

3.5 KI & Machine Learning

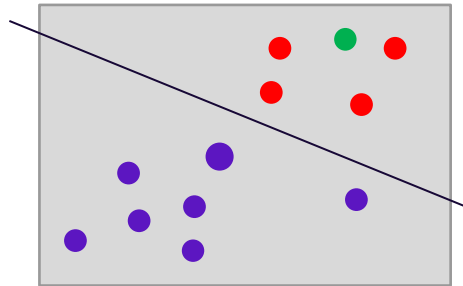
3

Klassifikation: K Nearest Neighbor

Klassifikation/Clustering: Welches sind die besten Kunden?

Einteilung in 2 Gruppen

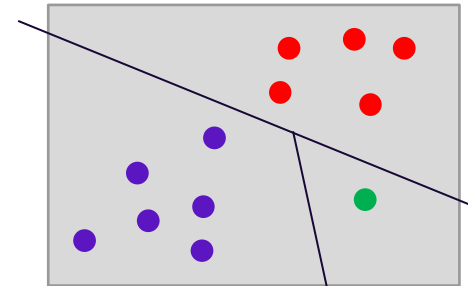
Häufigkeit
Einkauf



Warenkorbgröße

Einteilung in 3 Gruppen

Häufigkeit
Einkauf



Warenkorbgröße

Die Idee des **k-nearest neighbor** Algorithmus ist es, einen Datenpunkt der Klasse zuzuordnen, der auch die k Nachbarn angehören

- Gehört zu der Klasse des überwachten Lernens
- Meistens für Klassifikation, es gibt auch Variante für Regression
- Ggf. gewichtete Abstandsfunktion bei ungleich verteilten Trainingsdaten

- Das Package scikit-learn bietet im Untermodul neighbors mehrere Nearest Neighbor Algorithmen.
- Der KNN-Klassifikations-Algorithmus ist in der Funktion *sklearn.neighbors.KNeighborsClassifier()* implementiert

```
import sklearn.neighbors as skn

knn = skn.KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

vorhersage = knn.predict(X_test)
# Güte (accuracy)
accuracy = knn.score(X_test, y_test)
```

Wie lässt sich die Güte der Klassifikation bestimmen?





Wie lässt sich die Güte der Klassifikation bestimmen?
→ Vergleich der Vorhersage mit den echten Werten

Confusion Matrix
(Wahrheitsmatrix)



		Vorhersage	
		Ja	Nein
Wirklichkeit	Ja	Richtig Positiv (true positive)	Falsch Negativ (false negative, Fehler 2.Art)
	Nein	Falsch Positiv (false positive, Fehler 1.Art)	Richtig Negativ (true negative)

Der Schaden kann sich für Fehler 1. und 2. Art erheblich unterscheiden, z.B. bei Krankheitsdiagnosen (Test erkennt Krankheit nicht oder Test meldet fälschlicherweise Krankheit)

		Vorhersage	
		Ja	Nein
Wirklichkeit	Ja		
	Nein		

$$\begin{aligned}\text{Treffergenauigkeit} &= \frac{\text{Richtig Positiv} + \text{Richtig Negativ}}{\text{Gesamtanzahl}} \\ (\text{Accuracy}) & \\ &= \frac{a + d}{a + b + c + d}\end{aligned}$$

$$\begin{aligned}\text{Sensitivität} &= \frac{\text{Richtig Positiv}}{\text{Richtig Positiv} + \text{Falsch Negativ}} \\ (\text{Sensitivity, Recall}) & \\ &= \frac{a}{a + b}\end{aligned}$$

		Vorhersage	
		Ja	Nein
Wirklichkeit	Ja	a	b
	Nein	c	d

		Vorhersage	
		Ja	Nein
Wirklichkeit	Ja	a	b
	Nein	c	d

$$\begin{aligned}\text{Spezifität} &= \frac{\text{Richtig Negativ}}{\text{Falsch Positiv} + \text{Richtig Negativ}} \\ (\text{specificity}) &= \frac{d}{c + d}\end{aligned}$$

$$\begin{aligned}\text{Präzision} &= \frac{\text{Richtig Positiv}}{\text{Richtig Positiv} + \text{Falsch Positiv}} \\ (\text{Precision}) &= \frac{a}{a + c}\end{aligned}$$

		Vorhersage	
		Ja	Nein
Wirklichkeit	Ja	a	b
	Nein	c	d

		Vorhersage	
		Ja	Nein
Wirklichkeit	Ja	a	b
	Nein	c	d

Beispiel: HIV in Deutschland

Angenommen, die Sensitivität und die Spezifität für einen HIV-Test lägen bei 99,9%.

