NaiveBayes

June 5, 2021

1 Naive Bayes Klassifikator

Als einführendes Beispiel wollen wir mit Hilfe des Naive-Bayes Klassifikators Obsorten, Äpfel und Birnen, andhand des Gewichts und Zuckergehalts klassifizieren.

Wir laden die Daten in ein Dataframe: Zuckergehalt und Gewicht von Äpfeln und Birnen:

[1]:
```
import pandas as pd
import numpy as np

url="https://raw.githubusercontent.com/troescherw/datasets/master/obst.csv"
df = pd.read_csv(url, delimiter=";")
df
```

[1]:

	Zuckergehalt	Gewicht	Obstsorte
0	12.0	112	Apfel
1	10.0	100	Apfel
2	9.0	120	Apfel
3	12.0	119	Apfel
4	11.0	115	Apfel
5	13.0	113	Apfel
6	12.0	114	Apfel
7	15.0	150	Birne
8	16.0	149	Birne
9	14.0	147	Birne
10	13.6	151	Birne
11	15.0	150	Birne
12	14.7	149	Birne
13	13.0	140	Birne

Wir können nun unser Modell erstellen.

[2]:
```
from sklearn.naive_bayes import GaussianNB

X = df[["Zuckergehalt", "Gewicht"]]
y = df.Obstsorte
model = GaussianNB().fit(X,y)
```

Mit Hilfe des Modells können wir nun zwei "unbekannte" Obststücke klassifiziert werden. Haben wir ein Stück Obst, das z.B. ein Zuckergehalt von 52,5g und ein Gewicht von 125g verfügt, fragen wir das Modell, ob es sich um einen Apfel oder eine Birne handelt:

[3]:
```
unbekanntesObst = pd.DataFrame({"Zuckergehalt": [11.5, 15.1], "Gewicht":[110,␣
↪135]})

print(unbekanntesObst)
model.predict(unbekanntesObst)
```

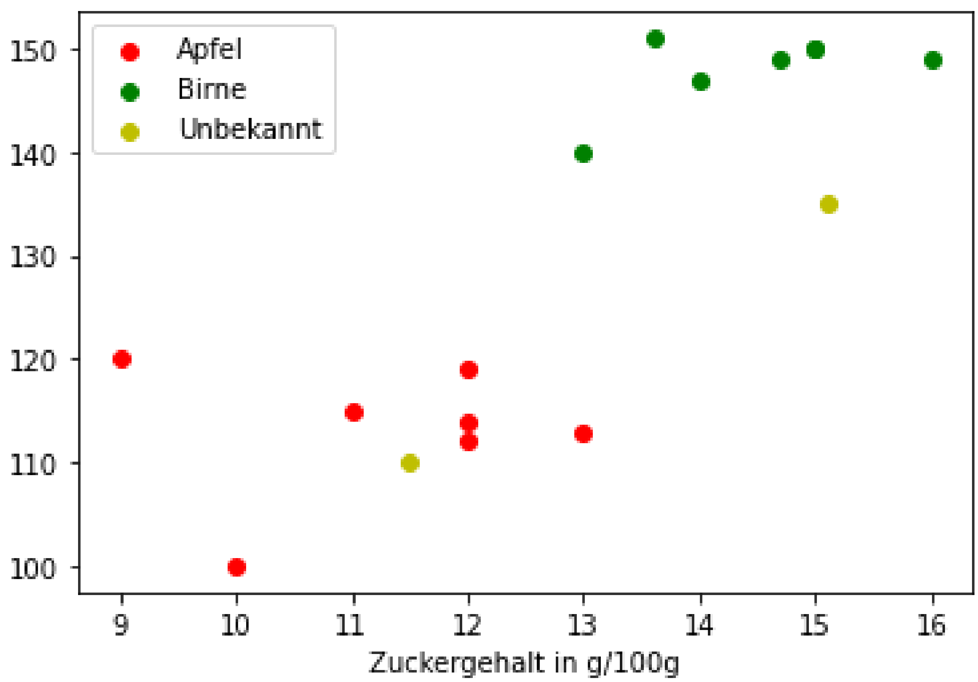
```
   Zuckergehalt  Gewicht
0          11.5      110
1          15.1      135
```

[3]:
```
array(['Apfel', 'Birne'], dtype='<U5')
```

Das Modell gibt eine **0** zurück, damit handelt es sich um einen Apfel. Wir erstellen ein Scatterplot und fügen dort auch das unbekannte Stück Obst ein:

