Woche 09: Programmierung – *k*-means Clustering mit Scikit-learn

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Ein *k-means* Clustering-Modell in Python erstellen und mit seiner Hilfe einen Datensatz in Cluster aufteilen können
- Erklären können, warum eine Normalisierung der Merkmalswerte eines Datensatzes vor dem Clustering wichtig ist
- Die Merkmalswerte eines Datensatzes auf einen Bereich zwischen 0 und 1 normalisieren können

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

Inhalt

Einstieg

Bei unüberwachten Machine Learning Verfahren geht es darum, die Struktur der Daten zu erforschen und nach unbekannten Mustern zu suchen, um sinnvolle und nützliche Informationen daraus zu extrahieren. Eines der am häufigsten benutzten unüberwachten Datenanalyseverfahren ist *Clustering*, dessen Ziel es ist, ähnliche Beobachtungen in Gruppen bzw. *Cluster* einzuteilen. In diesem Video lernst du, wie man einen Datensatz mit Hilfe des *k-means* Clustering-Algorithmus in Cluster zerlegt.

Vorbereitung des Datensatzes

Der *k-means* Clustering-Algorithmus gehört zu den sogenannten *partitionierenden* Verfahren. Das bedeutet, dass er jede Beobachtung eines Datensatzes einem der Cluster zuteilt. Da wir beim Clustering noch keine Informationen über die Struktur der Daten besitzen – denn diese sollen wir ja durch das Clustering herausfinden – können wir nach der Aufteilung des Datensatzes in Cluster erstmal nicht sagen, ob unsere Zerlegung auf irgendeine Weise sinnvoll oder nützlich ist. Deswegen arbeiten wir in diesem Video mit einem Datensatz, dessen Struktur durch die Labels der Beobachtungen bereits kenntlich gemacht wurde. Nach der Aufteilung des Datensatzes mit dem *k-means* Clustering-Algorithmus, können wir die Zuordnung der Beobachtungen zu den Clustern mit den originalen Klassenlabels vergleichen, um bewerten zu können, ob die gefundenen Cluster sinnvoll sind. Das entspricht natürlich nicht dem üblichen Vorgehen und wird hier nur zur Verdeutlichung der Funktionsweise der Clustering-Methode so gemacht.

Zuerst aber importieren wir alle Module, die wir in unserem Programm brauchen werden.

```
# Importiere die nötigen Module
from sklearn import cluster, preprocessing
import seaborn as sns
import numpy as np
import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objects as go
```

In diesem Video arbeiten wir mit dem *Palmer Archipelago (Antarctica) Penguin* Datensatz. Dieser Datensatz enthält Informationen über 344 Pinguine, die auf drei Inseln des Palmer-Archipels in der Antarktis beheimatet sind. Die Merkmale bzw. Features des Datensatzes enthalten verschiedene Messdaten, die über die Pinguine gesammelt wurden, wie Länge und Höhe des Schnabels, Länge der Flosse und die Körpermasse bzw. das Gewicht. Außerdem enthält das Merkmal *species* die Pingiunart der Pinguine. Dabei gehören die Beobachtungen bzw. die Pinguine im Datensatz zu einer der drei Arten, *Adelie*, *Chinstrap* (deutsch: Zügelpinguin) und *Gentoo* (deutsch: Eselspinguin).

Quelle [1]

CC BY

Die Pinguinart wurde von den Forschenden festgestellt, die Pinguine untersucht haben. Wir tun jetzt so, als würden wir nichts über die Zuordnung der Beobachtungen wissen. Also spielen wir das Szenario durch, in dem Forschende auf einen Haufen ihnen bisher unbekannte Pinguine getroffen sind. Sie haben den Schnabel und die Flossen jedes Pinguins gemessen und die Pinguine gewogen. Diese Daten haben sie an uns übergeben und wollen von uns wissen, ob man anhand der Merkmalswerte im Datensatz feststellen kann, ob es Gruppen von Beobachtungen, die auf den Ähnlichkeiten der Merkmalswerte basieren, im Datensatz gibt und wenn ja, welche. Aus dieser Analyse würden sie dann darauf schließen können, ob die ihnen begegneten Pinguine zu einer oder zu unterschiedlichen Pinguinarten gehören.

