Woche 08: Programmierung – Random Forest-Klassifikation mit Scikit-learn

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Ein Random Forest-Klassifikationsmodell für einen gegebenen Datensatz erstellen können
- Parameter des Random Forest-Modells, die einen großen Einfluss auf die Klassifikationsgüte haben, benennen können
- Ein trainiertes Random Forest-Modell nutzen können, um für Beobachtungen der Testmenge die Klasse zu prognostizieren
- Die Güte eines Random Forest-Modells mit Accuracy bewerten können

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

Inhalt

Einstieg

Einen *Random Forest* kannst du dir als eine Vielzahl von Entscheidungsbäumen vorstellen, die anhand verschiedener Teilmengen der Trainingsdaten trainiert werden. Dabei werden beim Training einzelner Entscheidungsbäume nur einige Merkmale bzw. Features berücksichtigt, die zufällig ausgewählt werden. Die Klassenzuweisung für neue Beobachtungen erfolgt durch eine Mehrheitsentscheidung über die Vorhersagen der einzelnen Bäume. D. h. eine neue Beobachtung wird der Klasse zugewiesen, die von den meisten Entscheidungsbäumen prognostiziert wurde.

Vorbereitung des Datensatzes

Wie im letzten Video arbeiten wir wieder mit dem *Wine Recognition Dataset*, das Ergebnisse einer chemischen Analyse von 178 Weinen von drei verschiedenen Rebsorten enthält. Wie du dich sicherlich erinnern kannst, repräsentieren die Merkmalswerte im Datensatz die in den Weinen gemessenen Inhaltsstoffmengen. 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.

```
# Lade die nötigen Module
from sklearn import ensemble, 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
```

Wie im letzten Video laden wir den *Wine*-Datensatz mit der Funktion *load_wine()*, und teilen diesen mit der Funktion *train_test_split()* in die Trainings- und die Testmenge auf.

Quelle [2]
Quelle [3]

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

# Teile den Datesatz 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)
```

Um ein *Random Forest*-Modell zu konstruieren, müssen wir nicht selbst einzelne Entscheidungsbäume erstellen und ihre Prognosen für neue Beobachtungen von Hand auswerten. Dafür bietet das Untermodul *Ensemble Methods* von *Scikit-learn* die Funktion *RandomForestClassifier()* an. Das Random Forest-Modell hat einen entscheidenden Vorteil gegenüber dem Entscheidungsbaummodell: Es ist weniger anfällig für eine Überanpassung an die Trainingsmenge. Da einzelne Entscheidungsbäume in einem Random Forest auf der Grundlage verschiedener Teilmengen der Trainingsdaten erstellt werden, beeinflussen kleine Abweichungen in den Trainingsdaten nur wenige Entscheidungsbäume des Random Forests. Die fehlerhafte Prognostizierung wird in diesem Fall durch eine Mehrheitsentscheidung über die Prognosen anderer Bäume ausgeglichen. Deswegen ist der einzige wichtiger Parameter, um den man sich beim Erstellen eines Random Forest-Modells wirklich kümmern muss, die Anzahl der Entscheidungsbäume im Random Forest. Dabei gilt: Je höher die Anzahl der Entscheidungsbäume im Random Forest, desto zuverlässiger sind die Voraussagen, die er liefert. Hier muss man allerdings zwischen der erforderlichen Rechenleistung und der Zuverlässigkeit der Ergebnisse abwägen.

Die Anzahl der Entscheidungsbäume im Random Forest wird durch den Parameter *n_estimators* in der Funktion *RandomForestClassifier()* festgelegt. Dieser Parameter ist mit 100 voreingestellt. Da die Anzahl der Beobachtungen in unserem Datensatz relativ klein ist und ein einziger Entscheidungsbaum relativ gute Ergebnisse geliefert hat, bleiben wir bei dieser Voreinstellung.

Quelle [4]

Neben der Anzahl der Entscheidungsbäume können mit unterschiedlichen Parametern auch einige Eigenschaften für alle Entscheidungsbäume im *Random Forest* vorbestimmt werden. Wir gehen auf diese Parameter aber nicht noch einmal ein, weil sie mit den Parametern für das Entscheidungsbaummodell übereinstimmen. Zwei Parameter könnten sich bei der Erstellung eines Random Forest-Modells als wichtig erweisen. Zum einen ist das der Parameter *bootstrap*, der darauf Einfluss nimmt, ob die einzelnen Entscheidungsbäume auf der Basis der Teilmengen oder der gesamten Trainingsmenge erstellt werden. In der Voreinstellung ist festgelegt, dass die Bäume auf der Grundlage verschiedener Teilmengen der Trainingsdaten erstellt werden. Wenn die Trainingsmenge aber nur wenige Beobachtungen enthält, weil der Datensatz an sich nicht besonders groß ist, ist es sinnvoll, für die Erstellung der einzelnen Entscheidungsbäume die Beobachtungen der gesamten Trainingsmenge heranzuziehen. In diesem Fall werden sich die Entscheidungsbäume aufgrund verschiedener Merkmale voneinander unterscheiden, die für die Konstruktion der Bäume ausgewählt wurden.

Der zweite Parameter ist *random_state,* der nicht nur die Wahl der Teilmengen für den Aufbau der Entscheidungsbäume regelt, sondern auch die Wahl der Merkmale bzw. Features für die Aufspaltung der inneren Knoten im Baum steuert. Wir gehen hier aus Komplexitätsgründen nicht näher darauf ein, wie dies funktioniert. An dieser Stelle ist nur wichtig zu wissen, dass die Zuweisung einer ganzen Zahl für den Parameter *random_state* zu

denselben Ergebnissen bei verschiedenen Aufrufen der Funktion *RandomForestClassifier()* führt. Die beliebten zufälligen Zahlen für den Parameter *random_state* sind 0 und 42. Wir weisen ihm den Wert 42 zu.

Mit der Funktion *fit()* trainieren wir unser Random Forest-Modell mit den Beobachtungen der Trainingsmenge.

