TU Clausthal

Seminararbeit

Data Mining und Maschinelles Lernen

Ein Einblick in die Thematik

Janek Boll
447344

Steven Minich
446680

14. August 2019

Institut für Wirtschafstinformatik
Prof. Dr. Jörg P. Müller

Eidestattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.
Mit einer Veröffentlichung der Arbeit in der Instituts- beziehungsweise Universitätsbibliothek bin ich einverstanden.

Clausthal-Zellerfeld, 14. August 2019

_____

Eidestattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.
Mit einer Veröffentlichung der Arbeit in der Instituts- beziehungsweise Universitätsbibliothek bin ich einverstanden.

Clausthal-Zellerfeld, 14. August 2019

_____

Zusammenfassung

Maschinelles Lernen stellt einen sich schnell entwickelnden Teilbereich der *künstlichen Intelligenz* dar. Durch erhöhte Rechenleistung und Speicherkapazität sind die verwendeten Verfahren auf immer größeren Datenmengen effektiv anwendbar. Daher ist es wenig verwunderlich, dass ein wichtiges Anwendungsgebiet für *maschinelles Lernen* das *Data Mining*, das heißt das Extrahieren von Wissen aus großen Datenmengen, ist. In dieser Arbeit sollen die notwendigen Grundlagen der Statistik sowie des Lernens mit Lehrer dargestellt werden.

Inhaltsverzeichnis

1 Einleitung

Als *Data Mining* werden Verfahren bezeichnet, deren Ziel es ist, Wissen aus großen Datenmengen zu extrahieren. Dazu werden Verfahren aus der *Statistik* und des *maschinellen Lernens* eingesetzt.[Ert08] *Maschinelles Lernen* ist ein Teilbereich der *Künstlichen Intelligenz*, dessen Aufgabe das Lernen von Abbildungen einer Eingabemenge in eine Ausgabemenge ist. Typische Anwendungen sind die *Klassifikation*, also eine Abbildung einer Eingabe auf eine Klasse, und *Approximation*, sprich die Berechnung eines Funktionswertes, der möglichst nahe an der Realität ist[Ert08, MS17]. In den vergangenen Jahren hat das *maschinelle Lernen* große Fortschritte gemacht, was im Wesentlichen auf die Zunahme von Rechenleistung und Speicherkapazität zurückgeführt werden kann.[MS17] So werden bereits heute medizinische Diagnosen unter Verwendung sogenannter *Expertensysteme* gestellt[Ert08] und Behandlungspläne für Krebspatienten durch maschinelles Lernen unter Verwendung der Patientendaten anderer Patienten automatisch erstellt.[MS17] Der Online-Versandhändler Amazon setzt Data Minig ein, um seinen Kunden relevante Artikel zu präsentieren.[Ert08] Auch bei der Kreditvergabe setzen Finazdienstleister heutzutage maschinelle Lernverfahren ein, um Kreditanfragen zu prüfen.[MS17] Mit zunehmender Bedeutung von *Data Mining* und *maschinellem Lernen* sind auch leicht zu verwendende Frameworks erschienen, die dem Anwender einen leichten Einstieg in die Materie ermöglichen. Zu den bekanntesten Vertretern gehören *Scikit Learn*[1], ein Framework für die Programmiersprache Python, und *Tensorflow*[2], ein Framework von Google. Da künstliche Intelligenz und insbesondere maschinelles Lernen bereits Einfluss auf das tägliche Leben haben, und davon auszugehen ist, dass dieser Einfluss in den kommenden Jahren weiter zunimmt, soll im Rahmen dieser Arbeit ein grundlegendes Verständnis für die zugrundeliegenden Verfahren hergestellt werden. Wir werden uns dabei auf statistische Lage-, Streuungs- und Zusammenhangsmaße im Bereich der Statistik und auf Lernverfahren mit Lehrer im Bereich des maschinellen Lernens konzentrieren.

2 Statistische Grundlagen

2.1 Skalenniveaus

Untersuchte Merkmale können unterschiedliche Eigenschaften bezüglich ihrer Messbarkeit aufweisen. Diese Eigenschaften haben wesentlichen Einfluss darauf, welche Lage-, Streuungs- und und Zusammenhangsmaße auf diese Merkmale angewendet werden können. Daher sollen im Folgenden sollen zunächst *Skalenniveaus* eingeführt werden, anhand welcher die Unterschiede der verschiedenen Messbarkeitseigenschaften deutlich werden.

