

Duale Hochschule Baden-Württemberg Mannheim

**Hausarbeit**

**K-Nearest-Neighbour Algorithmus**

**Studiengang Wirtschaftsinformatik**

Studienrichtung Data Science

|  |  |
| --- | --- |
| Verfasser/in: | Lukas Benner (6550912), Hanna Siegmann (6710117) |
| Firma: | Würth IT GmbH |
| Kurs: | WWI19DSC |
| Vorlesung: | Algorithmen und Datenstrukturen |
| Dozent: | Herr Voker Harms volker.harms@outlook.com |
| Bearbeitungszeitraum: | 19.05.2020 – 06.07.2020 |

**Inhaltsverzeichnis**

[**1 Theoretische Funktionsweise** **1**](#_Toc9857)

[1.1 Heranführung ans Thema 1](#_Toc9858)

[1.2 Algorithmus 1](#_Toc9859)

[1.2.1 Funktionsweise 1](#_Toc9860)

[1.2.2 Abstandsmaße 2](#_Toc9861)

[1.2.3 Parameter k 3](#_Toc9862)

[1.2.4 Lazy Learning 3](#_Toc9863)

[**2 Anwedung des KNN** **5**](#_Toc9864)

[2.1 Datenvorbereitung 5](#_Toc9865)

[2.2 Anwendung KNN mit Sklearn 6](#_Toc9866)

[2.3 Eigenentwicklung des KNN Algorithmus 7](#_Toc9867)

[**3 Fazit** **10**](#_Toc9868)

[**Literaturverzeichnis** **11**](#_Toc9869)

# Theoretische Funktionsweise

## Heranführung ans Thema

In einer datengetriebenen Welt ist es von essenzieller Bedeutung Daten einordnen und klassifizieren zu können. Dabei helfen verschiedene Algorithmen, welchen oftmals menschlichen Denkweisen zugrunde liegen. Eine davon ist das Prinzip der Ähnlichkeit. Wir Menschen neigen dazu Dinge, die ähnliche Merkmale besitzen gleiche Funktionen zuzuschreiben. Wissen wir zum Beispiel, dass ein bestimmter Pilz giftig ist, werden wir in Zukunft Pilze meiden, die ein ähnliches Aussehen aufweisen.

Die vorliegende Hausarbeit setzt sich mit dem k-nearest-neighbor Algorithmus auseinander. Wie der Name schon impliziert, verwendet er die nächsten Nachbarn, um Daten einzuordnen und Vorhersagen machen zu können. Dabei spielt die Frage, wie man Ähnlichkeiten in messbare Größen umwandeln kann eine bedeutete Rolle.

Das Ziel der Arbeit ist es den Aufbau und sowie die Funktionsweise des Machine learning Algorithmus zu verstehen. Des Weiteren wird ein selbst programmierter KNNAlgorithmus implementiert und eine Anwendungssituation simuliert.

Dabei wird es zunächst in Kapitel 1 eine Abhandlung über den theoretischen Unterbau geben. Anschließend wird in Kapitel 2 die Anwendung des selbst programmierten und Scikit-Learn Algorithmus auf einen ausgewählten Datensatz beschrieben. Am Ende der Arbeit befindet sich eine Auswertung der Ergebnisse.

## Algorithmus

### Funktionsweise

Der KNN-Algorithmus in Form der Klassifizierung zählt zu den supervised Machine Learning Algorithmen, das bedeutet es werden annotierte (gelabelte) Lerndaten verwendet, um nicht annotierten Datensätze das richtige Label zuzuordnen.[[1]](#footnote-1) Die Vorhersage wird durch die Grundannahme des Algorithmus, ähnliche Objekte existieren in unmittelbarer Nähe, getroffen.

