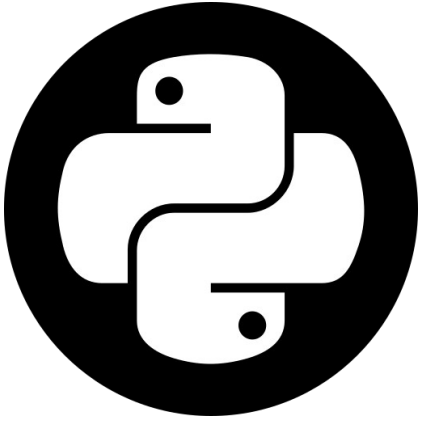
DATA SCIENCE

MIT PYTHON



GRUNDLAGEN UND TOOLS

Willkommen

- Ziele:
  - Einstieg in Data Science mit Python
  - Grundlagen für die Verarbeitung, Analyse und Interpretation großer Datenmengen schaffen
  - Data Science als Prozess verstehen
  - Tools und Methoden kennenlernen

Data Science

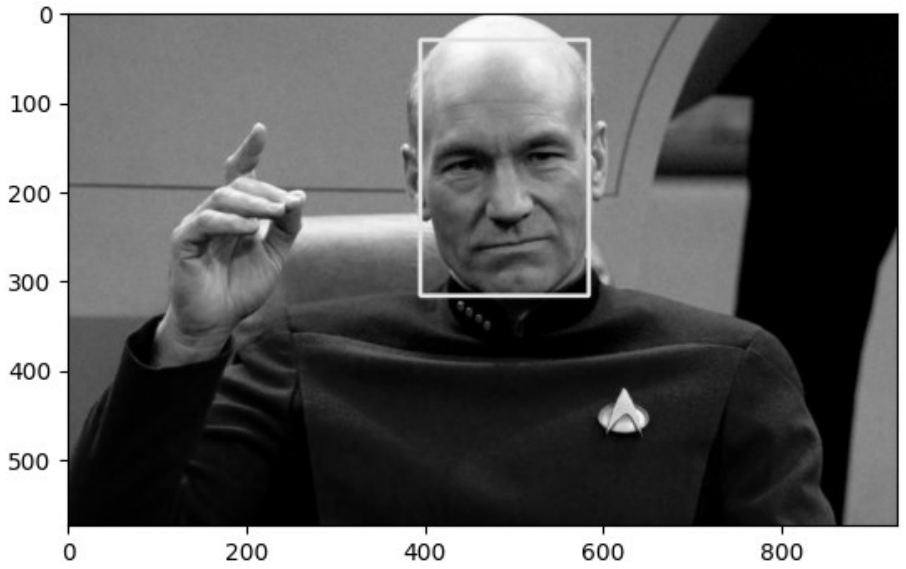
- Erkenntnisse aus Daten über Prozess:
  - Datenerfassung (und -bereinigung)
  - Datenverarbeitung
  - Datenanalyse
  - Datenvisualisierung

Data Science

- Verschiedene Technologien und Methoden (Statistik, Machine Learning, etc.)
- Identifikation von Mustern und Trends als mögliche Ziele
- Hilfreich für Entscheidungsfindungen

Data Science

```
plt.imshow(image)
plt.show()
```



(b2)

Big Data

- Große und komplexe Datensätze:
  - strukturiert oder unstrukturiert
  - unterschiedliche Formate
- Beispiele aus der Versicherungsbranche:
  - Verhaltensmuster von Kundinnen und Kunden
  - Wahrscheinlichkeiten von Schäden und Verlusten

Big Data

- Beispiele aus dem Gesundheitswesen:
  - Wirksamkeitsanalysen von Arzneimitteln
  - Sequenzierung des menschlichen Genoms
  - Anomaliedetektion und individuelle Behandlungen

Big Data (Exkurs)

Lektüre:

Wie wird Big Data in diesem Beitrag charakterisiert?