2.2 Nominalskala

Die einfachste Skalierung in der deskriptiven Statistik ist die sogenannte *Nominalskala*. Die untersuchten Merkmale lassen sich nur im Hinblick auf Gleich- beziehungsweise

[1] `https://scikit-learn.org/stable/`
[2] `https://www.tensorflow.org`

Ungleichheit untersuchen; eine Ordnung der Merkmalsausprägungen ist nicht möglich. [Koh05]

2.2.1 Beispiel

An einer Kreuzug werden die vorbeifahrenden Autos hinsichtlich ihrer Farbe untersucht. Mögliche Ausprägungen des untersuchten Merkmals Farbe sind etwa *Grün*, *Schwarz*, *Weiß*, *Gelb* und *Rot*. Zwar lassen sich die Merkmalsausprägungen voneinander unterscheiden, aber offensichtlich ist ein rotes Auto nicht besser oder schlechter als ein gelbes oder schwarzes Auto.

2.3 Ordinalskala

Ordinalskalierte Merkmale sind solche, bei denen neben der Unterscheidbarkeit der Merkmalsausprägungen auch eine natürliche Reihenfolge der Ausprägungen zugrunde liegt, wobei der Abstand zwischen den einzelnen Merkmalsausprägungen aber nicht sinnvoll interpretierbar ist. [Koh05]

2.3.1 Beispiel

Die gängigen Kleidergrößen von XS bis XXL verdeutlichen dies. Offenbar ist eine Größe M größer als eine Größe XS, jedoch ist keine Aussage über das Verhältnis des Größenunterschieds der beiden Größen möglich.

2.4 Metrische Skalen

Eine Skala deren Merkmalsausprägungen relle Zahlen sind, und die auch die Ordnungsstruktur der reellen Zahlen aufweist, wird *metrische Skala* genannt. Neben der Unterscheidung verschiedener Merkmalsausprägungen und der zugrunde liegenden Ordnungsrelation zeichnen sich metrische Zahlen dadurch aus, dass die Merkmalsausprägungen jeden möglichen Wert annehmen können. Da die Werte reelle Zahlen sind gibt es also überabzählbar unendlich viele Merkmalsausprägungen und der Wertebereich metrischer Skalen wird *stetig* genannt. Neben der Stetigkeit des Wertebeichs ist die Möglichkeit der sinnvollen Interpretation der Abstände von metrisch skalierten Merkmalen eine charakterisierende Eigenschaft ebendieser. [Koh05]

2.4.1 Beispiel

Die Entfernung zweier Punkte in Metern weist die Eigenschaften metrisch skalierter Merkmale auf: Zum Beispiel kann die Entfernung zweier Punkte jeden beliebigen reellen Wert annehmen, zum anderen lassen sich verschiedene Merkmalsausprägungen problemlos ordnen; eine Entfernung von $0,5m$ ist kleiner als eine Entfernung von $0,78m$. Darüber hinaus ist auch der Abstand interpretierbar, so ist ein Abstand von $4m$ augenscheinlich viermal so groß wie ein Abstand von $1m$.

2.4.2 Weitere Unterscheidung metrischer Merkmale

Anhand der Existenz eines natürlichen Nullpunktes lassen sich metrisch skalierte Merkmale weiter unterscheiden. Ein metrisches Merkmal heißt *intervallskaliert*, wenn es keinen natürlichen Nullpunkt gibt. [Koh05] Ein Beispiel hierfür ist die Temperatur, der Nullpunkt ist willkürlich festgesetz. Misst man die Temperatur in Grad Celsius, so entspricht der Nullpunkt dem Gefrierpunkt von Wasser, misst man jedoch in Grad Fahrenheit, so entspricht der Gefriepunkt von Wasser 32 Grad Fahrenheit und der Nullpunkt in Grad Fahrenheit entspricht etwa $-17,78$ Grad Celsius.
Exsitiert hingegen ein natürlicher Nullpunkt, so spricht man von *verhältnisskalierten* Merkmalen. [Koh05]
Ein Beispiel für ein verhältnisskaliertes Merkmal stellt das Gewicht dar. Zwar existieren verschieden Maßeinheiten wie Gramm, Pfund oder Unzen, jedoch entspricht ein Gewicht von 0 Gramm ebenso einem Gewicht von 0 Pfund und ebenso 0 Unzen.