Daraus ergibt sich folgendes Grundgerüst des Algorithmus:

1. Auswahl von Trainingsdaten und Selektieren von wichtigen Merkmalen.
2. Wähle k, um die Anzahl der betrachteten Nachbarn festzulegen.
3. Berechne alle Abstände zum klassifizierenden Datensatz.
4. Sortiere die Liste und nimm von den ersten k Einträgen das Label.
5. Treffe die Vorhersage je nach Aufgabe
   1. Klassifikation: verwende das häufigste diskrete Label.
   2. Regression: verwende den Durchschnitt der kontinuierlichen Label.2

Je nach Anwendungsfall muss der Parameter k und das verwendete Abstandsmaß individuell bestimmt werden. Deshalb lohnt es sich diese genauer anzuschauen.

### Abstandsmaße

Jeder Datensatz umfasst mehrere Merkmale, die ihn beschreiben. Um daraus eine Distanz zum nächsten Datensatz zu errechnen, wird jeder Datensatz in einem Merkmalsvektor repräsentiert. Nicht numerische Merkmale werden deshalb mithilfe von „Categorical Encoding“, wie zum Beispiel den OneHotEncoder von Scikit-learn, umgewandelt. Anschließend ist es möglich mithilfe der einfachen Geometrie die Distanz zu ermitteln. Die häufigste verwendete Distanz ist die euklidische Distanz, die wie folgt  
 .3

Eine weiteres wichtiges Distanzmaß ist die Mannhatten Metrik, auch Taxi Metrik genannt. Im Gegensatz zur euklidischen Distanz, wird hier „die Summe der Differenzen entlang der verschiedenen Dimensionen X und Y gebildet“. Die Gleichung sieht wie folgt aus: 4

Beide Distanzmaße können mithilfe der Minkowski-Metrik zusammengefasst werden.

Diese wird standardmäßig im Scikit-learn Algorithmus verwendet und sieht folgendermaßen aus:

2Vgl. Onel, Harrison 2018.

3Vgl. Provost 2017, S. 178f.

4Vgl. Provost 2017, S. 198.

Setzt man p=1, dann erhält man die Mannhatten Metrik und bei p = 2 den euklidischen Abstand. Allgemein wichtig bei der Verwendung dieser Distanzmaße ist, dass Merkmale durch Feature Scaling standardisiert werden müssen, um zu verhindern, dass ein Merkmal mehr zum Abstand beiträgt.[[2]](#footnote-2) Möglich ist dies mit einem MinMaxScaler(), dieser skaliert alle Werte auf einen Bereich zwischen 0 und 1.

Dies ist nur eine Auswahl an Distanzmaßen, unter der Scikit-learn Dokumentation sind noch weiter zu finden. Zum Beispiel auch speziell zur Messung der Nähe von Strings wie die Kosinus Metrik.[[3]](#footnote-3)

### Parameter k

Der Parameter k gibt die Anzahl der betrachteten Nachbarn zur Bestimmung des Labels für das zu klassifizierende Objekt an. Damit hat er eine starke Auswirkung auf den Prozess, denn je nachdem wie die Wahl ausfällt kommt es zu unterschiedlichen Ergebnissen im Validierungsprozess. Allgemein ist es von Vorteil k auf eine ungerade Zahl zu setzen, um ein Unentschieden in Mehrheitsentscheidungen zu vermeiden. Setzt man k minimal, dies bedeutet k=1, entsteht ein komplexes Modell. Da immer nur ein Nachbar verwendet wird, hat jeder Trainingsdatensatz seinen eigenen Bereich. Hat man nun Ausreißer im Datensatz, kann es zu Fehlentscheidungen kommen. Setzt man k hingegen maximal, also k entspricht der Anzahl an Trainingsdatensätzen, wird zur Vorhersage der komplette Datensatz verwendet. Dies hat zur Folge, dass für die Klassifikation immer die häufigste vorkommende Klasse verwendet wird. Bei der Regeression kommt es zum Durchschnitt aller Zielvariablen. Man hat kaum Komplexität im Modell. Deshalb muss man einen Kompromiss zwischen zu wenig und zu viel betrachteten Nachbarn schließen. Dazu kann man mehrere KNN-Modelle trainieren, die unterschiedliche k Werte besitzen und diese evaluieren. Danach wählt man das Modell mit dem besten Validierungswert auf den Trainingsdaten aus.

