Woche 08: Programmierung – Entscheidungsbaumklassifikation mit Scikit-learn

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Ein Entscheidungsbaummodell für einen gegebenen Datensatz erstellen können
- Ein trainiertes Entscheidungsbaummodell nutzen können, um für Beobachtungen der Testmenge die Klasse zu prognostizieren
- Die Güte eines Klassifikationsmodells mit Accuracy bewerten können
- Einen Entscheidungsbaum und die Entscheidungsregeln grafisch darstellen können

Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

Inhalt

Einstieg

Ein Entscheidungsbaummodell leitet aus den Merkmalswerten der Beobachtungen Entscheidungsregeln ab, mit deren Hilfe sich die Werte des Zielmerkmals bzw. die Klassenzugehörigkeit der Beobachtungen bestimmen lassen.

Vorbereitung des Datensatzes

Im Folgenden arbeiten wir mit dem *Wine Recognition Dataset*, das Ergebnisse einer chemischen Analyse von 178 Weinen von drei verschiedenen Rebsorten enthält. Bei der Analyse wurden die Mengen von 13 Inhaltsstoffen wie z. B. Alkohol, Apfelsäure und Magnesium bestimmt, die in den Weinen dieser drei Sorten enthalten sind. Die Merkmalswerte im Datensatz repräsentieren die in den Weinen gemessenen Inhaltsstoffmengen. Zumindest stimmt das für die meisten Merkmale, da für die Merkmale *Farbtiefe* und *Farbton* keine ausreichende Beschreibung vorhanden ist. Die Klassifikationsaufgabe bei diesem Datensatz besteht darin, anhand der Inhaltsstoffe eines Weins seine Rebsorte zu bestimmen.

Quelle [1]

Zuerst importieren wir alle benötigten Module in das Programm.

```
# Importiere die nötigen Module
from sklearn import tree, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objects as go
```

Wir laden den *Wine*-Datensatz mit der Funktion *load_wine()*, die im *Scikit-learn*-Untermodul *Data Sets* zur Verfügung steht. Damit die Werte der beschreibenden Merkmale und des Zielmerkmals in getrennte *ndarrays* geladen werden, wird der Parameter *return_X_y* auf *True* gesetzt. Wir speichern die Datenmatrix mit den Inhaltsstoffmengen der Weine im *ndarray X_wine* und die Werte der Zielvariable, also die Rebsorten, im *ndarray y_wine* ab.

Quelle [2]

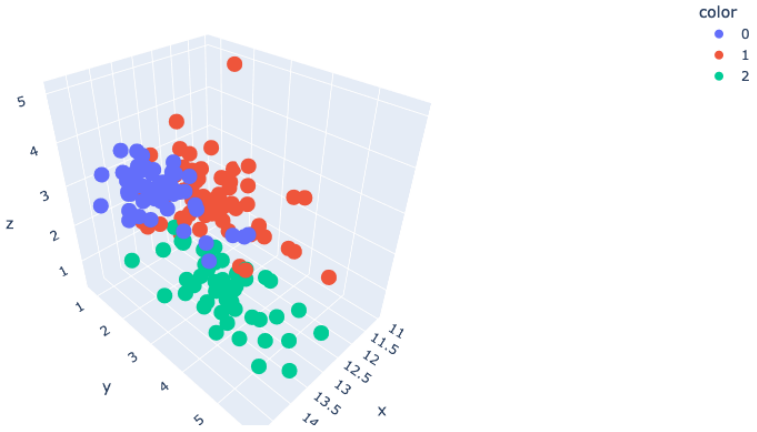
```
# Lade den Wine Datensatz
X_wine, y_wine = datasets.load_wine(return_X_y=True)
```

Wir stellen die Beobachtungen bzw. Weine durch ihre Werte für die Merkmale *Alkoholgehalt*, *Apfelsäuregehalt* und *Flavonoidgehalt* in einem 3-dimensionalen Diagramm

grafisch dar. Die Rebsorten der Weine werden durch unterschiedliche Farben der Datenpunkte markiert.

```
pyo.init_notebook_mode()
fig1 = px.scatter_3d(x=X_wine[:,0], y=X_wine[:,1],
                     z=X_wine[:,6], color=y_wine.astype(str))
fig1.show()
```

Im Diagramm sehen wir, dass die Weine verschiedener Rebsorten sich anhand der Werte dieser drei Merkmale zwar gut unterscheiden lassen, aber keine klare Trennung zwischen den einzelnen Rebsorten zu beobachten ist.