2.5 Lagemaße

Einen ersten Ansatz zur Datenanalyse liefern sogenannte *Lagemaße*. Sie liefern Aufschluss über die Häufigkeitsverteilung der untersuchten Merkmale einer Stichprobe. [Koh05]

2.5.1 Modus

Der *Modus* x_{mod} ist ein sehr einfaches Lagemaß, dass sich hauptsächlich für nominalskalierte Merkmale eignet. Der Modus ist die Merkmalsausprägung mit der größten Häufigkeit. [Koh05]

2.5.2 Median

Der *Median* x_{med} entspricht dem Wert in der Mitte. Es folgt, dass der Median nur für mindestens ordinalskalierte Merkmale angewendet werden kann, da den Merkmalsausprägungen eine Ordnungsstruktur zugrundeliegen muss. [Koh05]
Zur Berechnung des Medians wird wie folgt vorgegangen: Zuächst werden die Merkmalsausprägungen geordnet und anschließend das Mittlere Element ausgewählt. Seien dazu n Stichproben genommen worden und es entspreche $x_i, i \in \{1, \ldots, n\}$ dem Wert der i-ten Probe. Analog entspreche $x_{(i)}, i \in \{1, \ldots, n\}$ der geordneten Probe, d.h. $x_{(i)} \leq x_{(i+1)} \forall i \in \{1, \ldots, n\}$.
Zur Bestimmung des mittleren Elements der Probe muss unterschieden werden, ob die Probe aus einer geraden oder ungeraden Anzahl von Elementen besteht, das heißt, ist n gerade oder ungerade.
Fall 1:
n ungerade

$$x_{med} := x_{(\frac{n+1}{2})}$$

Fall 2:
n gerade

$$x_{med} := \frac{1}{2}\left(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}\right)$$

2.5.3 Arithmetisches Mittel

Das *arithmetische Mittel* stellt das am weitesten verbreitete Lagemaß für metrische Merkmale dar. Im Gegensatz zu Modus und Median berücksichtigt das arithmetische Mittel alle Merkmalsausprägungen einer Probe, wird dadurch aber auch anfällig für *Ausreißer*.

$$\bar{x} := \frac{1}{n} \sum_{i=1}^{n} x_i$$

Aus der Formel folgt direkt, dass das arithmetische Mittel nicht auf nominal- oder ordinalskalierte Merkmale angewendet werden kann, da auf diesen keine Addition definiert ist.

2.6 Streuungsmaße

Streuungsmaße geben Auskunft darüber, wie stark die Ausprägungen des untersuchten Merkmals von einem Lagemaß wie dem arithmetischen Mittel abweichen, also wie sehr sie um den Mittelwert streuen. Sie liefern damit eine Beurteilungsmöglichkeit für das verwendete Lagemaß.[Ert08, Koh05]

2.6.1 Varianz

Die *Varianz* σ^2 ist das am häufigsten verwendete Strueungsmaß. Sie beschreibt die *mittlere quadratische Abweichung* der Merkmalsausprägungen vom Mittelwert. Wird die Grundgesamtheit betrachtet, so gilt die folgende Formel (*empirische Varianz*):

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

Soll dagegen lediglich eine Stichprobe, das heißt eine Teilmenge der Grundgesamtheit untersucht werden, so muss die Formel angepasst werden (*Stichprobenvarianz*):

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

In beiden Fällen ist jedoch zu beachten, dass die Maßeinheit der Varianz, aufgrund der Quadrierung des Abstandes zum Mittelwert, nicht mehr der Maßeinheit der urspürnglichen Werte entspricht.[Koh05]

2.6.2 Standardabweichung

Die *Standardabweichung* σ ist die positive Quadratwurzel der Varianz. Wie bereits erläutert entspricht die Maßeinheit der Varianz nicht der ursprünglichen Maßeinheit der Werte. Dieses Manko lässt sich mit der Standardabweichung beheben.