Python

- Interpretierte und objektorientierte Programmiersprache:
  - klare, gut strukturierte Syntax
  - keine Kompilierung erforderlich
  - Unabhängig von Plattformen
  - Vielzahl von Bibliotheken

Python

- Vorteile:
  - einfach zu erlernen
  - automatisierte Skripte
  - Vielzahl an Anwendungsmöglichkeiten
  - große Community, viele Dokumentationen

Datenstrukturen in Python

- Datenstrukturen dienen der Speicherung und Verarbeitung von Daten
- Grundlegende Datenstrukturen (1):
  - Liste [ ]: Veränderliche Sammlung von Elementen
  - Tupel ( ): wie Liste, aber unveränderlich
  - Dictionary { }: Schlüssel-Wert-Paare

Datenstrukturen in Python

- Grundlegende Datenstrukturen (2):
  - Sets { }: Einzigartiger Elemente, ungeordnet
  - Array [ ]: Elemente desselben Typs, operabel
  - DataFrame { }: tabellarische Datenstruktur
  - Stack [ ]: lineare Datenstruktur

Datenstrukturen in Python

```
# Tuples (I)
neuer_tuple = (1, 2, 3, "female", "male")
neuer_tuple
```

```
(1, 2, 3, 'female', 'male')
```

```
# Tuples (II)
neuer_tuple * 2
```

```
(1, 2, 3, 'female', 'male', 1, 2, 3, 'female', 'male')
```

```
# Stacks (I)
neues_stack = [1, 2, 3]
neues_stack.append(4)
neues_stack
```

```
[1, 2, 3, 4]
```

```
# Stacks (II)
neues_stack.pop()
neues_stack
```

```
[1, 2, 3]
```

Jupyter Notebooks

- Interaktive Entwicklungsumgebung:
  - verschiedene Programmiersprachen
  - sofortiger Ergebniszugriff
  - einfache Dokumentation
  - Wiederverwendbarkeit
  - Open Source

Docker

- Container zur Distribution:
  - einfache Implementierung ermöglicht Skalierung
  - vollständige Entwicklungs- und Testumgebung
  - mit allen Abhängigkeiten und Bibliotheken
  - anwendungsübergreifend

Kubernetes

- Automatisierte Bereitstellung von Containern:
  - hohe Skalierbarkeit
  - komfortable Verwaltung
  - sehr gute Integration von Python
  - Multi-Container-Multi-Server-Architektur

GitHub

- Versionskontrollsystem:
  - kollaborative Projektarbeiten möglich
  - commit, push, pull, etc.
- Vorteile:
  - paralleles Programmieren
  - Backups und somit Testmöglichkeiten

CRISP-DM

- Cross Industry Standard Process for Data Mining:
  - Business- und Datenverständnis
  - Datenvorbereitung und Modellierung
  - Bewertung und Implementierung

CRISP-DM (Exkurs)

Lektüre:

Wie lässt sich der CRISP-DM Prozess adaptieren?

PYTHON

BIBLIOTHEKEN

NumPy

- Bibliothek für numerische Berechnungen:
  - viele Funktionen und Operationen
  - geeignet für große Datenmengen

Pandas

- Bibliothek zur Datenmanipulation:
  - wichtig für bspw. Series und DataFrames
  - demnach für ein- bzw. zweidimensionale Daten
- Funktionen (Auszug):
  - Datenimport und -export
  - filtern, sortieren, transformieren, etc.

SciKit-Learn

- Bibliothek für maschinelles Lernen:
  - Vielzahl an Algorithmen
  - Datenvorverarbeitung und Modellauswahl
  - überwachte und unüberwachte Verfahren
  - Tools zur Modellbewertung

Matplotlib

- Bibliothek für Grafiken und Diagramme:
  - verwendet Python-Objekte
  - mehrebenentauglich
  - hohe Flexibilität

GRUNDLAGEN STATISTIK

Data Mining und Data Crawling

- Data Mining:
  - Analyse von Daten zur Informationsfreilegung
  - Algorithmen ermöglichen die Analyse
- Data Crawling:
  - automatische Extraktion von Daten
  - bspw. Websites, öffentlich zugängliche Quellen, etc.