Ein weiterer Punkt ist die Gewichtung verschiedener Nachbarn. In der einfachen Betrachtungsweise des Algorithmus wird für die Entscheidung jeder Nachbar gleichbehandelt. Es könnte aber in manchen Anwendungen sinnvoll sein, näheren Nachbarn mehr Gewicht wie weiter entfernteren Nachbarn zuzuschreiben. Verwendet man diese Ansatzweise, verliert der k-Wert an Bedeutung im Gegensatz zur ungewichteten Mehrheitsentscheidung.[[4]](#footnote-4)

### Lazy Learning

Eine Besonderheit des KNN-Algorithmus ist, dass er zu den „Lazy-Learning Algorithmen (’träges Lernen’)“ zählt. Im Gegensatz zu anderen Machine Learning Algorithmen kommt es zu keinem Lernaufwand und zu keiner „Modellbildung“ während der Trainingsphase, denn der Algorithmus speichert lediglich alle Trainingsdaten. Der eigentliche Rechenaufwand geschieht erst bei der Einordnung einer Instanz. Deshalb bezeichnet man dies zusätzlich als instanzbasiertes Lernen. Da man sich dafür wieder an alle Traningsbeispiele „erinnern“ muss, wird die benötigte Rechenleistung bei zunehmender Größe eines Datensatz größer.[[5]](#footnote-5)

# Anwendung des KNN

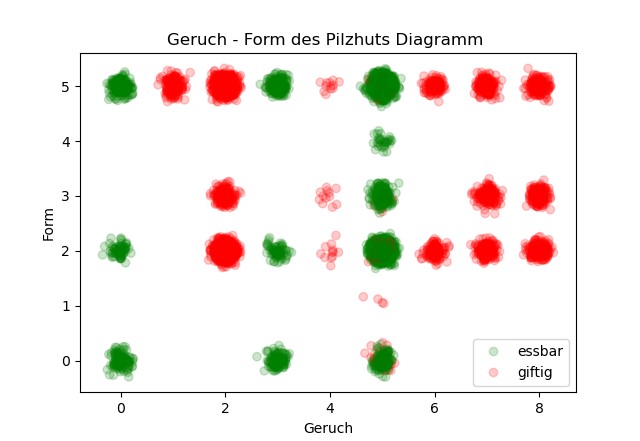
## Datenvorbereitung

In dieser Hausarbeit wird ein Datensatz über Pilze verwendet. Die Pilze sind entweder giftig oder essbar. Um dies zu beurteilen liegen 22 weitere Merkmale, sogenannte Features vor. Diese beschreiben das äußere Erscheinungsbild und den Geruch der Pilze. Der Datensatz enthält 8124 Einträge und ist dabei, mit 52% essbar und 48% giftig, gleichmäßig verteilt.[[6]](#footnote-6)

Die Daten liegen in einem CSV-File vor und werden mithilfe von Pandas DataFrame eingelesen. Da es sich bei den Features um Textdaten handelt, werden diese mithilfe eines „LabelEncoders“ in eindeutige Zahlenwerte umgewandelt. Daraufhin wird mit einem „MinMaxScaler()“ alle Daten auf einen Zahlenraum zwischen 0 und 1 skaliert. Dies ist notwendig, damit verschiedene Features vergleichbar werden. Für die spätere Verwendung wird anschließend der Datensatz in Trainingsdaten und Testdaten aufgeteilt.

Mithilfe einer Scatter-Matrix wird jedes Feature über jedes Feature in einem Plot dargestellt, sodass die Korrelationen zwischen den Merkmalen und der Zielvariable (essbar oder giftig) untersucht werden kann. Mithilfe dieser Analyse wurde aufgrund von Geschwindigkeitsvorteilen die Auswahl in dieser Hausarbeit auf die Merkmale Geruch und Form des Pilzhutes reduziert.