In diesem Video versuchen wir die Klassifikationsaufgabe mit Hilfe eines Entscheidungsbaummodells zu lösen. Bevor aber das Modell erstellt wird, muss der Datensatz in eine Trainings- und eine Testmenge zerlegt werden. Dafür benutzen wir die Funktion *train_test_split()* des Untermoduls *Model Selection* von *Scikit-learn*. Diese Funktion erwartet als Eingabe die Datenmatrix mit dem entsprechenden Zielmerkmal. Mit dem Parameter *test_size* legen wir die Größe der Testmenge fest. In den Voreinstellungen der Funktion ist festgelegt, dass die Beobachtungen vor der Zerlegung gemischt werden. Für diesen Datensatz ist es besonders wichtig, diese Voreinstellung beizubehalten, weil die Beobachtungen in der ursprünglichen Datenmatrix nach dem Zielmerkmal sortiert sind. Würden wir auf das Mischen verzichten, wäre die resultierende Trainingsmenge für die gesamte Datenmenge nicht repräsentativ, da eine Rebsorte entweder unterrepräsentiert oder gar nicht in der Trainingsmenge vertreten wäre. Mit der Wertzuweisung des Parameters *random_state* stellen wir aber sicher, dass die Beobachtungen im Datensatz auf eine bestimmte Weise durchmischt werden. Damit sichern wir auch die Reproduzierbarkeit unserer Zerlegung.

Quelle [3]

```
# Teile den Datensatz in die Trainings- und die Testmenge auf
X_train, X_test, y_train, y_test = train_test_split(
    X_wine, y_wine, test_size=0.1, random_state=45)
```

Entscheidungsbaummodell

Wir erstellen das Entscheidungsbaummodell mit der Funktion *DecisionTreeClassifier()* des *Scikit-learn*-Untermoduls *Decision Trees*. Das Entscheidungsbaummodell kann durch viele Parameter dieser Funktion vorbestimmt werden. Im Folgenden schauen wir uns nur die Wichtigsten von ihnen an.

- Mit dem Parameter *criterion* kann die Funktion zur Qualitätsmessung einer Aufspaltung der inneren Knoten im Baum gewählt werden. Dabei kann man zwischen dem *Gini-Index* und der *Entropie* wählen. Die beiden Funktionen bewerten die Verunreinigung der Klassen in den Teildatensätzen auf etwas unterschiedliche Weisen.
- Der Wert für den Parameter *max_depth* bestimmt die maximale Tiefe des Entscheidungsbaums. Mit diesem Parameter können wir also festlegen, wie viele Entscheidungsregeln maximal geprüft werden müssen, um für eine Beobachtung eine Klasse zu prognostizieren.
- Mit dem Parameter *max_leaf_nodes* wird bestimmt, wie viele Blätter der resultierende Entscheidungsbaum enthalten soll. Dabei werden die besten Blätter ausgewählt, d. h. möglichst reine Blätter in Bezug auf die Klassenzugehörigkeit der Beobachtungen.
- Der Wert des Parameters *min_samples_leaf* bestimmt die minimale Anzahl der Beobachtungen in den Blättern des Entscheidungsbaums.
- Bei der Aufspaltung eines inneren Knotens kann es passieren, dass mehr als nur ein Merkmal zur Verringerung der Verunreinigung der Klassen in den Teildatensätzen beitragen kann. In diesem Fall wird ein Merkmal zufällig ausgewählt, auf deren Grundlage die Aufspaltung des Knotens durchgeführt wird. Mit der Wertzuweisung des Parameters *random_state* kann die Merkmalswahl in solchen Fällen festgelegt werden.