Stichprobe und Grundgesamtheit

- Grundgesamtheit:
  - Gesamtheit aller Elemente
  - bspw. alle Menschen in Deutschland
- Stichprobe:
  - Teilmenge der Grundgesamtheit
  - bspw. die Sonntagsfrage zur Bundestagswahl

Konfidenzniveau und -intervall

- Das Konfidenzniveau ist die Wahrscheinlichkeit, mit der ein geschätzter Parameter in einem bestimmten Intervall liegt
- Ein Konfidenzintervall bei einem Konfidenz-niveau von 95% bedeutet bspw., dass das Intervall in 95% der Fälle den wahren Wert des Parameters enthalten könnte

Fehlermarge

- Differenz zwischen den oberen und unteren Grenzen des Konfidenzintervalls
- u.a. abhängig von der Stichprobengröße
- Interpretationsmöglichkeiten:
  - breitere Fehlermarge: mehr Ungenauigkeit
  - schmalere Fehlermarge: weniger Ungenauigkeit

Fehlermarge (Exkurs)

Lektüre:

Welchen Einfluss haben Stichprobengröße und Konfidenzniveau auf die Fehlermarge?

Modus

- Häufigste Merkmalsausprägung einer Variable
- Anwendbar bei nominalskalierten Daten

$$\bar{x}_d = Häufigster\ Beobachtungswert$$

Median

- Der Wert in der Mitte zwischen Minimum und Maximum von ordinal- bzw. intervallskalierten Variablen

$$\tilde{x}_{ungerade} = x_{\frac{n+1}{2}} \quad \text{bzw.} \quad \tilde{x}_{gerade} = \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right)$$

Arithmetisches Mittel

- Summenwert aller Merkmalsausprägungen verhältnisskalierter Variablen, dividiert durch die Anzahl der Fälle

$$\bar{x} = \frac{1}{n}(x_1 + x_2 + x_3 + \cdots + x_n)$$

Varianz

- Statistisches Maß für die Streuung von Datenpunkten um das arithmetische Mittel

$$s_x{}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Kovarianz

- Bei der Kovarianz wird davon ausgegangen, dass die Streuung einer Variable X Einfluss auf die Streuung einer Variable Y nimmt

$$\hat{\sigma}_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Standardabweichung

- Standardisiertes Maß für die Streuung von Datenpunkten um den Mittelwert
- Wichtig für viele Formeln und Algorithmen

$$s_x = \sqrt{{s_x}^2}$$

Standardabweichung

- Ermöglicht bspw. z-Transformation

```
# z-Transformation der ersten unabhängigen Variable
x = df.iloc[:,[0]]
scaled_x = scale.fit_transform(x)
```

```
# Mittelwert der standardisierten Variable
mean_scaled_x = np.mean(scaled_x)
round(mean_scaled_x)
```

```
0
```

```
# Standardabweichung der standardisierten Variable
sd_scaled_x = np.std(scaled_x)
round(sd_scaled_x)
```

```
1
```

Korrelation

- Im Vergleich zur Kovarianz ein standardisiertes Zusammenhangsmaß für die Variablen X und Y
- Misst die Linearität des Zusammenhangs

$$r_{xy} = \frac{\hat{\sigma}_{xy}}{s_x * s_y} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} * \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Korrelation (Exkurs)

Lektüre:

Welcher Korrelationskoeffizient lässt sich aus der Linearität der Zusammenhänge ableiten?