Also laden wir den Datensatz und speichern ihn in der Variable *penguins*. Der Datensatz wird in einer Datenstruktur namens *DataFrame* gespeichert. Daher wandeln wir die Datenstruktur *DataFrame*, in der unser Datensatz momentan vorliegt, in die Datenstruktur *ndarray* um. Dafür verwenden wir die Funktion *to_numpy().*

Quelle [2]

```
# Lade den Palmer Archipelago (Antarctica) penguin Datensatz
penguins = sns.load_dataset("penguins")

# Wandle das DataFrame in ein NumPy-Array um
penguins_arr = penguins.to_numpy()
```

Wie bereits erwähnt, konzentrieren wir uns für die weitere Analyse nur auf die Merkmale *Schnabellänge*, *Schnabelhöhe*, *Flossenlänge* und *Körpermasse*. Also wählen wir diese aus dem Array *penguins_arr* aus und speichern ihre Werte im *ndarray X_raw* ab. Die erste Spalte des Arrays *penguins_arr* enthält die entsprechenden Pinguinarten. Für den späteren Vergleich wählen wir dieses Merkmal aus und speichern die Pinguinarten im Zielarray *y_raw* ab.

```
# Speichere das dritte (Schnabellänge), das vierte (Schnabelhöhe),
# das fünfte (Flossenlänge) und das sechste (Körpermasse)
# Merkmal in der Datenmatrix
X_raw = np.array(penguins_arr[:,2:6]).astype(float)

# Speichere das erste Merkmal als Zielmerkmal
y_raw = penguins_arr[:,0]
```

Bei zwei Beobachtungen im Datensatz fehlen einige Werte. Da man zwischen diesen und den vollständigen Beobachtungen keine Distanzen berechnen kann, bekommt der Clustering-Algorithmus Probleme. Deswegen entfernen wir die unvollständigen Beobachtungen aus dem Datensatz. Mach dir keine Sorgen, wenn du die Befehle dafür nicht verstehst. Das würde an dieser Stelle zu weit führen.

```
# Lösche alle Zeilen aus der Datenmatrix, die
# fehlende Wert enthalten
X = X_raw[~np.isnan(X_raw).any(axis=1)]
y = y_raw[~np.isnan(X_raw).any(axis=1)]
```


CC BY

Nach Bereinigung verbleiben im Datensatz noch 342 Beobachtungen.

Um sich ein besseres Bild über die eigentliche Verteilung der Beobachtungen im Datensatz zu machen, visualisieren wir sie in einem Diagramm. Da es schwierig ist, die Datenpunkte durch alle vier Merkmale grafisch darzustellen, beschränken wir uns auf die Darstellung der Beobachtungen durch die Merkmale *Schnabellänge*, *Schnabelhöhe* und *Flossenlänge* in einem 3-dimensionalen Diagramm. Dabei markieren wir die Zugehörigkeit zur Pinguinart durch unterschiedliche Farben der Datenpunkte. Die Adelie-Pinguine werden durch blaue, die Chinstrap-Pinguine durch rote und die Gentoo-Pinguine durch grüne Punkte dargestellt.

```
pyo.init_notebook_mode()
fig_orig = px.scatter_3d(x=X[:,0], y=X[:,1], z=X[:,2], color=y)
fig_orig.show()
```

Im Diagramm sehen wir, dass die Adelie-Pinguine zwar durchschnittlich einen längeren Schnabel als die Chinstrap-Pinguine haben, aber keine klare Trennung zwischen diesen beiden Pinguinarten anhand der drei gewählten Merkmale zu beobachten ist. Dagegen lassen sich die Gentoo-Pinguine durch eine längere Flosse von den anderen Arten besser trennen. Mal schauen, ob der *k-means* Clustering-Algorithmus in der Lage sein wird, die Beobachtungen so in die Cluster aufzuteilen, dass diese den tatsächlichen drei Pinguinarten in etwa entsprechen.