Quelle [4]

Für unser Entscheidungsbaummodell setzen wir nur den Wert des Parameters *max_depth* auf 3, um die Tiefe des Entscheidungsbaums kleinzuhalten. Dieser Parameter wird oft gesetzt, um eine Überanpassung des Entscheidungsbaums zu vermeiden. Eine *Überanpassung* oder *Overfitting* tritt dann auf, wenn ein Modell genaue Vorhersagen für die Beobachtungen der Trainingsmenge liefert, nicht aber für die Beobachtungen der Testmenge. Von einem überangepassten Modell können wir also nicht erwarten, dass es auch die neuen Beobachtungen richtig klassifizieren kann.

Um die Reproduzierbarkeit unseres Entscheidungsbaummodells und folglich unserer Klassifikationsergebnisse für den Wine-Datensatz zu garantieren, setzen wir noch den Parameter *random_state* auf 45.

Mit der Funktion *fit()* trainieren wir unser Entscheidungsbaummodell mit den Beobachtungen der Trainingsmenge.

```
# Erstelle das Entscheidungsbaummodell
dt_classifier = tree.DecisionTreeClassifier(max_depth=3, random_state=45)

# Trainiere das Modell mit der Trainingsmenge
dt_classifier.fit(X_train, y_train)
```

Mit der Funktion *predict()* kannst du die Klasse für neue Beobachtungen prognostizieren. Wir prognostizieren die Klassenzuordnung für alle Beobachtungen der Testmenge.

```
# Erstelle die Vorhersage für die Testmenge
y_pred = dt_classifier.predict(X_test)
```

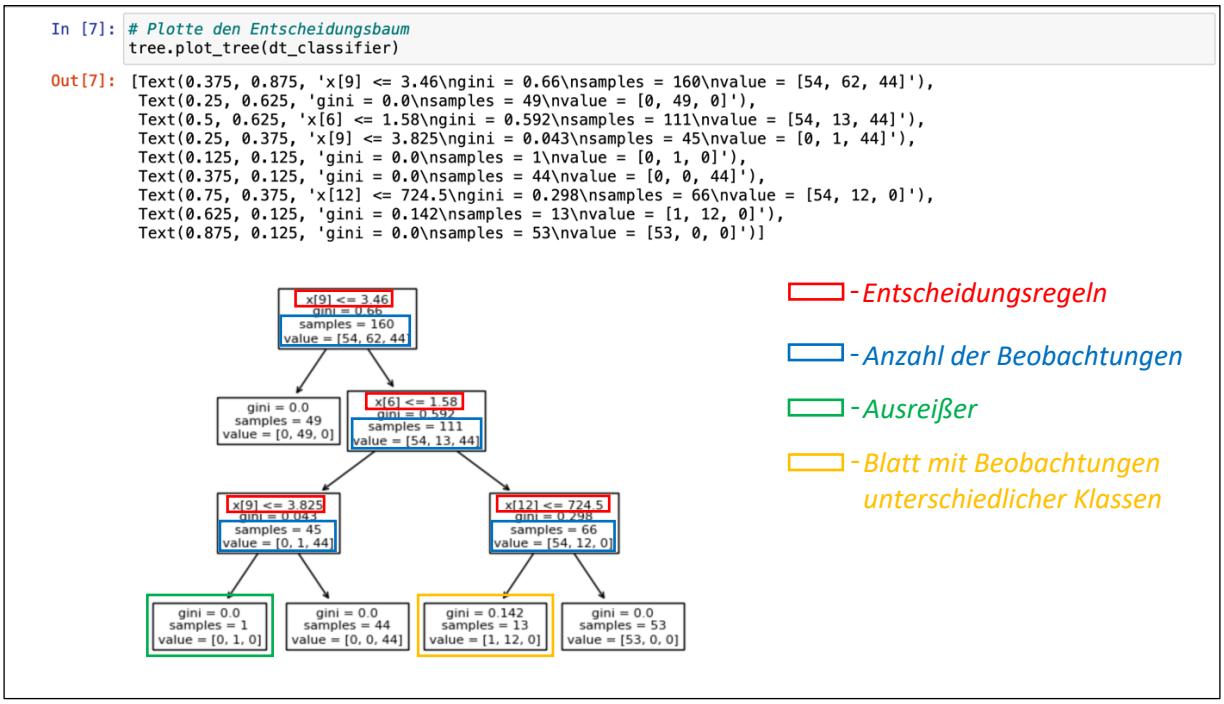
Mit der Funktion *plot_tree()* kann man den Entscheidungsbaum und die erlernten Regeln grafisch darstellen. Das machen wir jetzt.