Aufgrund nur zweier Features kann auch folgender Plot erstellt werden. Dieser zeigt, dass die gewählten Features den Datensatz gut genug teilen. Jedoch ist auch zu erkennen, dass dies nicht perfekt geschieht. Manche giftigen Pilze sind nur umgeben von essbaren. In diesen Fällen hat der Algorithmus keine Chance die Zielvariable richtig zu bestimmen. Es muss eine Abwägung zwischen Komplexität und Richtigkeit getroffen werden. Da es sich bei den Daten um kategorische Daten handelt, sind viele Datenpunkte an identischen Stellen. Um trotzdem ein aussagekräftiges Schaubild zu erzeugen, wurde den Daten ein wenig zufällige Abweichung gegeben (Random Noise) und außerdem die Deckraft der Punkte auf 0,2 reduziert.



## Anwendung KNN mit Sklearn

Das Erstellen des KNN Models mithilfe von Scikit-learn erfolgt folgendermaßen:

1. knn = KNeighborsClassifier(n\_neighbors=k,
2. metric="euclidean",algorithm="brute")

„n\_neighbors“ gibt die Anzahl an untersuchten Nachbarn an. In dieser Hausarbeit werden die 5 nächsten Nachbarn untersucht. Als Abstandsberechnung für die Ermittlung der Nachbarn wird der Euklidische Abstand verwendet. Zusätzlich kann in Scikit-learn ein Algorithmus ausgewählt werden. Dieser entscheidet zu welchen Nachbarn überhaupt der Abstand ermittelt werden muss. Dies macht das ganze Modell deutlich effizienter. Normalerweise wird dies automatisch gewählt. In diesem Fall wird „brute“, also Brute-Force verwendet, das bedeutet es wird der Abstand zu jedem Nachbarn ermittelt. Bei kleinen Datenmengen wie hier reicht die Geschwindigkeit aus. Zudem ist dadurch eine bessere Vergleichbarkeit mit der Eigenentwicklung gegeben, denn dort wird ebenfalls der Abstand zu jedem anderen Datensatz berechnet. Alternativen wären Baum-Algorithmen, sodass während der Trainingsphase ein Baum gebildet wird, welcher die Anzahl der zu berechnenden Nachbarn reduziert. Eine Möglichkeit ist durch relative Abstände zu Dritten. „[...]if point A is very distant from point B , and point C is very close to point B, then we know that points A and C are very distant[...]“.[[7]](#footnote-7)

Nun wird das erstellte Modell wie folgt trainiert und getestet:

1. knn.fit(X\_train, y\_train)
2. y\_prediction = knn.predict(X\_test)

3

1. accurarcy = metrics.accuracy\_score(y\_test,y\_prediction)
2. class\_report = metrics.classification\_report(y\_test,y\_prediction)

„knn“ ist das erstellte Modell. Mit „fit()“ kann trainiert werden, da Brute-Force ausgewählt wurde werden hier nur die Trainingsdaten dem Modell übergeben. Der Rechenprozess findet erst bei „predict()“ statt. Dem Modell wird ein neuer für ihn unbekannter Pilz gegeben und er ermittelt den Abstand zu jedem anderen ihm bekannten Pilz. Die 5 Pilze, die dem neuen Pilz am Nächsten sind, entscheiden mit ihrer Zielvariable (essbar oder gifitg) per Mehrheitsentscheid (weights=“uniform“) die Zielvariable des unbekannten Pilzes. Dies bedeutet jeder Nachbar hat die gleiche Stimmkraft. Eine andere Möglichkeit wäre die Stimmkraft vom Abstand zum unbekannten Pilz abhängig zu machen (weights=“distance“).

Um nun herauszufinden ob die Vorhersage zutrifft, werden die Kenngrößen Accurarcy, Precision und Recall zur Rate gezogen. Mit diesem Datensatz kann eine Genauigkeit von fast 100% bestätigt werden. Wenn die Auswahl nicht nur auf 2 Features reduziert wird, kann auch eine Genauigkeit, Precision und Recall von jeweils 100% festgestellt werden. Das bedeutet, er teilt alle Datensätze in die richtige Klasse ein.