Wir haben gesehen, wie die eigentliche Verteilung der Beobachtungen im Datensatz aussieht. Wir gehen aber davon aus, dass uns die Zuordnung der Pinguine zu unterschiedlichen Pinguinarten nicht bekannt ist und wir diese noch erforschen müssen. Also würde die Visualisierung unserer Ausgangsdaten in einem 3-dimensionalen Diagramm wie folgt aussehen:

```
fig1 = px.scatter_3d(x=X[:,0], y=X[:,1], z=X[:,2])
fig1.show()
```

In der grafischen Darstellung kann man bereits erkennen, dass es mindestens zwei Gruppen von Beobachtungen im Datensatz gibt, die in drei Merkmalen ähnliche Werte aufweisen. So sieht die Verteilung der Beobachtungen aus, nur wenn man die Merkmale *Schnabellänge*, *Schnabelhöhe* und *Flossenlänge* betrachtet. Wenn man die Beobachtungen durch die Werte anderer Merkmale visualisiert, kann man vielleicht auch andere Cluster erkennen. Deswegen berücksichtigt der *k-means* Clustering-Algorithmus bei der Zerlegung eines Datensatzes in Cluster die Werte aller Merkmale.

k-means Clustering-Algorithmus

Der *k-means* Clustering-Algorithmus teilt alle Beobachtungen eines Datensatzes in *k* Cluster auf. Die Anzahl der Cluster ist aber in der Regel vorher nicht bekannt. Es gibt zwar einige Verfahren, wie diese bestimmt bzw. geschätzt werden kann, ihre Erklärung würde aber an dieser Stelle zu weit führen. Deswegen nehmen wir *k* = 3 für die Anzahl der Cluster im Pinguin-Datensatz an.

Ein *k-means* Clustering-Modell wird mit der Funktion *KMeans()* des *Scikit-learn*-Untermoduls *Clustering* erstellt. Mit dem Parameter *n_clusters* wird die Anzahl der Cluster festgelegt, in die die Beobachtungen eines Datensatzes zerlegt werden. Wir wollen die Beobachtungen des Pinguin-Datensatzes in drei Cluster aufteilen, also weisen wir dem Parameter *n_clusters* eine 3 zu. In der Regel werden die Mittelwerte bzw. die Zentren der Cluster am Anfang mit zufällig aus dem Datensatz ausgewählten Beobachtungen initialisiert. In den Voreinstellungen der Funktion *KMeans()* ist aber eine andere ausgeklügeltere, aber aufwändigere Initialisierungsmethode festgelegt. Durch die Zuweisung des Wertes *random* für den Parameter *init* stellen wir sicher, dass die *k* (hier 3) zufällig aus dem Datensatz ausgewählten Beobachtungen als anfängliche Clustermittelwerte fungieren. Der Parameter *random_state* bestimmt die entsprechende Zufallszahlengenerierung für die Wahl der anfänglichen Clusterzentren. Um die Reproduzierbarkeit unserer Clusteringergebnisse zu garantieren, weisen wir ihm einen Wert, z. B. 42 zu. Mit dem Parameter *n_init* wird die Anzahl der Durchläufe des *k-means* Algorithmus mit jeweils unterschiedlichen zufällig ausgewählten Sart-Clusterzentren bestimmt. Wir weisen *n_init* den Wert 10 zu. Das heißt, dass der *k-means* Algorithmus 10 Mal mit jeweils unterschiedlichen anfänglichen Clusterzentren gestartet wird. Am Ende erhalten wir eine Aufteilung in die Cluster, bei der die Beobachtungen am nächsten an den Zentren ihrer Cluster liegen. Die Funktion *KMeans()* hat auch weitere Parameter, mit denen das Clustering-Modell exakter spezifiziert werden kann. Nähere Informationen zu diesen findest du in der API Refernce von Scikit-learn.

Quelle [3]

Mit der Funktion *fit()* berechnen wir die Aufteilung der Beobachtungen unseres Datensatzes in Cluster mit dem erstellten Clustering-Modell.

```
# Erstelle das Clustering-Modell für k = 3
k_means = cluster.KMeans(n_clusters=3, init="random",
                         random_state=42, n_init=10)

# Berechne das Clustering für den Datensatz X
k_means.fit(X)
```

Wir stellen die Aufteilung der Beobachtungen in Cluster in einem 3-dimensionalen Diagramm dar. Dafür verwenden wir wieder nur die ersten drei Merkmale: die Schnabellänge, die Schnabelhöhe und die Flossenlänge. Wir markieren die Clusterzuweisungen der Beobachtungen durch unterschiedliche Farben. Dabei können die Labels bzw. die Clusterzuweisungen der Beobachtungen mit der Anweisung *labels_* ausgegeben werden. Da die Clusterlabels mit ganzen Zahlen angegeben werden, wandeln