Lineare Regression

- Ermöglicht bei linearen Zusammenhängen die Vorhersage der Variable Y über die Mekmals-ausprägungen der Variable X

$$a = \bar{y} - b_1 \bar{x}_1 \qquad \text{mit:} \qquad b_1 = r_{xy} * \frac{s_y}{s_x}$$

bei Bestimmtheitsmaß: $R^2 = (r_{xy})^2$

Lineare Regression

```
# Multivariate Statistik (I)
y = np.array(df.Y)
x = df.iloc[:,[0,1]]
```

```
# Multivariate Statistik (II)
model = LinearRegression()
model.fit(x, y)
```

```
# Multivariate Statistik (III)
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('coefficients:', model.coef_)
```

```
coefficient of determination: 0.9479500377816745
intercept: -57.98765891838086
coefficients: [4.7081605  0.33925123]
```

Logistische Regression

- Bei einer nominalskalierten Variable ist ein lineares Vorhersagemodell ungeeignet
- Vereinfacht ausgedrückt geht es um die Wahrscheinlichkeit (p) für Y = 1
- Die logistische Regression ist eine häufige Aktiverungsfunktion bei neuronalen Netzen

Logistische Regression

- Die nominalskalierte abhängige Variable wird in der logistischen Regression über logarithmierte Odds Ratios bestimmt
- Dazu ein Beispiel: Die logarithmierte Odds Ratio ein nerdiger Star Wars Fan zu sein

Logistische Regression

- 7 von 10 Nerds sind Star Wars Fans
- 4 von 10 Normalos sind Star Wars Fans
  - 7 / 10 = 70% als Wahrscheinlichkeit für Nerds
  - 4 / 10 = 40% als Wahrscheinlichkeit für Normalos
  - 70% / (1 – 70%) = 2,33 als Odds für Nerds
  - 40% / (1 – 40%) = 0,66 als Odds für Normalos

Logistische Regression

- 2,33 als Odd (Nerd) und 0,66 (kein Nerd):
  - 2,33 / 0,66 = 3,5 als Odds Ratio
  - ln(3,5) = 1,25 als logarithmierte Odds Ratio
- Dieser Wert wird auch Logit genannt
- Logit = ln(Odds Ratio)

Interaktionseffekte

- Der Effekt einer Variable X auf eine Variable Y hängt von einer weiteren unabhängigen Variable ab
- Ein Interaktionsterm ist dabei das Produkt zweier unabhängiger Variablen, der Regressionsanalysen komplementieren kann

Interaktionseffekte

```
# Bivariate Statistik (I)
df.corr()
```

	Y	X1	X2	X3	X4	X5
Y	1.000000	0.353079	-0.645883	-0.663789	0.463685	0.416556
X1	0.353079	1.000000	-0.686542	-0.639523	0.401095	-0.060859
X2	-0.645883	-0.686542	1.000000	0.698415	-0.572742	-0.114022
X3	-0.663789	-0.639523	0.698415	1.000000	-0.153859	-0.099322
X4	0.463685	0.401095	-0.572742	-0.153859	1.000000	0.175496
X5	0.416556	-0.060859	-0.114022	-0.099322	0.175496	1.000000

```
# Multivariate Statistik (IV)
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('coefficients:', model.coef_)
```

```
coefficient of determination: 0.7067350015927254
intercept: 66.91518167896871
coefficients: [-0.17211397 -0.25800824 -0.87094006  0.10411533  1.07704814]
```

(3)

Bootstrapping

- Ausgehend von einer Stichprobe wird eine Vielzahl an künstlichen Stichproben erstellt
- Prinzip: Ziehen mit Zurücklegen
- Anwendbar bei unbekannter Verteilung der Grundgesamtheit zur Ermittlung vom Konfidenzintervallen der relevanten Parameter

Signifikanz

- Vereinfacht ausgedrückt: Grad der Gewissheit, dass ein Zusammenhang zwischen den Variablen X und Y nicht nur zufällig ist
- Bei statistischen Testverfahren wird gegen die sogenannte Nullhypothese getestet
- Signifikanzniveaus sind kritisch zu reflektieren

Signifikanz (Exkurs)

Lektüre:

Was sind die Researcher degrees of freedom und welchen Einfluss können diese auf das Signifikanz-niveau nehmen?

