09_KNN

April 13, 2021

1 K NEAREST NEIGHBOURS (KNN)

1.1 Objetivo

Aprender a usar el algoritmo KNN conocido más frecuentemente en español “k vecinos”.

1.2 FICHA DEL ALGORITMO

1.2.1 Ámbito de aplicación

KNN se puede aplicar tanto en ámbitos de clasificación como de regresión.

Acepta directamente targets categóricas de múltiples valores.

Se aplica en datasets pequeños o medianos.

1.2.2 Pros

- Algoritmo muy sencillo
- Cubre casi todos los casos: target dicotómica, categórica de muchas clases y contínua
- Se puede implementar sin necesidad de SW analítico, por ej en Excel
- Rápido de usar

1.2.3 Contras

- Difícil de migrar (requiere llevarse todo el dataset)
- No indicado para datasets de muchos datos
- Sensible a maldición de la dimensionalidad
- Peor si el dataset tiene muchas categóricas porque incrementan más la dimensionalidad
- Es imprescindible escalar los datos
- Difícil ajustar el parámetro K

1.2.4 Necesidad de preprocesamiento

Sensible a outliers.

El escalado es imprescindible.

La preselección de variables es recomendable ya que con muchas variables no funcionará bien.

1.2.5 Supuestos e hipótesis No tiene.

1.2.6 Sobre ajuste

Depende mucho del parámetro K.

Podemos usar lo que aprenderemos en la sección de hiperparametrización para estimar el mejor valor de K. De momento “a ojo”.

1.2.7 Grado de interpretación

Alto.

La salida son los registros más similares al input, así que es muy fácil comparar, poner como ejemplo, etc.

1.2.8 Importación

En Sklearn la versión de clasificación y la de regresión vienen implementados en diferentes funciones.

[1]:
```
#Clasificación
from sklearn.neighbors import KNeighborsClassifier

#Regresión
from sklearn.neighbors import KNeighborsRegressor
```

1.2.9 CLASIFICACIÓN

Principales parámetros

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClas

- n_neighbors: lo que llamamos K
- weights: por defecto es ‘uniform’ pero podemos probar a ponerle ‘distance’ para que pondere más los casos más cercanos
- n_jobs: permite paralelizar

Principales atributos de resultado: No los usaremos mucho.

Principales métodos

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifie

- fit(): para entrenar
- predict_proba(): para generar el scoring
- kneighbors(): encuentra los k-vecinos de un punto dado
- get_params(): para extraer los parámetros del modelo entrenado

1.2.10 REGRESIÓN

Principales parámetros

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsReg

- n_neighbors: lo que llamamos K
- weights: por defecto es ‘uniform’ pero podemos probar a ponerle ‘distance’ para que pondere más los casos más cercanos
- n_jobs: permite paralelizar

Principales atributos de resultado No los usaremos mucho.

Principales métodos

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegress

- fit(): para entrenar
- predict(): para generar el scoring
- kneighbors(): encuentra los k-vecinos de un punto dado
- get_params(): para extraer los parámetros del modelo entrenado

1.3 EJEMPLO

1.3.1 Opciones y paquetes

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

#Automcompletar rápido
%config IPCompleter.greedy=True
```

1.3.2 Importación de datos

Vamos a usar el dataset sintético de target continua.

[3]:
```
#Cargamos el dataset sintetico_continua.csv que está en
00_DATASETS df =
pd.read_csv('../00_DATASETS/sintetico_continua.csv') df
```

[3]:
```
           x1       x2       x3       x4       x5       x6       x7 \
0   -0.333643 0.385530 1.733067 0.457594 0.047734 0.057823 0.986681
1   -0.915244 0.182435 0.962427 0.285146 0.538592 0.604320 1.274675
2   0.719608 -0.840527 -1.999640 -0.893353 0.826154 -1.076738 -1.902825
    3 -1.040670 -0.318717 -1.332247 -0.605161 -0.724380 -0.103789
    0.767642
40.118715 -0.593328 1.836431 -0.183321 -0.558974 -0.989050 1.208619
..       …       …       …       …       …       …       …
995  -1.466331 0.138283 -1.069003 -0.729966 0.248520 1.901812 -0.503945
996  1.503471 1.702940 -1.562491 -1.218632 1.402215 1.865504 -1.020819
997  0.148292 -0.829697 -0.192043 -0.204354 0.544293 1.538875 -0.119157
998  0.571149 0.955130 0.871865 0.689523 -0.163711 1.984590 -1.180553
999  -0.529185 1.460991 -0.111199 -0.017312 -0.774974 -0.234066 0.733808

          x8       x9     target
0 -0.310969 2.492347 137.101716
1 -1.306736 1.274135 43.190649
```