$$\sigma := +\sqrt{\sigma^2}$$

Durch das Ziehen der Quadratwurzel misst die Standardabweichung die mittlere Streuung um den Mittelwert in der gleichen Maßeinheit wie die ursprünglichen Werte.[Koh05]

2.7 Zusammenhangsmaße

Häufig, so auch im Bereich des *Data Mining*, interessiert man sich jedoch nicht nur für die Ausprägungen eines einzelnen Merkmals. Viel interessanter erscheint eine Untersuchung wie verschiedene Merkmale zueinander in Beziehung stehen. Unter Umständen Lassen sich aus den Beobachtungen eines Merkmals Rückschlüsse auf ein anderes Merkmal ziehen.

2.7.1 Kovarianz

Die *Kovarianz* $cov(x, y)$ beschreibt, wie sich zwei Merkmale eines beobachteten Merkmalsvektors zueinander in Bezug auf Abweichung ihres jeweiligen Mittelwerts verhalten. [Ert08]
Definition der Kovarianz:
Seien x, y zwei Merkmale eines Merkmalsvektors, n die Anzahl der in einer Stichprobe untersuchten Merkmalsvektoren. Dann gilt für die Kovarianz:

$$cov(x, y) := \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

[Koh05]
Aus der Definition folgt unmittelbar, dass der i. Merkmalsvektor einen positiven Beitrag zur Summe liefert, wenn die Merkmale x und y in die gleiche Richtung vom Mittelwert abweichen, analog liefert eine Abweichung der Merkmale in unterschiedliche Richtung vom Mittelwert einen negativen Beitrag zur Summe. [Ert08]

2.7.2 Korrelationskoeffizient

Da der Wert der Kovarianz unter Anderem auch von den Absolutwerten der Variablen abhängt, ist ein Vergleich der Werte problematisch. [Ert08]
Um diese Abhängikeit von den Absolutwerten zu eliminieren wird der *Korrelationskoeffizient* K_{xy} als normierte Kovarianz definiert als Kovarianz der beiden Variablen geteilt durch das Produkt der Standardabweichung von x und y:

$$K_{xy} := \frac{cov(x, y)}{\sigma_x \cdot \sigma_y}$$

Der Korrelationskoeffizient ist normiert auf das Intervall $[-1, 1]$, das heißt es gilt $-1 \leq K_{xy} \leq 1$) [Koh05]

3 Maschinelles Lernen

3.1 Lernen mit und ohne Lehrer

Eine erste Einteilung verschiedener maschineller Lernverfahren ist die, ob das Lernverfahren auf Trainingsdaten, also auf solche Daten, für die eine korrekte Klasseneinordnung

oder Funktionswert bereits bekannt ist, verwenden. In diesem Fall wird von *Lernen mit Lehrer* oder *supervised learning* gesprochen. Die richtige Klassifikation wird häufig von *Experten* vorgenommen.[Ert08]
Lernverfahren die hingegen *ohne* Trainingsdaten auskommen werden als Verfahren aus dem Bereich *Lernen ohne Lehrer* (*unsupervised learning*) bezeichnet.
In dieser Arbeit beschränken wir uns ausschließlich auf Lernverfahren, die auf der Verwendung von Trainingsdaten beruhen, bleiben also im Bereich des *Lernens mit Lehrer*.

3.2 Agenten

Der Begriff des *Agenten* ist in vielen Teilbereichen der Informatik von zentraler Bedeutung, allerdings wird der Agentenbegriff in verschiedenen Teilbereichen unterschiedlich ausgelegt. So stellt ein Agent in der klassischen Informatik eine Abbildung von einer Eingabe auf eine Ausgabe dar und wird als *Software-Agent* bezeichnet, während ein Agent in der Robotik einen *Hardware-Agenten* bezeichnet, der eine Erweiterung des Software-Agenten um Sensoren und Aktoren darstellt, die es dem Hardware-Agenten ermöglichen, seine Umgebung wahrzunehmen und zu verändern. [Ert08]