GRUNDLAGEN

MACHINE /

DEEP LEARNING

Maschinelles Lernen

- Teilgebiet der Künstlichen Intelligenz
- Muster in Daten erkennen, um bspw. Vorhersagen zu ermöglichen
- Verbreitete Anwendungsbeispiele:
  - Spracherkennung
  - Bilderkennung

Maschinelles Lernen

```
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```
Epoch 1/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2186 - accuracy: 0.9359
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0955 - accuracy: 0.9706
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0682 - accuracy: 0.9783
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0533 - accuracy: 0.9830
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0418 - accuracy: 0.9863
313/313 [==============================] - 0s 553us/step - loss: 0.0632 - accuracy: 0.9809
[0.06316164135932922, 0.98089998960495]
```

```
example = x_train[1701]
pyplot.imshow(example.reshape(28, 28), cmap="gray")
```

```
output_predict = model.predict(example.reshape(1, 28, 28, 1))
print("Predicted figure:")
np.argmax(output_predict)
```

```
1/1 [==============================] - 0s 44ms/step
Predicted figure:
0
```

(b1)

Maschinelles Lernen

- Auszug aus dem zugrundeliegenden Prozess:
  - Daten sammeln, bereinigen, transformieren, aufbereiten
  - Modell auswählen, trainieren, validieren
  - Vorhersagen treffen, ggfls. optimieren

Maschinelles Lernen

```
# Manuelle Identifikation
identification_group_a_df = df.loc[(df.X3 <= 1.9) & (df.X4 <= 0.6)]
print('correct:', identification_group_a_df.Y.count(), '/ 50')

##############################
### SIEHE AUFGABENSTELLUNG ###
##############################
```

```
correct: 50 / 50
```

(4)

Maschinelles Lernen (Exkurs)

Lektüre:

Welche Fehler treten häufig beim maschinellen Lernen auf?

Trainingsdatensatz

- Aufteilung der Daten in:
  - Trainingsdatensatz
  - Validierungsdatensatz
- Trainingsdatensatz als ausreichend große Stichprobe im Umfang von 70% bzw. 80%
- Modell trainiert mit dem Trainingsdatensatz

Validierungsdatensatz

- Ermöglicht die Bewertung der Performance
- Der Validierungsdatensatz sollte nicht zum Training des Modells verwendet werden (!)
- Übliche Größe: 30% bzw. 20%

Metriken zur Evaluierung

- Klassifikationsmodelle zur Vorhersage von nominalskalierten Variablen können bspw. über verschiedene Metriken evaluiert werden
- Dabei wird die Leistung des Modells in Bezug auf die Anzahl an korrekten und falschen Vorhersagen ausgewiesen

Metriken zur Evaluierung

- Gängige Metriken sind:
  - Accuracy als Genauigkeit
  - Precision als Präzision
  - Recall als Sensitivität
  - F1-Score basiert auf Precision und Recall

Metriken zur Evaluierung

```
# Trainingsdatensatz (80%) und Testdatensatz (20%) einteilen
res = train_test_split(x, y,
    train_size=0.8,
    test_size=0.2,
    random_state=12)
train_data, test_data, train_labels, test_labels = res
```

```
# K-Nearest-Neighbor
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)
print("Predictions from the classifier:")
knn_predicted_data = knn.predict(test_data)
print(knn_predicted_data)
print("Target values:")
print(test_labels)
```

```
Predictions from the classifier:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 1 0 1 2 1 0 2 1 1 0 0]
Target values:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 1 0 1 2 1 0 2 1 1 0 0]
```

```
# K-Nearest-Neighbor Accuracy
print(accuracy_score(knn_predicted_data, test_labels))
```

```
1.0
```

(5)

Deep Learning

- Teilgebiet des maschinellen Lernens
- Basiert auf künstlichen neuronalen Netzen (KNN)
- Vereinfachte Darstellung der Funktionsweise:
  - Eingabedaten werden durch Schichten geleitet
  - Aktivierungsfunktionen beeinflussen die Weiterleitung
  - Anpassungen bzw. Optimierungen erfolgen iterativ

Deep Learning (Exkurs)

Lektüre:

Welche Bedeutung hat das Gradient Decent Verfahren für Künstliche neuronale Netze?