CC BY

wir sie in Zeichenketten um, damit sie als kategoriale Merkmalswerte von der *scatter_3d()*-Funktion wahrgenommen werden.

```
# Plotte das Clusteringergebnis
fig2 = px.scatter_3d(x=X[:,0], y=X[:,1], z=X[:,2],
                     color=k_means.labels_.astype(str))
fig2.show()
```

Im Diagramm kannst du sehen, dass unser Clusteringergebnis nicht ganz unseren Vorstellungen entspricht. Irgendwie ergeben die Clusterzuweisungen einiger Beobachtungen überhaupt keinen Sinn. Einige *Gentoo*-Pinguine wurden anderen Pinguinarten zugeordnet, obwohl sie in den dargestellten drei Merkmalen ähnlichere Werte zu den Werten anderer *Gentoo*-Pinguine als zu Pinguinen anderer Arten aufweisen. Aber woran liegt das? Diese Frage können wir beantworten, indem wir die Beobachtungen anhand anderer Merkmale darstellen. Nehmen wir dafür die Merkmale Schnabellänge, Schnabelhöhe und Gewicht.

```
# Plotte das Clusteringergebnis
fig3 = px.scatter_3d(x=X[:,0], y=X[:,1], z=X[:,3],
                     color=k_means.labels_.astype(str))
fig3.show()
```

Wenn du dir die Zuweisungen der Beobachtungen zu Clustern im Diagramm genauer anschaust, stellst du fest, dass die Aufteilung der Beobachtungen in Clustern hauptsächlich auf Grund des Merkmals Gewicht stattgefunden hat. Das liegt daran, dass der Algorithmus bei der Berechnung der Distanzen zwischen den Beobachtungen die Bedeutung der Maßeinheiten, in denen die Merkmalswerte der Beobachtungen angegeben wurden, außer Acht lässt. So weist der Algorithmus zum Beispiel dem Unterschied von 30 mm bei der Schnabellänge die gleiche Bedeutung wie dem Unterschied von 30 g beim Gewicht zu.

Wenn wir die Spannweiten der Werte einzelner Merkmale bzw. Features berechnen, sehen wir, dass diese für verschiedene Merkmale sehr unterschiedlich sind. So liegt der Unterschied zwischen dem kleinsten und dem größten Wert für das Merkmal *Gewicht* bei 3600, während die Spannweite der Werte für das Merkmal *Schnabelhöhe* nur 8.4 beträgt. Deswegen ist es nicht überraschend, dass das Merkmal *Gewicht* bei der Berechnung der Distanzen zwischen den Beobachtungen dominiert.

```
# Gebe die Spannweiten der Werte einzelner
# Merkmale des Datensatzes aus
print(np.max(X, axis=0)-np.min(X, axis=0))
```

Um allen Merkmalen die gleiche Bedeutung bei der Distanzberechnung zuzuweisen, sollten die Werte aller Merkmale vor dem Clustering am besten normalisiert werden. Bei der *Normalisierung* handelt es sich um eine Methode, bei der die Werte numerischer Merkmale bzw. Features so geändert werden, dass sie die gleiche Größenordnung haben. Dabei sollten die Unterschiede in den Wertebereichen nicht verzerrt werden. Meistens werden die Merkmalswerte auf das Intervall [0, 1] normalisiert.

CC BY

Das *Scikit-learn*-Untermodul *Preprocessing and Normalization* bietet eine Funktion namens *minmax_scale()* an, mit deren Hilfe die Werte jedes Merkmals im Datensatz auf einen vorgegebenen Bereich normalisiert werden können. Da der Wertebereich mit 0 bis 1 in der Funktion voreingestellt ist, brauchen wir nichts weiter zu tun, als das Datenarray der Funktion zu übergeben. Um die originalen Werte des Datenarrays nicht zu überschreiben, speichern wir den normalisierten Datensatz im *ndarray X_normalized* ab.