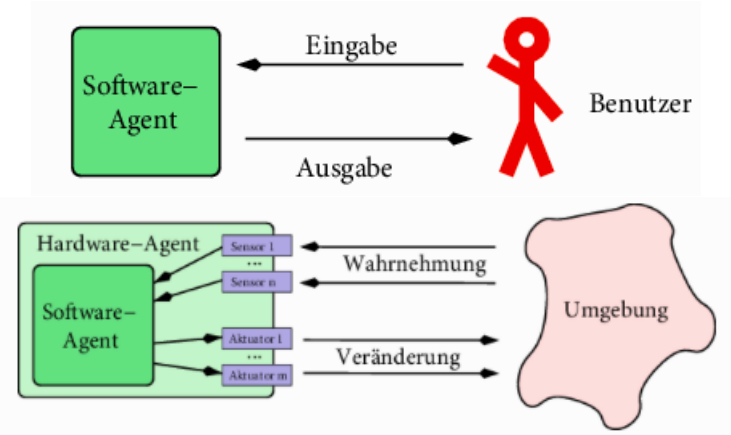

Abbildung 1: Soft- und Hardware-Agent
[Ert08, S.18, Abb. 1.5, S19, Abb. 1.6]

Auch bezüglich ihrer Intelligenz unterscheiden sich Agenten voneinander. Ein Agent, der lediglich eine Abbildung von der Menge aller möglichen Eingaben auf die Menge aller möglichen Ausgaben implementiert wird als *Reflex-Agent* bezeichnet. Ein Reflex-Agent ist in der Lage, Probleme zu lösen, bei denen es sich um Markov-Entscheidungsprozesse handelt. Zur optimalen Lösung dieser Klasse von Problemen ist lediglich Wissen über den aktuellen Zustand der Umgebung erforderlich.[Ert08]
Agenten mit Gedächtnis hingegen sind Agenten, die bei ihren Entscheidungen Wissen über vorangegange Zustände der Umgebung berücksichtigen können.[Ert08]

3.2.1 Der lernende Agent

Der im Rahmen dieser Arbeit zentrale Agentenbegriff ist der des *lernenden Agenten*. Der lernende Agent entspricht einer Abbildung eines Eingabevektors auf eine diskrete Klasse

oder einen stetigen Funktionswert. Er unterscheidet sich vom Reflex-Agenten dadurch, dass die konkrete Abbildungsvorschrift nicht vom Programmierer vorgegeben, sondern aus *Trainingsdaten* anhand eines *Lernverfahrens* eigenständig gelernt wird[Ert08].

Abbildung 2: lernender Agent

[Ert08, S.194, Abb. 8.4(rechts)]

Die *Aufgabe* des *lernenden Agenten* ist das lernen einer Abbildung von einem Merkmalsvektor auf eine Klasse oder einen stetigen Funktionswert anhand von *Trainingsdaten*. Es muss im Vorfeld festgelegt werden, welche Art von Agent, das heißt welches Lernverfahren, eingesetzt wird (siehe Abschnitt 4, S. 15 *Perzeptron* und Abschnitt 6.1, S. 25 *Nearest Neighbour Methoden*). Bei der Auswahl der Trainingsdaten ist zu beachten, dass diese repräsentativ für die zu lernende Aufgabe sind, da die gelernte Abbildung ansonsten nur schlecht auf unbekannte Daten generalisiert.
Nachdem der Agent eine Abbildung anhand von Trainingsdaten gelern hat, muss diese anhand eines geeigneten *Leistungsmaßes* unter Verwendung von *Testdaten* evaluiert werden. *Testdaten* sind dabei Merkmalsvektoren, für die die richtige Klasseneinteilung oder der richtige Funktionswert bereits bekannt sind, die aber keine Trainingsdaten waren[Ert08].

3.3 Eager Learning

Als *Eager Learning* werden solche Lernverfahren bezeichnet, bei denen in einem ersten Schritt in einer (in der Regel aufwändigen) Lernphase bestehendes Wissen aus den Trainingsdaten in Form einer Abbildung extrahiert wird. Der in der Lernphase betrieben Aufwand rechnet sich aber in der Anwendung, da diese Klasse von Lernverfahren, sobald die Abbildung gelernt wurde, sehr effizient auf neue, ungesehene Daten angewendet werden kann.[Ert08]

4 Das Perzeptron

In diesem Abschnitt der Arbeit soll mit dem *Perzeptron* ein einfaches, aber dennoch aufschlussreiches Beispiel für eine konkrete Implementierung eines lernenden Agenten gegeben werden.
Das Perzeptron ist eine *linearer Klassifizierer*, das heißt, es ist in der Lage, einen Eingabevektor auf eine von zwei Klassen abzubilden, sofern es eine Hyperebene gibt, die die beiden Klassen voneinander trennt[Ert08].