Batch / Epoch(e)

- Batch:
  - Anteil der Trainingsdaten, die gleichzeitig vom künstlichen neuronalen Netz verarbeitet werden
  - diese vorher festgelegte Aufteilung in Anteile / Gruppen ermöglicht die Anpassung der Gewichte
  - die Gewichte weisen dabei eine Analogie zur Regressionsanalyse auf

Batch / Epoch(e)

- Epoch(e):
  - vollständiger Durchlauf aller Trainingsdaten
  - die vorher festgelegte Anzahl an Durchläufen (Epochen) kann das Gradient Decent Verfahren maßgeblich beeinflussen (!)
- Batch-Verarbeitung beschleunigt KNNs

Aktivierungsfunktionen

- Die Schichten der künstlichen neuronalen Netze beinhalten Neuronen
- Deren Funktionsweise hängt von den zugrundeliegenden Aktivierungsfunktionen ab
- Aktivierungsfunktionen bestimmen maßgeblich die Weitergabe von Daten zu einer Schicht

Aktivierungsfunktionen

- Häufige Aktivierungsfunktionen:
  - Sigmoid, bekannt von der logistischen Regression
  - ReLU, Weiterentwicklung von Sigmoid
  - Tanh, symmetrische Variante von Sigmoid
  - Softmax, hilfreich bei nicht-binären Klassifikationen

Aktivierungsfunktionen

```
# Neuronales Netz spezifizieren
model = Sequential()
model.add(Dense(16, input_dim=3, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['binary_accuracy'])
model.fit(training_data, target_data, epochs=15)
scores = model.evaluate(training_data, target_data)
```

```
# Performance des neuronalen Netzes
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print(model.predict(training_data).round())
```

```
binary_accuracy: 75.00%
1/1 [==============================] - 0s 37ms/step
[[1.]
 [1.]
 [1.]
 [0.]]
```

AUTOMATISIERTE TEXTANALYSE

Transformer

- Ermöglichen die Verarbeitung von Sequenzen
- Elemente einer Sequenz haben Bedeutung
- Ermöglicht Beziehungen zwischen Elementen

Transformer

- Methodische Grundlage des modernen Natural Language Processings (NLP)
- Funktionsweise: Multi-Head-Attention-Modul
- Ermöglicht parallele Verarbeitung von Eingabedaten, bspw. Anfang, Mitte und Ende eines Satzes

Transformer (Exkurs)

Lektüre:

Wie profitiert die automatisierte Sprachübersetzung vom Multi-Head-Attention-Modul?

Natural Language Processing

- Verarbeitung und Analyse von natürlicher Sprache
- Dafür muss Sprache in ein maschinen-lesbares Format überführt werden

Natural Language Processing

- Grundlagen des NLP-Prozesses:
  - Tokenisierung, bspw. Unterteilung von Wörtern
  - Morphologische Analyse, bspw. Wortart
  - Syntaxanalyse, bspw. Struktur des Textes
  - Semantische Analyse, bspw. Kontextualisierung
  - Diskursanalyse, bspw. Gesamtzusammenhang

Natural Language Processing

```
line = input('> ')
word = line.strip().split(' ')[-1]
if word not in lexicon:
    print('Sorry...')
else:
    options = lexicon[word]
    predicted = np.random.choice(list(options))
    print(predicted)
```

```
>  This workshop was
good
```

```
list(options.keys())
```

```
['good', 'bad', 'ugly']
```

(b3)

Sentimentanalyse

- Analyse von Stimmungen und Emotionen
- Vereinfacht Customer-Feedback-Analysen
- Grundlegende Sentiments:
  - positiv
  - neutral
  - negativ

Sentimentanalyse (Exkurs)

Lektüre:

Lassen sich über Sentimentanalysen Börsendynamiken abbilen?