[4]:
```
import matplotlib.pyplot as plt

plt.scatter(df[df.Obstsorte=="Apfel"].Zuckergehalt, df[df.Obstsorte=="Apfel"].
↪Gewicht, c="r", label="Apfel")
plt.scatter(df[df.Obstsorte=="Birne"].Zuckergehalt, df[df.Obstsorte=="Birne"].
↪Gewicht, c="g", label="Birne")
plt.scatter(unbekanntesObst.Zuckergehalt, unbekanntesObst.Gewicht, c="y",␣
↪label="Unbekannt")
plt.xlabel("Zuckergehalt in g/100g")
plt.legend()
plt.plot()
```

[4]:
```
[]
```



1.1 Beispiel: Naive-Bayes-Modell für die Vorhersage von Brustkrebs

Wir erstellen anhand des bereits verwendeten Datensatzes *breast_cancer* eine Prognose, ob eine Patientin anhand der vorliegenden Daten an Brustkrebs erkrankt ist (gutartiges oder bösartiges Melanom). Um die Modellqualität zu prüfen führen wir eine k-Fold-Cross-Validation durch (mit k=10).

Wir laden dazu den Datensatz, der von sklearn stammt, und geben die Beschreibung aus:

[5]:
```python
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()
print(bc.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
```

```
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features.  For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

===================================== ====== ======
                                       Min    Max
===================================== ====== ======
radius (mean):                        6.981  28.11
texture (mean):                       9.71   39.28
perimeter (mean):                     43.79  188.5
area (mean):                          143.5  2501.0
smoothness (mean):                    0.053  0.163
compactness (mean):                   0.019  0.345
concavity (mean):                     0.0    0.427
concave points (mean):                0.0    0.201
symmetry (mean):                      0.106  0.304
fractal dimension (mean):             0.05   0.097
radius (standard error):              0.112  2.873
texture (standard error):             0.36   4.885
perimeter (standard error):           0.757  21.98
area (standard error):                6.802  542.2
smoothness (standard error):          0.002  0.031
compactness (standard error):         0.002  0.135
concavity (standard error):           0.0    0.396
concave points (standard error):      0.0    0.053
symmetry (standard error):            0.008  0.079
fractal dimension (standard error):   0.001  0.03
radius (worst):                       7.93   36.04
texture (worst):                      12.02  49.54
perimeter (worst):                    50.41  251.2
area (worst):                         185.2  4254.0
```

```
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======
```

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction

```
      for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
      Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
      San Jose, CA, 1993.
    - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
      prognosis via linear programming. Operations Research, 43(4), pages
570-577,
      July-August 1995.
    - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning
techniques
      to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
      163-171.
```

[6]:
```python
# Wir laden die Daten in X und y
X = bc.data
y = bc.target
```

Wir teilen die Datensätze in 10 Folds auf.

[7]:
```python
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle=True)
```

Nun iterieren wir durch die Folds und trainieren das Modell jeweils mit den Daten in den Folds:

[8]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    shuffle=True,
                                                    random_state=1,
                                                    test_size=0.3)

scores = [] # Leere Liste für Scores

for index_train, index_test in kf.split(X):
    X_train = X[index_train]
    X_test = X[index_test]
    y_train = y[index_train]
    y_test = y[index_test]
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))

scores
```

[8]:
```
[0.8947368421052632,
 0.9298245614035088,
 0.9122807017543859,
```

```
0.9298245614035088,
 0.9122807017543859,
 0.9473684210526315,
 0.9122807017543859,
 0.9473684210526315,
 1.0,
 1.0]
```

[9]:
```python
print(f"Mittelwert Accuracy:  {np.mean(scores)}")
print(f"Standardabweichung der Accuracy: {np.std(scores)}")
```

```
Mittelwert Accuracy:  0.9385964912280702
Standardabweichung der Accuracy: 0.03442353836903261
```

1.2 Beispiel: Ziffernerkennung

Im Package *sklearn.datasets* befindet sich ein Datensatz *digits*, der Graustufenbilder von handschriftlich erstellten Ziffern 0..9 enthält. Wir versuchen nun mit Hilfe eines Naiven Bayes Klassifikators diese Graustufenbilder den richtigen Klassen (0..9) zuzuordnen.

Wir laden den Datensatz und geben die Beschreibung aus.

[10]:
```python
from sklearn.datasets import load_digits
digits = load_digits()
print(digits.DESCR)
```

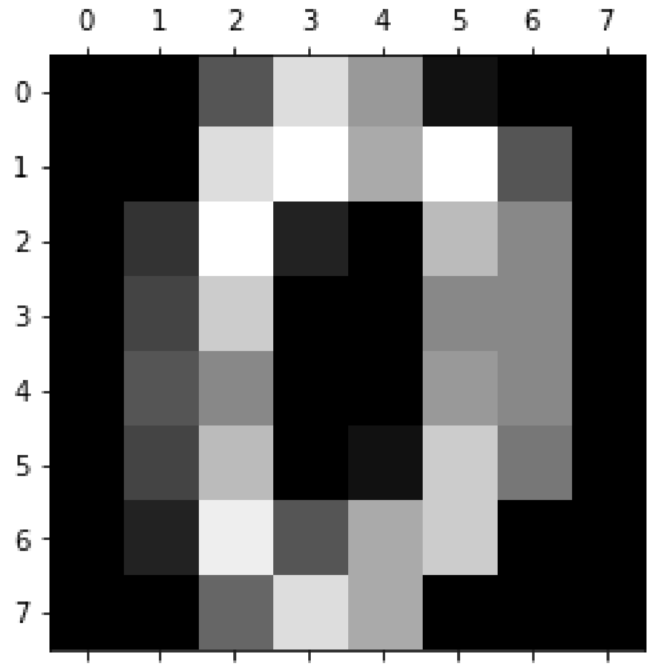
```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digit
s