Quelle [5]

```
# Plotte den Entscheidungsbaum
tree.plot_tree(dt_classifier)
```

In der Wurzel und den inneren Knoten des Entscheidungsbaums kannst du die Regeln sehen, die bei der Aufspaltung der Beobachtungen getroffen wurden. Außerdem ist die Anzahl der Beobachtungen insgesamt und nach Klasse aufgeteilt innerhalb der inneren Knoten aufgelistet.

```
In [7]: # Plotte den Entscheidungsbaum
        tree.plot_tree(dt_classifier)

Out[7]: [Text(0.375, 0.875, 'x[9] <= 3.46\ngini = 0.66\nsamples = 160\nvalue = [54, 62, 44]'),
         Text(0.25, 0.625, 'gini = 0.0\nsamples = 49\nvalue = [0, 49, 0]'),
         Text(0.5, 0.625, 'x[6] <= 1.58\ngini = 0.592\nsamples = 111\nvalue = [54, 13, 44]'),
         Text(0.25, 0.375, 'x[9] <= 3.825\ngini = 0.043\nsamples = 45\nvalue = [0, 1, 44]'),
         Text(0.125, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
         Text(0.375, 0.125, 'gini = 0.0\nsamples = 44\nvalue = [0, 0, 44]'),
         Text(0.75, 0.375, 'x[12] <= 724.5\ngini = 0.298\nsamples = 66\nvalue = [54, 12, 0]'),
         Text(0.625, 0.125, 'gini = 0.142\nsamples = 13\nvalue = [1, 12, 0]'),
         Text(0.875, 0.125, 'gini = 0.0\nsamples = 53\nvalue = [53, 0, 0]')]
```



Das linke Blatt in unserem Entscheidungsbaum enthält nur eine Beobachtung. Wir können davon ausgehen, dass diese Beobachtung ein sogenannter Ausreißer ist. Als *Ausreißer* oder *Outlier* bezeichnet man Beobachtungen, deren Eigenschaften sich von den Eigenschaften

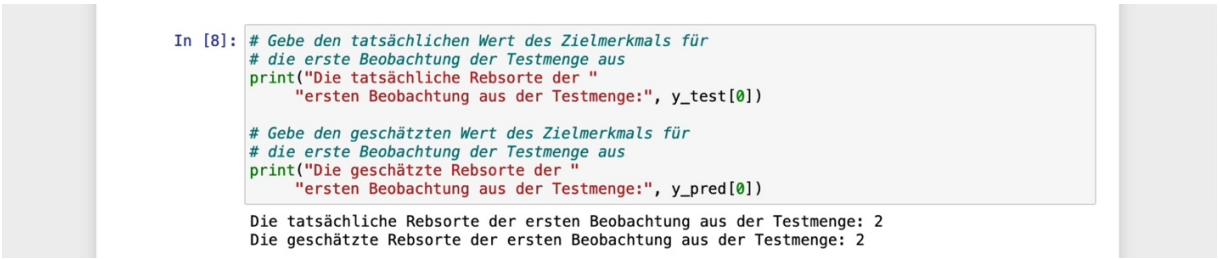
anderer Beobachtungen stark unterscheiden. Bei dieser Beobachtung können wir zumindest davon ausgehen, dass es sich um einen Ausreißer innerhalb seiner Klasse handelt.

In der Abbildung kannst du außerdem sehen, dass das dritte Blatt von links Beobachtungen unterschiedlicher Klassen enthält. Das ist an dieser Stelle gar nicht schlimm und sogar gewollt, weil nur eine Beobachtung zu einer anderen Klasse gehört. Wahrscheinlich handelt es sich wieder um einen Ausreißer. Würden wir diesen Knoten weiter aufsplitten, würden wir dadurch unser Entscheidungsbaummodell überanpassen, was wir eigentlich vermeiden wollen.