ALGORITHMEN

Clusteranalyse

- Clusteranalyse als ein mögliches ML-Verfahren
- Für jedes Verfahren stehen mehrere Algorithmen zur Verfügung stehen
- Bei der Clusteranalyse als Verfahren geht es um die Gruppierung von Objekten in homogene Untergruppen

Clusteranalyse

- Prozess der Clusteranalyse:
  - Daten- und Variablenauswahl sowie -aufbereitung
  - Analyse mittels Distanz- oder Ähnlichkeitsmaße
  - Auswahl geeigneter Algorithmen
  - Ergebnisinterpretation
  - Clusteridentifikation

Clusteranalyse

- Kontext: Unsupervised Machine Learning
- Gängige Algorithmen:
  - K-Means, Random Forest, Mean Shift, etc.
  - Agglomeratives Clustering, etc.
  - BIRCH, DBSCAN, etc.

Clusteranalyse

```
# Datensatz für Plot simulieren
X, y = make_blobs(n_samples=500, centers=4, cluster_std=1.75, random_state=7)
```

```
# Simulierten Datensatz plotten
for class_value in range(4):
    row_ix = where(y == class_value)
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```

```
# Agglomeratives Clustering Algorithmus
model = AgglomerativeClustering(n_clusters=4)
yhat = model.fit_predict(X)
clusters = unique(yhat)
```

```
# Clusterlösung visualisieren
for cluster in clusters:
    row_ix = where(yhat == cluster)
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```

Clusteranalyse (Exkurs)

Lektüre:

Wie unterscheiden sich die Ziele der Algorithmen bei den Verfahren Classification, Regression, Clustering und Dimensionality Reduction?

KNN - Algorithmus

- K-Nearest Neighbors (KNN)
- Klassifikationsalgorithmus (!)
- Kontext: Supervised Machine Learning
- Ermöglicht Vorhersage von Labeln (Klassen)
- Tipp: Ergebnisvergleich mit K-Means

K-Means - Algorithmus

- Algorithmus beginnt mit zufälligen Zentren
- Analysiert nahegelegene Datenpunkte hinsichtlich Homogenität
- Iteration der Zentren

Random Forest - Algorithmus

- Clustering über mehrere Entscheidungsbäume
- Basiert auf Durchschnittswert der Vorhersagen
- Vor- und Nachteile:
  - Bedingt resistent gegenüber Overfitting (!)
  - Funktioniert auch bei fehlenden Werten
  - rechen- und zeitintensiv

Agglomeratives Clustering

- Bottom-up-Clustering, d.h. jeder Datenpunkt stellt initial ein eigenes Cluster dar
- Datenpunkte werden sukzessive zusammengefasst
- Ergebnis: Dendrogramm

BIRCH - Algorithmus

- Balanced Iterative Reducing and Clustering using Hierachies (BIRCH)
- Optimierung für große Datenmengen
- Über Schwellenwerte werden Datenpunkte zu Clustern zusammengefasst

DBSCAN - Algorithmus

- Density Based Spatial Clustering of Application with Noise (DBSCAN)
- Ermöglicht Ausreißer-Detektion
- Optimiert für unterschiedliche Clusterormen und -größen

Spectral Clustering - Algorithmus

- Basiert auf Spektraltheorie der Graphen
- Struktur der Graphen entspricht Datenstruktur
- Datenpunkte sind als Knoten repräsentiert
- Hintergrund: K-Means und Eigenwerte