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
```

```
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

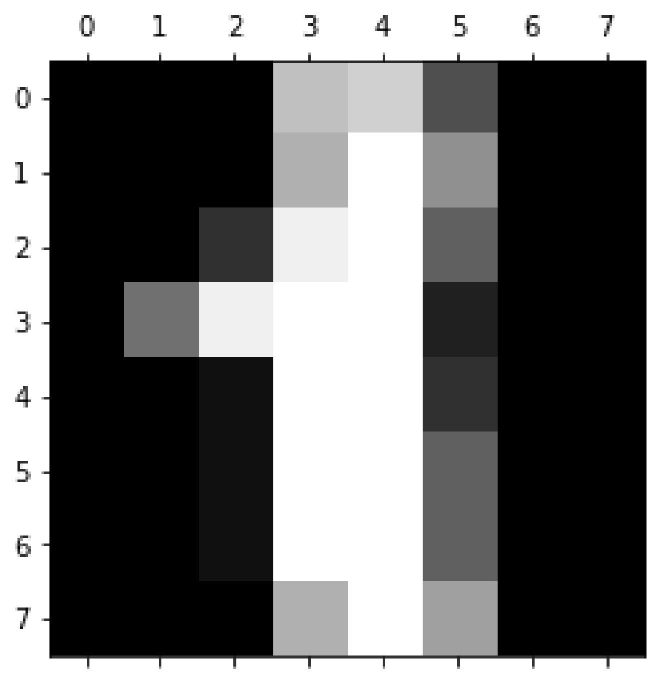
Wir geben exemplarisch die ersten 5 Bitmap-Bilder aus:

[11]:
```python
import matplotlib.pyplot as plt
for i in range(5):
    plt.gray()
    plt.matshow(digits.images[i])
    plt.show()
```

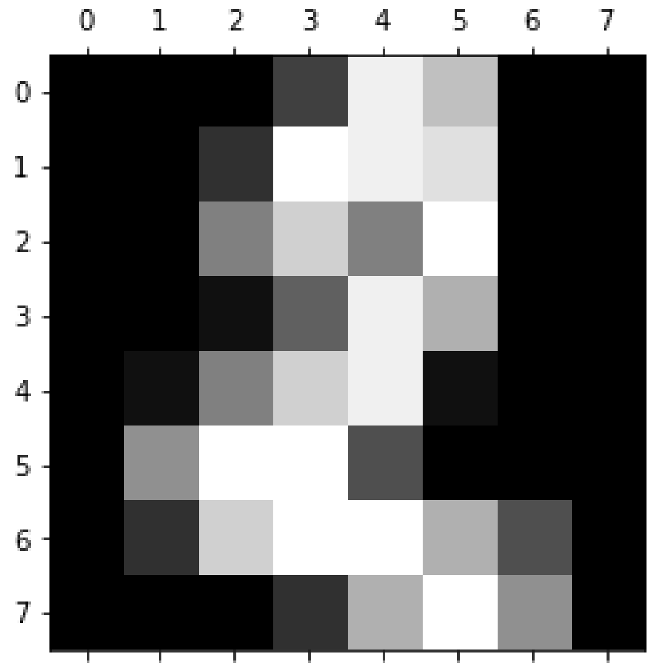
```
<Figure size 432x288 with 0 Axes>
```