4.1 Linear separable Mengen

Zunächst soll verdeutlicht werden was es heißt dass zwei Mengen durch eine Hyperebene voneinander getrennt werden. Hierzu führen wir den Begriff der *linear separablen Mengen* ein.
Zwei Mengen $M_1 \subset \mathbb{R}^n$ und $M_2 \subset \mathbb{R}^n$ heißen *linear separabel*, genau dann, wenn es eine Schwelle θ und reelle Zahlen $a_1, \ldots, a_n$ gibt, sodass

$$\sum_{i=1}^{n} a_i x_i > \theta \forall x \in M_1$$

und

$$\sum_{i=1}^{n} a_i x_i \leq \theta \forall x \in M_2$$

gilt[Ert08].
In der Terminologie der linearen Algebra beduetet dies, es existiert eine $n-1$ dimensionale Hyperebene, die durch die Gleichung

$$\sum_{i=1}^{n} a_i x_i = \theta$$

definiert wird und die die beiden Mengen M_1 und M_2 voneinander trennt.

4.2 Abbildungsvorschrift

Wie bereits erwähnt handelt es sich beim *Perzeptron* um einen linearen Klassifizierer, der genau die linear separablen Mengen klassifizieren kann. Das heißt, das Perzeptron ist eine Abbildung eines Eingabevektors x auf einen diskreten Klassenwert. Es wird definiert durch die Abbildungsvorschrift

$$P(x) = \begin{cases} 1, & \text{falls } \sum_{i=1}^{n} w_i x_i > 0 \\ 0, & \text{falls } \sum_{i=1}^{n} w_i x_i \leq 0 \end{cases}$$

[Ert08]
Aus der Abbildungsvorschrift wird ersichtlich, dass n Skalare, $w_1, \ldots, w_n$ benötigt werden. Der Vektor $w = (w_1, \ldots, w_n)$ wird als *Gewichtsvektor* bezeichnet. Dieser Gewichtsvektor w ist im Allgemeinen zunächst unbekannt und muss in einem vorgelagerten Schritt gelernt werden.

4.3 Lernverfahren

Eine Möglichkeit, den Gewichtsvektor w zu ermitteln, stellt der Algorithmus PerzeptronLernen dar, der in diesem Abschnitt vorgestellt werden soll. Der Algorithmus PerzeptronLernen nimmt als Eingabe zwei Mengen M_+ und M_-, wobei die Menge M_+ Eingabevektoren enthält, von denen bereits bekannt ist, dass sie mit 1 klassifiziert werden sollen. Analog enthält die Menge M_- Eingabevektoren, von denen bereits bekannt

ist, dass sie mit 0 klassifiziert werden sollen.
Nun wird ein beliebiger initialer Gewichtsvektor w gewählt.
Für alle Elemente $x \in M_+$ wird geprüft, ob $w \cdot x := \sum_{i=1}^{n} w_i x_i \leq 0$ erfüllt ist. Ist dies der Fall, so wissen wir, dass das Element x falsch klassifiziert wurde, der Gewichtsvektor w also noch nicht richtig sein kann. Folglich wird der Gewichtsvektor w angepasst, indem der Wert des falsch klassifizierten Elementes $x \in M_+$ komponentenweise aufaddiert wird. Analog wird für die Elemente aus M_- vorgegangen, nur das hier geprüft wird, ob $\sum_{i=1}^{n} w_i x_i > 0$ gilt und w gegebenfalls durch Substraktion des Wertes des Eingabevektros x angepasst wird.