Mean Shift - Algorithmus

- Analyse der Dichte der Datenpunkte
- Verschiebung von Fensterbereichen / Frames
- Iteration bis höchste Dichte im Zentrum
- Zusammenfassung der Konvergenzpunkte

Gaussian Mixture - Algorithmus

- Probabilistisches Clustering
- Datenpunkte: Mischung normalverteilter Cluster
- Basiert auf Maximum-Likelihood-Methode
- Optimiert für höherdimensionale Daten

VERTIEFUNGEN

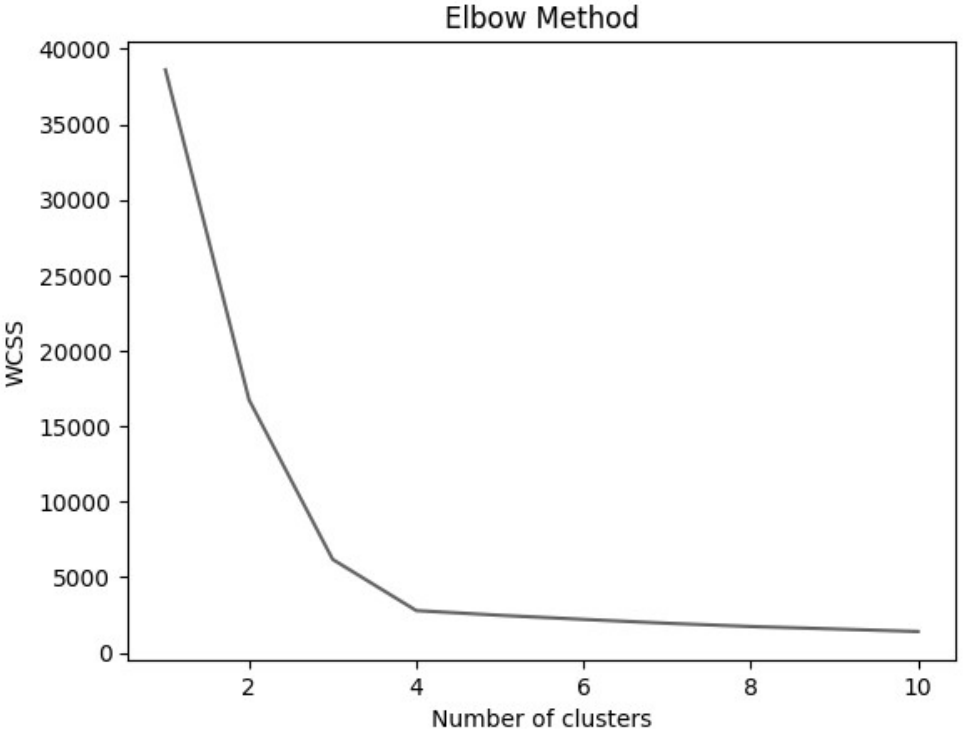
Maximum-Likelihood-Methode

- Statistisches Verfahren zur Schätzung der Parameter einer Wahrscheinlichkeitsverteilung
- Annahme: Beobachtete Daten stammen aus einer bestimmten Wahrscheinlichkeitsverteilung
- Gesucht sind die Parameter mit der größten Wahrscheinlichkeit für die beobachteten Daten

Within Cluster Sum of Squares

- Bewertung der Qualität von Clustering-Ergebnissen
- Basiert auf der Summe der quadratischen Abweichungen jedes Punktes innerhalb eines Clusters von seinem Zentrum

Within Cluster Sum of Squares



(8)

Faktorenanalyse

- Identifikation von zugrunde liegenden Variablen (Faktoren)
- Verfahren zur Dimensionsreduktion
- Basiert ebenfalls auf Analyse der Eigenwerte
- Hilfreich bei explorativen Datenanalysen

Eigenwerte

- Beziehen sich auf die charakteristischen Werte einer Matrix
- Ein Eigenwert gibt an, wie eine Matrix auf einen Vektor wirkt
- Hilfreich bei Varianzanalyse


The End