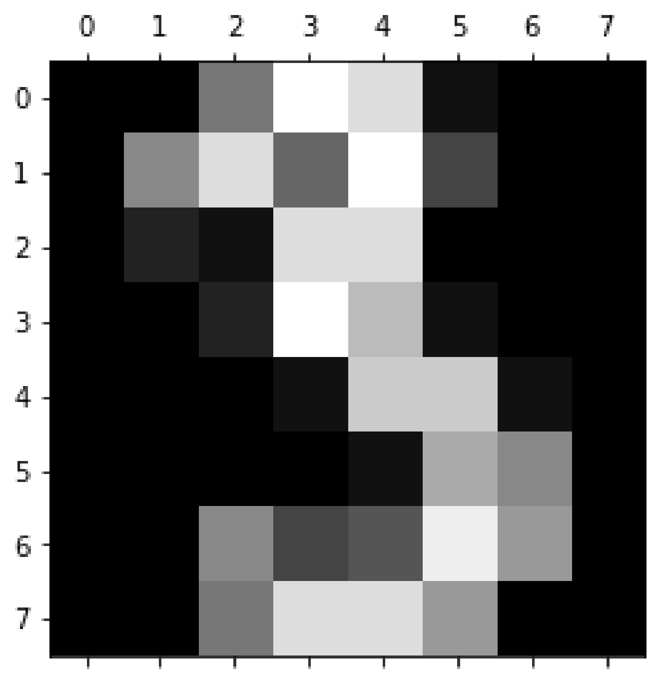
<Figure size 432x288 with 0 Axes>



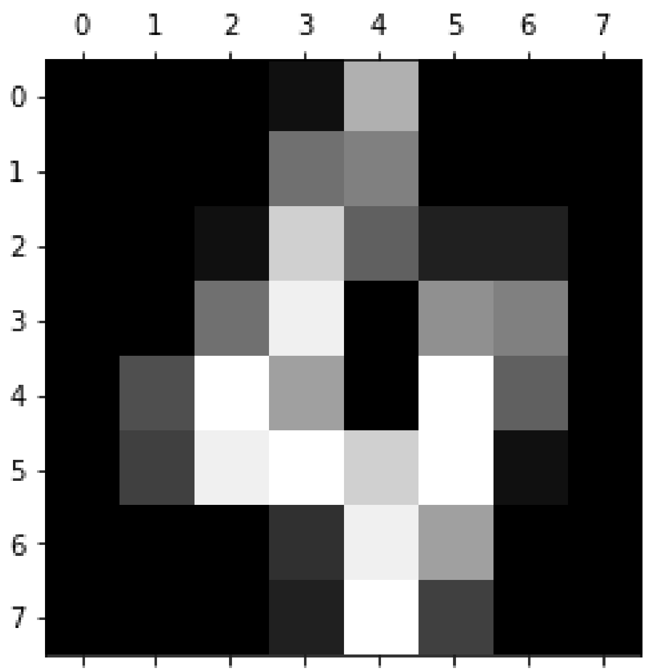
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

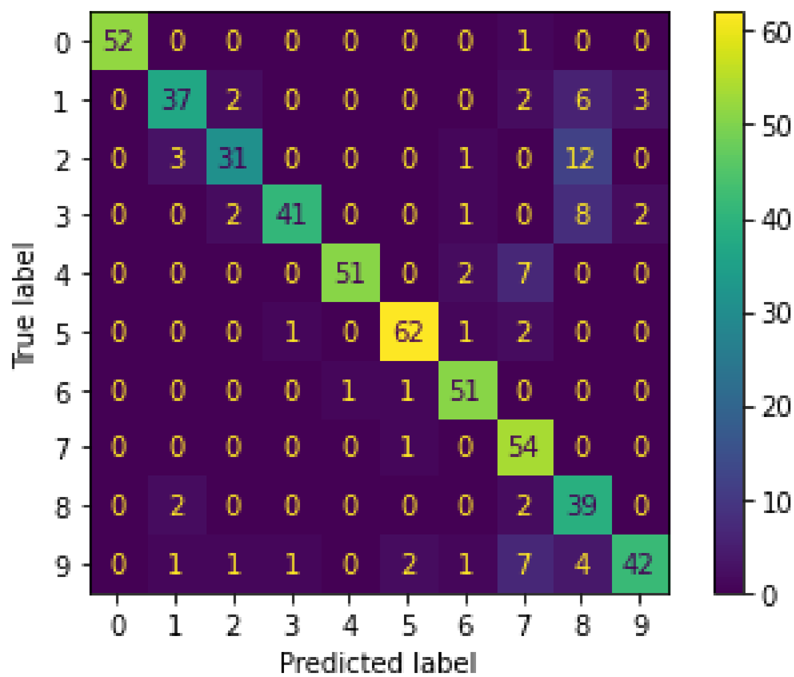


Es werden wie üblich die Train- Testdatasets erstellt und das Modell und eine Prediction erstellt. Wir geben die Accuracy und die Confusion Matrix aus.

[12]:
```
from sklearn.metrics import plot_confusion_matrix
digits = load_digits()

X = digits.data
y = digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪shuffle=True, random_state=42)

model = GaussianNB().fit(X_train, y_train)
pred = model.predict(X_test)

print(model.score(X_test, y_test))
plot_confusion_matrix(model, X_test, y_test)
```

```
0.8518518518518519
```

[12]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1cf452a3e20>
```


True label
Predicted label
0
1
2
3
4
5
6
7
8
9
60
50
40
30
20
10
0