```
2     0.094109 0.414780 -
78.126159 31.352745 1.483947 -
219.544675
4   0.762238 0.775032 139.807003
..       …       …         …
995  1.018632 -0.580477 -140.829803
996  0.441283 -1.775826     65.138366
997  -1.241458 1.042977 -14.257548
998  0.297048 -0.118282 137.454992
999  -0.478443 -1.559640 -137.125562

[1000 rows x 10 columns]
```

1.3.3 Modelo

En lecciones anteriores habiámos visto que para este dataset las variables más predictoras eran: x3, x5, x1, x8, x4.

Así que vamos a coger solo estas para no incrementar el espacio de análisis.

[4]:
```
#Ajustamos df
df = df[['x3', 'x5', 'x1', 'x8', 'x4', 'target']]
df
```

[4]:
```
           x3       x5       x1       x8       x4     target
0    1.733067 0.047734 -0.333643 -0.310969 0.457594 137.101716
1    0.962427 0.538592 -0.915244 -1.306736 0.285146     43.190649
2    -1.999640 0.826154 0.719608 0.094109 -0.893353 -78.126159 3 -
     1.332247 -0.724380 -1.040670 1.352745 -0.605161 -219.544675
4 1.836431 -0.558974 0.118715 0.762238 -0.183321 139.807003
..       …       …       …       …       …         …
995  -1.069003 0.248520 -1.466331 1.018632 -0.729966 -140.829803
996  -1.562491 1.402215 1.503471 0.441283 -1.218632     65.138366
997  -0.192043 0.544293 0.148292 -1.241458 -0.204354 -14.257548
998  0.871865 -0.163711 0.571149 0.297048 0.689523 137.454992
999  -0.111199 -0.774974 -0.529185 -0.478443 -0.017312 -137.125562

[1000 rows x 6 columns]
```

Separar predictoras y target

[16]:
```
y = df['target']
```
```
x = df.drop(columns = 'target')
```

Separar train y test

[17]:
```
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y =

train_test_split(x,y,test_size=0.3)
```

Entrenar el modelo

[21]:
```
from sklearn.neighbors import KNeighborsRegressor

#Instanciar
kr = KNeighborsRegressor()

#Entrenar
```

```
kr.fit(train_x, train_y)
```

[21]:
```
KNeighborsRegressor()
```

Predecir sobre test

[22]:
```
pred = kr.predict(test_x)
```

Vamos a visualizar unos ejemplos para hacernos una idea

[23]:
```
pd.DataFrame({'real': test_y, 'prediccion': pred}).head(10)
```

[23]:

	real	prediccion
522	-135.764206	-70.245471
464	91.633837	81.906703
648	310.165780	199.837558
497	-122.039562	-112.029002
108	5.840548	-11.123119
591	18.769188	-9.839626
345	-256.927734	-176.135320
894	152.350566	95.687079
819	-154.135048	-152.585370
148	-31.988389	-22.447952

Evaluar sobre test

[24]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(test_y,pred)
```

[24]:
```
30.37186827450155
```

Revisión de los parámetros de entrenamiento

[25]:
```
kr.get_params()
```

[25]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

Extracción de los casos más cercanos En muchas aplicaciones de KNN lo que querremos hacer no es tanto clasificar un nuevo caso, si no extraer los más cercanos del histórico al caso de referencia.

Por ejemplo extraer los pacientes históricos con síntomas más parecidos al paciente actual para ver cómo se trataron.
En ese caso podemos usar el método kneighbors().

A este método hay que pasarle el caso actual como un array con una forma concreta con reshape(1, -1).

Como ejemplo vamos a buscar los 3 casos más parecidos al primer registro de test.

Este método nos devuelve las distancias de cada caso al nuestro y los índices para que podamos buscarlos.

[26]:
```
#Transformarmos al formato que pide
caso_actual = test_x.iloc[0].values.reshape(1, -1)
caso_actual
```

[26]:
```
array([[-2.19692174, 1.12226787, -0.09742149, -2.11959698,
2.01184202]])
```

[27]:
```
kr.kneighbors(caso_actual,n_neighbors=3)
```

```
C:\Users\isaac\miniconda3\envs\PDSM\lib\site-
packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
KNeighborsRegressor was fitted with feature names
  warnings.warn(
```

[27]:
```
(array([[1.34612418, 1.47978748, 1.59443132]]),
 array([[ 97, 600, 135]], dtype=int64))
```

Vamos a extraer el caso más parecido.

[28]:
```
df.loc[680]
```

[28]:
```
x3      -1.066498
x5       1.233139
x1       1.667772
x8      -0.044105
x4       0.203026
target   128.209237
```

```
Name: 680, dtype: float64
```

[ ]:

[ ]:

[ ]: