K_naechste_nachbarn

January 2, 2022

1 K-nächste Nachbarn (K-nearest Neighbors)

Wir wenden den K-nächste-Nachbarn-Algorithmus auf den IRIS-Datensatz an.

[1]:
```
# Wir laden den Datensatz
import pandas as pd
url = "https://raw.githubusercontent.com/troescherw/datasets/master/iris.csv"
iris = pd.read_csv(url, delimiter=";")
iris
```

[1]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species	SpeciesID
0	5.1	3.5	1.4	0.2	setosa	0
1	4.9	3.0	1.4	0.2	setosa	0
2	4.7	3.2	1.3	0.2	setosa	0
3	4.6	3.1	1.5	0.2	setosa	0
4	5.0	3.6	1.4	0.2	setosa	0
..	…	…	…	…	…	…
145	6.7	3.0	5.2	2.3	virginica	2
146	6.3	2.5	5.0	1.9	virginica	2
147	6.5	3.0	5.2	2.0	virginica	2
148	6.2	3.4	5.4	2.3	virginica	2
149	5.9	3.0	5.1	1.8	virginica	2

[150 rows x 6 columns]

[2]:
```
# Aufteilen in X und y
X = iris.iloc[:, :4]
y = iris.SpeciesID
```

[3]:
```
# Aufteilen in Trainings- und Testdaten
# "Training" bedeutet hier das Erstellen einer Abstandsmatrix

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True,␣
↪test_size=0.3)
```

Wir erstellen das Modell und verwenden für k den Wert 5.

[4]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)
```

Wir klassifizieren die Objekte aus dem Test-Datensatz.

[5]:
```
pred = knn.predict(X_test)
pred
```

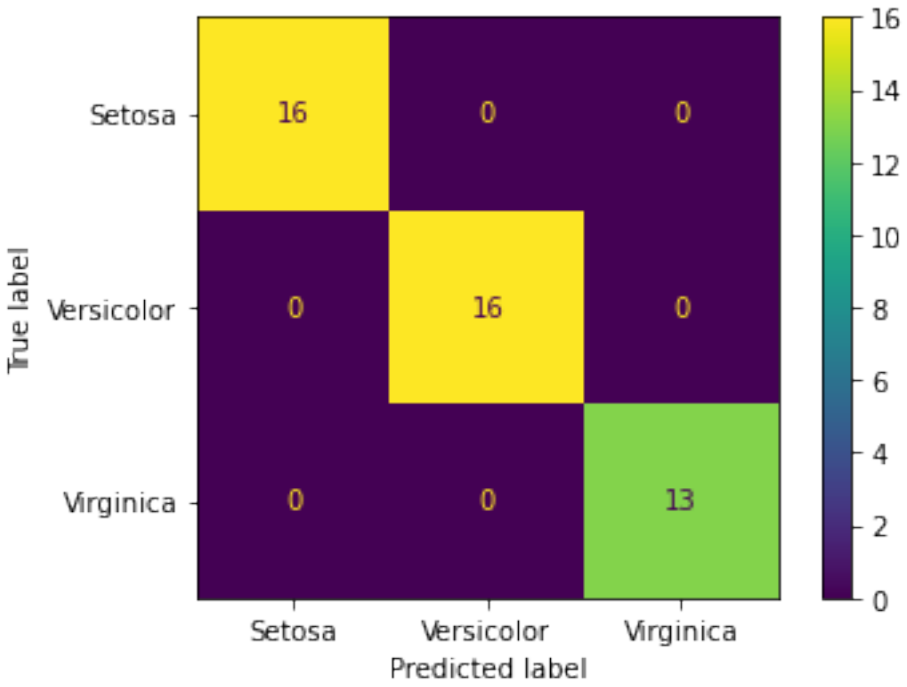
[5]:
```
array([2, 1, 1, 2, 0, 1, 2, 0, 1, 1, 0, 0, 0, 1, 0, 2, 0, 0, 1, 2, 1, 2,
       2, 2, 2, 1, 1, 2, 0, 0, 0, 1, 1, 0, 2, 0, 0, 1, 1, 0, 2, 1, 1, 2,
       0], dtype=int64)
```

Wir stellen das Ergebnis in einer Confusion Matrix dar und berechnen die Accuracy.

[6]:
```
from sklearn.metrics import plot_confusion_matrix
import numpy as np

classes = ["Setosa", "Versicolor", "Virginica"]
_=plot_confusion_matrix(knn, X_test, y_test, display_labels=classes)
```

```
C:\Users\dea40349\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



[7]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

[7]: 1.0

1.1 Optimierung des Algorithmus

Für die Zuordnung des Objektes zu einer Klasse gibt es mehrere Möglichkeiten: Im einfachsten Fall wird wie schon erwähnt eine schlichte Mehrheitsentscheidung getroffen, was aber nicht immer das beste Ergebnis liefert, insbesondere wenn sich ähnlich oder sogar gleich viele Objekte der jeweiligen Klasse in unmittelbarer Nachbarschaft befinden. Daher kann man altnernativ die Objekte in der Nachbarschaft gewichten: Je näher ein Objekt, desto mehr "Gewicht" erhält die jeweilige Klasse.

Dem Konstruktor der Klasse *KNeighborsClassifier* kann man deshalb noch das Attribut *weights* bestimmen. Standardmäßig ist dies auf *uniform* gesetzt, was der ersten, einfachen Methode entspricht. Man kann es aber auch auf den Wert *distance* setzen, dann werden die Abstände gewichtet.

Im folgenden Skript wollen wir nun anhand eines Datensatzes das optimale K und auch die optimale Gewichtung der Abstände bestimmen. Wir verwenden hierfür einen Datensatz aus dem *sklearn*-Package, der über chemische Analysedaten von Weinen verfügt. Jeder der 178 Weine stammt von einem von drei italienischen Winzern ("cultivator"). Wie gut kann unser Modell vorhersagen, von welchem Winzer der Wein stammt?

Um unser Modell zu optimieren werden wir ein K von 1 bis 10 verwenden und für jedes K ein unterschiedliches Verfahren für die Gewichtung verwenden. Es werden also insgesamt 20 Modelle durchprobiert und für jedes ermitteln wir die Accuracy. Die Kombination aus K und dem Typ für

die Gewichtung mit der höchsten Accuracy liefert (vermutlich) die besten Hyperparameter.

[8]:
```
# Laden des Datensatzes
from sklearn.datasets import load_wine

wines = load_wine()
print(wines.DESCR)
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
                - Total phenols
                - Flavanoids
                - Nonflavanoid phenols
                - Proanthocyanins
                - Color intensity
                - Hue
                - OD280/OD315 of diluted wines
                - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================== ==== ===== ======= =====
                                    Min   Max   Mean     SD
    ============================== ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
```

```
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================ ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
```

(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

[9]:
```
# Aufteilen in Trainings- und Testdaten
# "Training" bedeutet hier das Erstellen einer Abstandsmatrix

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(wines.data, wines.target,␣
↪shuffle=True, test_size=0.3, random_state=23 )
```

[10]:
```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np


weights = ["uniform", "distance"]
ks = np.arange(1,11)


for weight in weights:
    for k in ks:
        knn = KNeighborsClassifier(n_neighbors=k, weights=weight).fit(X_train,␣
↪y_train)
        print(f"K={k}, weights={weight}, Accuracy = {knn.score(X_test, y_test):.
↪3}")
```

```
K=1, weights=uniform, Accuracy = 0.796
K=2, weights=uniform, Accuracy = 0.685
K=3, weights=uniform, Accuracy = 0.704
K=4, weights=uniform, Accuracy = 0.648
K=5, weights=uniform, Accuracy = 0.648
K=6, weights=uniform, Accuracy = 0.704
K=7, weights=uniform, Accuracy = 0.667
K=8, weights=uniform, Accuracy = 0.685
K=9, weights=uniform, Accuracy = 0.648
K=10, weights=uniform, Accuracy = 0.704
K=1, weights=distance, Accuracy = 0.796
K=2, weights=distance, Accuracy = 0.796
K=3, weights=distance, Accuracy = 0.759
K=4, weights=distance, Accuracy = 0.722
K=5, weights=distance, Accuracy = 0.722
K=6, weights=distance, Accuracy = 0.685
```

```
K=7, weights=distance, Accuracy = 0.722
K=8, weights=distance, Accuracy = 0.722
K=9, weights=distance, Accuracy = 0.759
K=10, weights=distance, Accuracy = 0.741
```

1.2 KNN für Regressionsanalyse

KNN kann auch für Regression verwendet werden. Im Beispiel werden Mietpreise für Wohnungen vorherhergesagt.

[11]:
```
from sklearn.neighbors import KNeighborsRegressor
import numpy as np

# Mietpreise
X = np.array([
        [69, 1685],
        [28, 625],
        [42, 524],
        [113, 2100],
        [54, 1200],
        [43, 750],
        [62, 1178],
        [24, 900],
        [33, 715],
        [92, 2915],
        [53, 1440]
       ]
    )

knn = KNeighborsRegressor(n_neighbors=3).fit(X[:,0].reshape(-1,1), X[:,1].
 ↪reshape(-1,1))

# Predictions
qm = np.array([50,70,90]).reshape(-1,1)
print(knn.predict(qm))
```

```
[[1130.        ]
 [1354.33333333]
 [2233.33333333]]
```

[12]:
```
# Visualisierung
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")
plt.scatter(X[:,0], X[:,1], label="Trainingsdaten")
plt.scatter(qm, knn.predict(qm), label="Vorhersagen")
plt.legend()
plt.title("Mietpreise Wohnungen mit KNN")
plt.xlabel(r"Größe der Wohnung in $m^2$")
```

```
plt.ylabel("Mietpreis in €")
plt.show()
```