Um sehen zu können, wie gut das Entscheidungsbaummodell die Klasse für neue Beobachtungen prognostizieren kann, geben wir exemplarisch für die erste Beobachtung der Testmenge die tatsächliche und die durch das Klassifikationsmodell geschätzte Klasse bzw. Rebsorte aus.

```
# Gebe den tatsächlichen Wert des Zielmerkmals für
# die erste Beobachtung der Testmenge aus
print("Die tatsächliche Rebsorte der "
      "ersten Beobachtung aus der Testmenge:", y_test[0])

# Gebe den geschätzten Wert des Zielmerkmals für
# die erste Beobachtung der Testmenge aus
print("Der geschätzte Rebsorte der "
      "ersten Beobachtung aus der Testmenge:", y_pred[0])
```

In der Ausgabe sehen wir, dass die durch das Klassifikationsmodell prognostizierte Klasse mit der tatsächlichen Klasse für diese Beobachtung übereinstimmt.



Du kannst diese Werte auch für die anderen Beobachtungen der Testmenge ausgeben und vergleichen. Um dir etwas Arbeit dabei zu ersparen, erstellen wir eine Variable für den Index der Beobachtung und geben ihren Namen an den entsprechenden Stellen ein.

```
# Die Variable "index" enthält den Index der für uns interessanten
# Beobachtung der Testmenge
index = 0

# Gebe den tatsächlichen Wert des Zielmerkmals für
# eine Beobachtung der Testmenge aus
print("Die tatsächliche Rebsorte der "
      "ersten Beobachtung aus der Testmenge:", y_test[index])

# Gebe den geschätzten Wert des Zielmerkmals für
# eine Beobachtung der Testmenge aus
```

```
print("Die geschätzte Rebsorte der "
      "ersten Beobachtung aus der Testmenge:", y_pred[index])
```

Beim Ausprobieren musst du nur beachten, dass die Testmenge nicht so viele Beobachtungen enthält und du irgendwann einen *IndexError* bekommen kannst.

Mit der Funktion *accuracy_score()* des Untermoduls *Metrics* von *Scikit-learn* berechnen wir die *Accuracy* für unser Entscheidungsbaummodell anhand der tatsächlichen und der prognostizierten Klassen für die Beobachtungen der Testmenge.

Quelle [6]

```
# Berechne die Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

Wie du in der Ausgabe sehen kannst, erreicht unser Klassifikationsmodell einen Accuracy-Wert von 0.94, d. h. für 94 % der Beobachtungen wurde die Klasse richtig prognostiziert. Das ist kein schlechter Wert für diesen Datensatz.

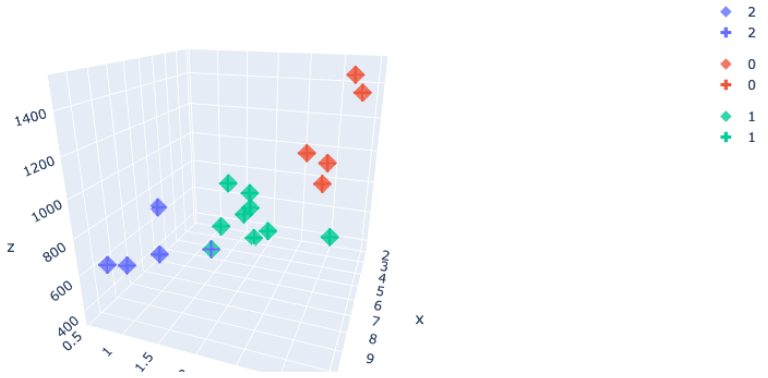
```
In [10]: # Berechne die Accuracy
         accuracy = accuracy_score(y_test, y_pred)
         print(accuracy)