Quelle [4]

```
# Normalisiere die Merkmalswerte auf den Bereich [0,1]
X_normalized = preprocessing.minmax_scale(X)
```

Wenn wir jetzt die Spannweiten der Werte einzelner Merkmale im normalisierten Datensatz berechnen, [PAUSE] sehen wir, dass diese für alle Merkmale gleich Eins sind.

```
# Gebe die Spannweite der normalisierten Werte einzelner
# Merkmale des Datensatzes aus
print(np.max(X_normalized, axis=0)-np.min(X_normalized, axis=0))
```

Wir berechnen die Aufteilung der Beobachtungen des normalisierten Datensatzes in Cluster mit dem Clustering-Modell, das wir zuvor erstellt haben.

```
# Berechne das Clustering für
# den Datensatz X_normalized
k_means.fit(X_normalized)
```

Wir stellen die Aufteilung der Beobachtungen in Cluster in einem 3-dimensionalen Diagramm grafisch dar. Dafür verwenden wir wieder nur die ersten drei Merkmale: die Schnabellänge, die Schnabelhöhe und die Flossenlänge. [PAUSE] Wir markieren die Clusterzuweisungen der Beobachtungen durch unterschiedliche Farben. Dabei stellen wir die tatsächlichen Klassen der Beobachtungen durch Kreise und die von *k-means* Algorithmus berechneten Clusterzuweisungen durch Kreuze dar.

```
# Visualisiere die tatsächliche Klassenzugehörigkeit der
# Beobachtungen als Kreis und die von k-means berechnete
# Clusterzugehörigkeit als Kreuz
fig4 = px.scatter_3d(x=X[:,0], y=X[:,1], z=X[:,2],
                     color=y, opacity=0.8,
                     symbol_sequence=["circle"])
fig5 = px.scatter_3d(x=X[:, 0], y=X[:, 1], z=X[:, 2],
                     color=k_means.labels_.astype(str),
                     symbol_sequence=["cross"])
fig6 = go.Figure(data=fig4.data + fig5.data)
fig6.show()
```

Im Diagramm kannst du sehen, dass die Zuordnung der Beobachtungen zu den Clustern den tatsächlichen Klassenlabels sehr nahekommt. Das erkennst du daran, dass die Farben der Kreise mit den Farben der Kreuze für die meisten Beobachtungen übereinstimmen. Natürlich wurden einige Adelie-Pinguine der Chinstrap-Art zugeordnet, aber im Wesentlichen wurden drei Gruppen bzw. Cluster der Beobachtungen, die in vier zur

Verfügung stehenden Merkmalen ähnliche Werte aufweisen, vom *k-means* Algorithmus richtig aufgedeckt. Denn darum geht es beim Clustering. Die Zuweisung der Beobachtungen zu den richtigen Klassen ist die Aufgabe der Klassifikation.

Take-Home Message

In diesem Video hast du gelernt, wie du in Python ein *k-means* Clustering-Modell erstellen und mit seiner Hilfe die Beobachtungen eines Datensatzes in Cluster aufteilen kannst. Außerdem hast du gesehen, warum es so wichtig ist, eine Normalisierung der Merkmalswerte des Datensatzes vor dem Clustering durchzuführen. Da du jetzt auch weißt, wie du einen Datensatz in Python normalisieren kannst, entgeht dir kein Cluster auf deiner Cluster-Entdeckungstour.

Quellen

Quelle [1] Horst, A. M., Hill, A. P., & Gorman, K. B. (2020). *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*. *R package version 0.1.0.* https://allisonhorst.github.io/palmerpenguins/ https://doi.org/10.5281/zenodo.3960218

Quelle [2] NumFOCUS, Inc. (2022). DataFrame. In *pandas API Reference, Release 1.5.2* https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_numpy.html

Quelle [3] Scikit-learn (2022). Clustering. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Quelle [4] Scikit-learn (2022). Preprocessing and Normalization. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax_scale.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 09: Programmierung – *k*-means mit Scikit-learn“, Ludmila Himmelspach.

Dieses Transkript wurde im Rahmen des Projekts ai4all des Heine Center for Artificial Intelligence and Data Science (HeiCAD) an der Heinrich-Heine-Universität Düsseldorf unter der Creative Commons Lizenz CC-BY 4.0 veröffentlicht. Ausgenommen von der Lizenz sind die verwendeten Logos, alle in den Quellen ausgewiesenen Fremdmaterialien sowie alle als Quellen gekennzeichneten Elemente.