4.3.1 Algorithmus PerzeptronLernen

Data: $M_+, M_-, w \in \mathbb{R}^n$ beliebig
Result: Gewichtsvektor w
repeat
  for $x \in M_+$ **do**
    if $w \cdot x \leq 0$ **then**
      $w = w + x$
    end
  end
  for $x \in M_-$ **do**
    if $w \cdot x > 0$ **then**
      $w = w - x$
    end
  end
until *Alle Vektoren korrekt klassifiziert*;

Algorithm 1: PerzeptronLernen

[Ert08]

4.3.2 Beispiel

Das Vorgehen des Lernverfahrens soll im Folgenden anhand eines einfachen Beispiels verdeutlicht werden.
Als Trainingsdaten dienen die Mengen $M_+ = \{(0, 1.8), (2, 0.6)\}$ und
$M_- = \{(-1.2, 1.4), (0.4, -1)\}$, alsi initialer Gewichtsvektor w dient $(1, 1)$.
1. Durchlauf:
M_+
$(1, 1) \cdot (0, 1.8) = 1.8 > 0 \rightarrow$ erster Vektor korrekt klassifiziert.
$(1, 1) \cdot (2, 0.6) = 2.6 > 0 \rightarrow$ zweiter Vektor korrekt klassifiziert
M_-
$(1, 1) \cdot (-1.2, 1.4) = 0.2 \geq 0 \rightarrow$ dritter Vektor falsch klassifiziert
$\rightarrow w_{neu} = (1, 1) - (-1.2, 1.4) = (2.2, -0.4)$
$(2.2, -0.4) \cdot (0.4, -1) = 1.28 \geq 0 \rightarrow$ vierter Vektor falsch klassifiziert
$\rightarrow w_{neu} = (2.2, -0.4) - (0.4, -1) = (1.8, 0.6)$

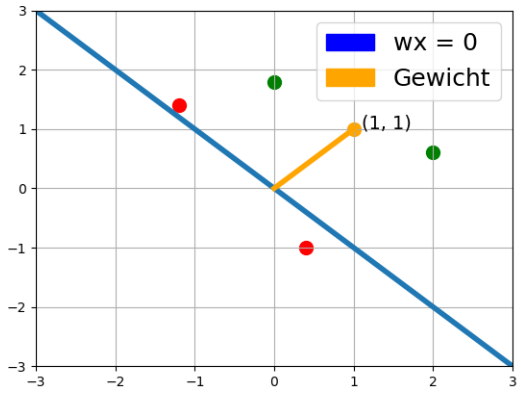

Abbildung 3: Situation vor Iteration 1

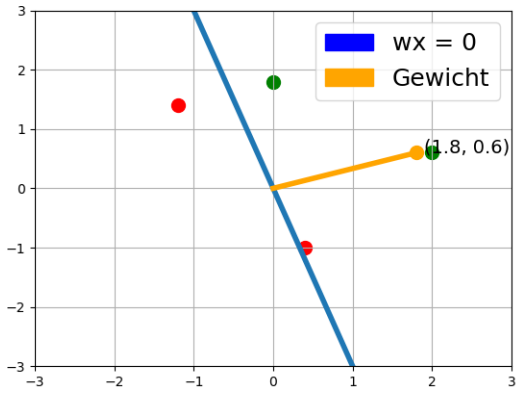

Abbildung 4: Situation nach Iteration 1

2. Durchlauf:
M_+
$(1.8, 0.6) \cdot (0, 1.8) = 1.08 > 0 \rightarrow$ erster Vektor korrekt klassifiziert.
$(1.8, 0.6) \cdot (2, 0.6) = 3.96 > 0 \rightarrow$ zweiter Vektor korrekt klassifiziert.
M_-
$(1.8, 0.6) \cdot (-1.2, 1.4) = -1.32 < 0 \rightarrow$ dritter Vektor korrekt klassifiziert.
$(1.8, 0.6) \cdot (0.4, -1) = 0.12 \geq 0 \rightarrow$ vierter Vektor falsch klassifiziert.
$\rightarrow w_{neu} = (1.8, 0.6) - (0.4, -1) = (1.4, 1.6)$