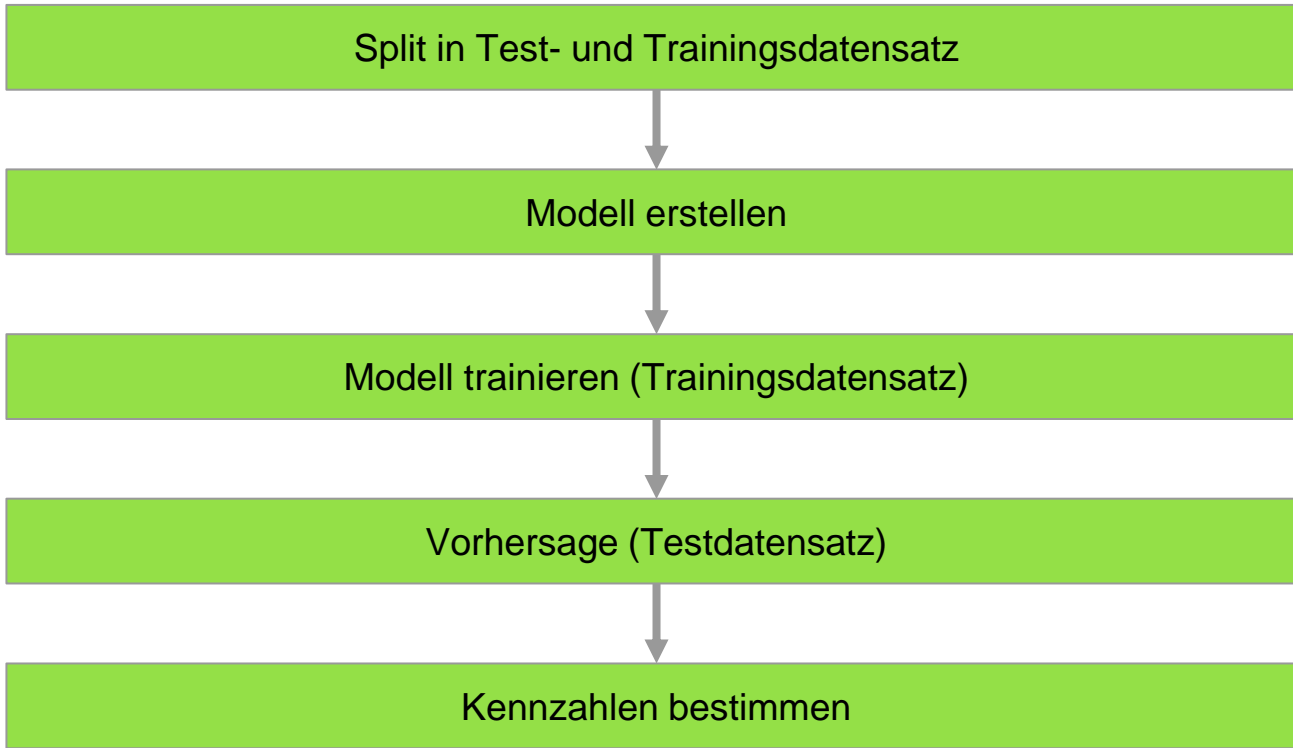
In Deutschland leben etwa 90.000 Menschen mit HIV-Infektion. Was passiert, wenn man die ganze Bevölkerung testen würde?

		Vorhersage		
		Ja	Nein	
Wirklichkeit	Ja	89.910	90	90.000
	Nein	81.910	81.828.090	81.910.000
		171.820	81.828.180	82.000.000

Das **F1-Maß** (F1-Score) ist das harmonische Mittel von **Precision** (Präzision) und **Recall** (Sensitivität)

$$F_1 = 2 \frac{P * R}{P + R}$$

Das F1-Maß ist besser als die accuracy geeignet, die Güte widerzuspiegeln, wenn die Klassen unbalanciert sind, d.h. eine Klasse deutlich mehr Elemente besitzt



1. Split in Trainings- und Testdatensatz

```
X_train, X_test, y_train, y_test =  
sklearn.model_selection.train_test_split(X, y, test_size=0.2)
```

2. Modell erzeugen, z.B.

```
knn = sklearn.neighbors.KNeighborsClassifier(n_neighbors=5)
```

3. Modell anhand des Trainingsdatensatzes trainieren

```
knn.fit(X_train, y_train)
```

4. Vorhersage

```
vorhersage = knn.predict(X_test)
```

5. Kennzahlen, z.B. Confusion Matrix und Güte-Maße

```
sklearn.metrics.confusion_matrix(y_test, vorhersage)  
sklearn.metrics.classification_report(y_test, vorhersage)  
sklearn.metrics.accuracy_score(y_test, vorhersage)
```