## Eigenentwicklung des KNN Algorithmus

Um den KNN Algorithmus selbst entwickeln zu können, werden nur die Bibliotheken pandas und numpy benötigt. Pandas ist dafür da, die Daten bereitzustellen und numpy zum Berechnen des euklidischen Abstands. Des Weiteren wurde auch in dieser Version Scikit-learn zum Daten vorbereiten und validieren verwendet. Somit hat die Eigenentwicklung dieselben Voraussetzungen wie die Entwicklung mit dem Algorithmus aus

Scikit-learn.

Nachdem die Daten vorbereitet und in Testdaten und Trainingsdaten aufgeteilt sind, wird der Algorithmus ausgeführt.

Dafür werden zuerst die echten Zielvariablen gesondert gespeichert und eine Liste für die Vorhersagen angelegt:

1. real\_values = y\_test.iloc[:bereich].tolist()
2. preds = []

Anschließend wird eine Schleife so oft wiederholt, wie die Anzahl der Trainingsdaten beträgt. Für Testzwecke kann hier auch eine geringere Anzahl gewählt werden, sodass wie hier nur die ersten 100 Datensätze untersucht werden.

1. *#Parameter festlegen*
2. bereich = 100 *#len(X\_test)*
3. k=5

4

1. **for** i **in range**(bereich):
2. neighbours = get\_neighbour(X\_train.to\_numpy(),
3. X\_test.iloc[i].to\_numpy(), k, y\_train )
4. pred = predict(neighbours)
5. preds.append(pred)

10

1. *#Evaluierung*
2. eval\_results(preds, real\_values)

Die Zuweisung k=5 gibt an, dass die 5 nächsten Nachbarn entscheidend sein sollen. Für jeden Schleifendurchgang werden nun die k nächsten Nachbarn ermittelt. Dies Funktioniert mit der Funktion get\_neighbour(). Anschließend wird in der Funktion predict() mit einer einfachen Mehrheitsentscheidung die Klasse (giftig oder essbar) bestimmt. Im letzten Schritt erfolgt noch die Evaluierung analog zu der Entwicklung mit Sklearn. Die Funktion get\_neighbour() funktioniert folgendermaßen:

1. def calc\_distance(x1, x2):
2. distance = np.linalg.norm(x1-x2)
3. return distance

4

1. def get\_neighbour(points, x1, k,y):
2. distances = []
3. for i in range(len(points)):
4. dist=calc\_distance(x1, points[i])
5. distances.append((dist,y.iloc[i]))
6. distances.sort(key=lambda t: t[0])
7. return distances[:k]

Dieser Funktion werden alle bekannten Punkte, den zu untersuchenden Punkt, sowie der Parameter k und die Zielvariablen der Punkte übergeben. Innerhalb der Schleife wird der Abstand von x1 zu jedem anderen Punkt gemessen und notiert. Anschließend wird diese Liste sortiert und die k nächstliegenden Punkte werden zurückgegeben.

Die Abstandsberechnung erfolgt in der Funktion calc\_distance(x1,x2) dafür wird mithilfe von Numpy der erste Punkt vom zweiten abgezogen. (Möglich ist dies, da beim Funktionsaufruf von get\_neighbour() alle Werte als Numpy-Array übergeben werden.) Der Befehl np.linalg.norm() ermittelt lediglich den Betrag eines Vektors, was in diesem Fall den Abstand der zwei Punkte beschreibt.

