato promedio = (169.6944, 155)
x = clasificador.predict([[169.6944, 155]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
rojo
```