```
# Erstelle das Random Forest-Modell
rf_classifier = ensemble.RandomForestClassifier(random_state=42)

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

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

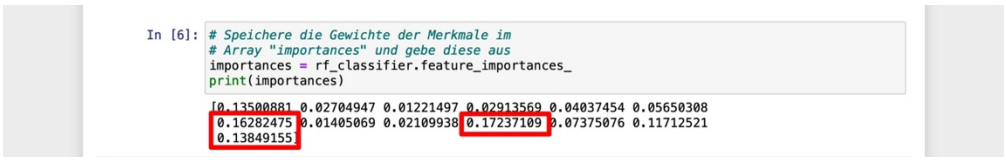
```
# Erstelle die Vorhersage für die Testmenge
y_pred = rf_classifier.predict(X_test)
```

Das Random Forest-Modell lässt sich zwar nicht so einfach interpretieren wie ein Entscheidungsbaummodell, du kannst jedoch mit der Anweisung *feature_importances_* die Gewichte der einzelnen Merkmale bzw. Features ausgeben lassen. Diese Gewichte sind maßgebend dafür, wie viel ein Merkmal zur Verringerung der Verunreinigung der Klassen beiträgt. Je höher der Wert ist, desto wichtiger ist das entsprechende Merkmal für die Bestimmung der Klassenzugehörigkeit der Beobachtungen.

Wir speichern die Gewichte der Merkmale im Array *importances* und geben diese aus.

```
# Speichere die Gewichte der Merkmale im
# Array "importances" und gebe diese aus
importances = rf_classifier.feature_importances_
print(importances)
```

In der Ausgabe kannst du sehen, dass das Gewicht für das zehnte Merkmal am höchsten ist. Das bedeutet, das zehnte Merkmal trägt am meisten zur relativen Verringerung der Verunreinigung der Klassen bei. Den zweit- und den dritthöchsten Wert haben das siebte und das dreizehnte Merkmal. Bei diesen drei wichtigsten Merkmalen handelt es sich um die Farbtiefe, den Flavonoidgehalt und den Prolingehalt. Genau diese Merkmale wurden auch im Entscheidungsbaummodell im letzten Video als die wichtigsten Merkmale erkannt.



Wir verzichten auf den Vergleich der tatsächlichen und der durch das Klassifikationsmodell geschätzten Klasse für eine einzelne Beobachtung der Testmenge. Stattdessen berechnen wir mit der Funktion *accuracy_score()* des Untermoduls *Metrics* von *Scikit-learn* die

Accuracy für unser Random Forest-Modell anhand der tatsächlichen und der prognostizierten Klassen für die Beobachtungen der gesamten Testmenge.

Quelle [5]

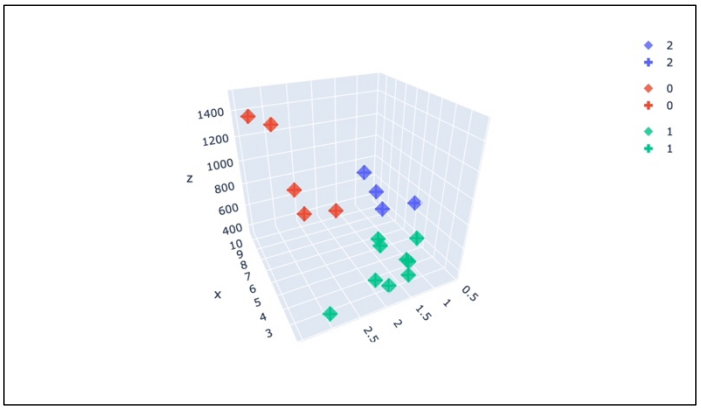
```
# Berechne die Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

Wie du in der Ausgabe sehen kannst, erreicht unser Klassifikationsmodell einen Accuracy-Wert von 1.0, d. h. für alle Beobachtungen aus der Testmenge wurde die Klasse richtig prognostiziert. Im Vergleich zum Entscheidungsbaum aus dem letzten Video haben wir eine Steigerung der Klassifikationsgüte für diesen Datensatz erzielt.

Um unseren Erfolg zu veranschaulichen, stellen wir die Beobachtungen der Testmenge in einem 3-dimensionalen Diagramm grafisch dar. Dafür verwenden wir wieder drei Merkmale, die für die Bestimmung der Klassenzugehörigkeit der Beobachtungen von Random Forest-Modell als die wichtigsten eingestuft wurden. Das sind in unserem Fall wieder die Farbtiefe, der Flavonoidgehalt und der Prolingehalt. 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, dass alle Beobachtungen der Testmenge richtig klassifiziert wurden. Das erkennst du daran, dass die Farben der Diamanten mit den Farben der Kreuze übereinstimmen. Das ist aber keine Überraschung, weil wir das bereits am Wert der Accuracy erkennen konnten.



Take-Home Message

In diesem Video hast du gelernt, wie man ein Random Forest-Klassifikationsmodell in Python erstellt und welche Parameter des Modells einen entscheidenden Einfluss auf die Klassifikationsgüte haben. Außerdem hast du an einem konkreten Datensatz gesehen, dass der Einsatz des Random Forest-Modells einen Vorteil gegenüber einem Entscheidungsbaummodell hat.

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). Ensemble Methods. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Quelle [5] 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.