# Fazit

Durch die theoretische Betrachtung und praktische Implementierung des KNN-Algorithmus lässt sich am Ende dieser Arbeit folgendes feststellen. Der Algorithmus zählt zu den einfachen Machine Learning Algorithmen, dies merkt man an der einfachen Handhabung und Implementierung, besonders durch die Bibliothek „Scikit-learn“. Des Weiteren liegt dem Algorithmus ein sehr verständliches Konzept zu Grunde, wie in der Einleitung schon erwähnt, beurteilen wir Menschen neue Informationen oft durch ähnlich Erlebtes. Deshalb findet der KNN-Algorithmus in vielen Lebensbereichen Anwendung, so beruhen zum Beispiel Netflix Filmempfehlungen darauf. Zudem ist der Trainingsprozess nicht zeitintensiv. Und selbst in großen Datenmengen können heutzutage Nachbarn durch K-D-Bäume und Hashing Verfahren schneller gefunden werden. Diese werden auch bei Scikit-learn verwendet. Deshalb ist dieser Algorithmus auch besser als der selbst programmierte Algorithmus[[8]](#footnote-8) .

Trotz der scheinbaren Einfachheit des Algorithmus, kann man feststellen, dass während der Implementierung einiges zu beachten gilt. Besonders wichtig sind die Vorverarbeitung und Auswahl von Daten. Damit der Abstand aussagekräftig wird, müssen alle Merkmale zum einen in Zahlen umgewandelt werden (Categorical Encoding) und zum anderen skaliert (Feature Scaling) werden.

Besonders schwierig ist die Auswahl von passenden Merkmalen. Der KNN-Algorithmus hat mit zu vielen Dimensionen Schwierigkeiten, denn „der Merkmalsraum einer Trainingsdatenmenge fester Größe [ist] zunehmend dünner besetzt, wenn die Anzahl der Dimensionen ansteigt [...],dass selbst die engsten Nachbarn in einem hochdimensionalen Raum zu weit voneinander entfernt sind, um eine gute Abschätzung vornehmen zu können.“12 Nun stellt sich die Frage, welche Merkmale relevant sind und den Sachverhalt passend beschreiben. Und genau in dieser Auswahl befindet sich die größte Herausforderung für den KNN Algorithmus.

# Literaturverzeichnis

*1.6. Nearest Neighbors — scikit-learn 0.23.1 documentation* (2020). url: [https:// scikit-learn.org/stable/modules/neighbors.html](https://scikit-learn.org/stable/modules/neighbors.html) (besucht am 14.06.2020).

*Mushroom Classification* (2020). Library Catalog: www.kaggle.com. url: [https:// kaggle.com/uciml/mushroom-classification](https://kaggle.com/uciml/mushroom-classification) (besucht am 14.06.2020).

Onel, Harrison (10. Sep. 2018). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. url: [https://towardsdatascience.com/machine-learningbasics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761](https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761) (besucht am 30.05.2020).

Provost, Foster (2017). *Data Science für Unternehmen: Data Mining und datenanalytisches Denken praktisch anwenden*. 1. Aufl. Frechen: MITP Verlags-GmbH et Co. KG. isbn: 978-3-95845-546-7.

Raschka, Sebastian (2018). *Machine Learning mit Python und Scikit-Learn und TensorFlow: das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. 2. Aufl. Frechen: MITP Verlags-GmbH et Co. KG. isbn: 9783-95845-734-8.

**Ehrenwörtliche Erklärung**

Wir Versicherern hiermit, dass wir die vorliegende Arbeit mit dem Thema: *K-Nearest Neighbour Algorithm* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

24.06.2020

Datum Hanna Siegmann, Lukas Benner

1. Vgl. *1.6. Nearest Neighbors — scikit-learn 0.23.1 documentation* 2020. [↑](#footnote-ref-1)
2. Vgl. Raschka 2018, S. 122. [↑](#footnote-ref-2)
3. Vgl. Provost 2017, S. 198. [↑](#footnote-ref-3)
4. Vgl. Provost 2017, S. 187ff. [↑](#footnote-ref-4)
5. Vgl. Provost 2017, S. 189, 195. [↑](#footnote-ref-5)
6. Vgl. *Mushroom Classification* 2020. [↑](#footnote-ref-6)
7. K-D Tree *1.6. Nearest Neighbors — scikit-learn 0.23.1 documentation* 2020. [↑](#footnote-ref-7)
8. Vgl. Provost 2017, S. 196. 12Raschka 2018, S. 122. [↑](#footnote-ref-8)