         0.9444444444444444
```

In einem 3-dimensionalen Diagramm stellen wir die Beobachtungen der Testmenge grafisch dar. Dafür verwenden wir dieses Mal die Merkmale, die zum Aufspalten der inneren Knoten im Entscheidungsbaum vom Algorithmus gewählt wurden. Das sind in unserem Fall die Farbtiefe, der Flavonoidgehalt und der Prolingehalt. Die Werte dieser drei Merkmale tragen am meisten dazu bei, die Beobachtungen entsprechend ihrer Klassenzugehörigkeit zu unterscheiden. Die Rebsorten der Beobachtungen markieren wir durch unterschiedliche Farben. Dabei stellen wir die tatsächlichen Klassen der Beobachtungen durch Diamanten und die prognostizierten durch Kreuze dar.

```
# Visualisiere die tatsächliche Zugehörigkeit als Diamant
# und die geschätzte als Kreuz
fig3 = px.scatter_3d(x=X_test[:,9], y=X_test[:,6], z=X_test[:,12],
                     color=y_test, opacity=0.8,
symbol_sequence=["diamond"])
fig4 = px.scatter_3d(x=X_test[:,9], y=X_test[:,6], z=X_test[:,12],
                     color=y_pred, symbol_sequence=["cross"])
fig5 = go.Figure(data=fig3.data + fig4.data)
fig5.show()
```

Im Diagramm kannst du sehen, welche Beobachtungen richtig und welche falsch klassifiziert wurden. Bei den richtig klassifizierten Beobachtungen stimmen die Farben der Diamanten mit den Farben der Kreuze überein. Bei den falsch klassifizierten Beobachtungen sind Diamanten und Kreuze unterschiedlich eingefärbt. Es wurde nur ein Wein der falschen Rebsorte zugeordnet. Wenn man sich die Entscheidungsregeln genauer ansieht, erkennt man, dass diese Beobachtung nur anhand der Farbtiefe direkt der zweiten Klasse

zugeordnet wurde. An diesem Beispiel erkennt man den Unterschied zum Beispiel zum *k*-nächste-Nachbarn-Verfahren, das bei der Klassenzuordnung der Beobachtungen Werte aller Merkmale in Betracht zieht.



Take-Home Message

In diesem Video hast du gelernt, wie man ein Entscheidungsbaum-Klassifikationsmodell in Python erstellt und mit seiner Hilfe die Klassenzugehörigkeit für neue Beobachtungen prognostiziert. Außerdem weißt du jetzt, wie man die Überanpassung eines Entscheidungsbaummodells an die Trainingsmenge vermeiden und wie man mögliche Ausreißer anhand eines Entscheidungsbaums erkennen kann. Wenn du unterschiedliche Entscheidungsbaummodelle mit verschiedenen Parametern auf demselben Datensatz trainierst, wirst du feststellen, dass deine Modelle auf diesem Datensatz unterschiedlich gute Ergebnisse bei der Klassifikation neuer Beobachtungen erzielen. Möchtest du wissen, wie du die Entscheidungsbäume kombinieren kannst, um die besten Ergebnisse zu erreichen, schau dir das nächste Video an. Dort wirst du erfahren, wie du ein *Random Forest*-Modell in Python trainieren und zur Klassifikation der Beobachtungen benutzen kannst.

Quellen

Quelle [1] Forina, M., Leardi, R., Armanino, C., & Lanteri, S. (1988). *PARVUS - An Extendible Package for Data Exploration, Classification and Correlation.* Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy.

Quelle [2] Scikit-learn (2022). Datasets. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html

Quelle [3] Scikit-learn (2022). Model Selection. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Quelle [4] Scikit-learn (2022). Decision Trees. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Quelle [5] Scikit-learn (2022). Decision Trees. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html

Quelle [6] Scikit-learn (2022). Metrics. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Weiterführendes Material

Nguyen, C. N., & Zeigermann, O. (2021). *Machine Learning -- kurz & gut: Eine Einführung mit Python, Pandas und Scikit-Learn*. O'Reilly

Disclaimer

Transkript zu dem Video „Woche 08: Programmierung – Entscheidungsbaumklassifikation mit Scikit-learn", Ludmila Himmelspach.