3. Durchlauf:
M_+
$(1.4, 1.6) \cdot (0, 1.8) = 2.88 > 0 \rightarrow$ erster Vektor korrekt klassifiziert.
$(1.4, 1.6) \cdot (2, 0.6) = 3,76 > 0 \rightarrow$ zweiter Vektor korrekt klassifiziert.
M_-
$(1.4, 1.6) \cdot (-1.2, 1.4) = 0,56 > 0 \rightarrow$ dritter Vektor falsch klassifiziert.
$\rightarrow w_{neu} = (1.4, 1.6) - (-1.2, 1.4) = (2.6, 0.2)$
$(2.6, 0.2) \cdot (0.4, -1) = 0.84 > 0 \rightarrow$ vierter Vektor falsch klassifiziert.
$\rightarrow w_{neu} = (2.6, 0.2) - (0.4, -1) = (2.2, 1.2)$

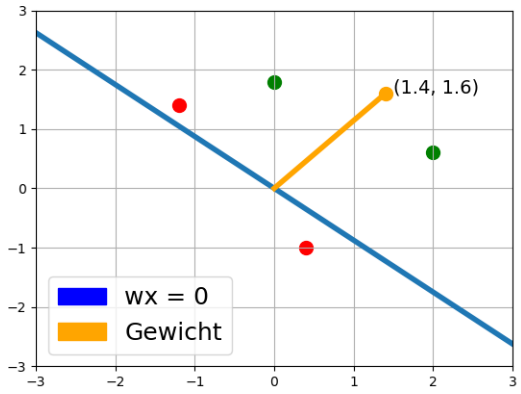

Abbildung 5: Situation vor Iteration 3

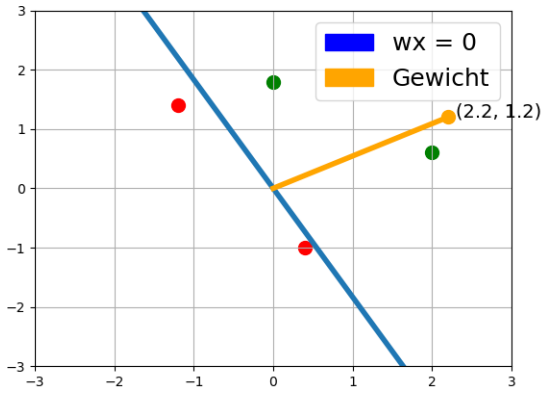

Abbildung 6: Situation nach Iteration 3

4. Durchlauf:
M_+
$(2.2, 1.2) \cdot (0, 1.8) = 2.16 > 0 \rightarrow$ erster Vektor korrekt klassifiziert.
$(2.2, 1.2) \cdot (2, 0.6) = 5.12 > 0 \rightarrow$ zweiter Vektor korrekt klassifiziert.
M_-
$(2.2, 1.2) \cdot (-1.2, 1.4) = -0.96 < 0 \rightarrow$ dritter Vektor korrekt klassifiziert.
$(2.2, 1.2) \cdot (0.4, -1) = -0,32 < 0 \rightarrow$ vierter Vektor korrekt klassifiziert.

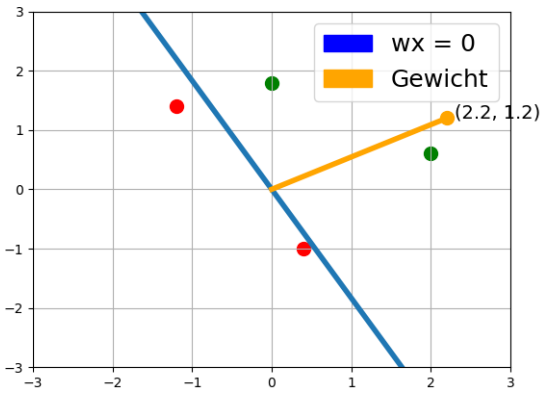

Abbildung 7: Situation nach Terminierung des Lernverfahrens

Der Gewichtsvektor $w = (2.2, 1.2)$, klassifiziert alle Elemente der Trainingsdaten korrekt, der Lernalgorithmus terminiert und gibt $w = (2.2, 1.2)$ als Ergebnis zurück.
Die Abbildungen 3, 4, 5, 6 und 7 veranschaulichen, wie eine Anpassung des Gewichtsvektors w die Trennebene beeinflusst. Zu beachten ist auch, dass der richtige Vektor bereits in der vorletzten Iteration gefunden wird. Da dieser Vektor aber noch auf allen Trainingsdaten getestet werden muss bedarf es einer weiteren Iteration.

4.3.3 Finden guter Initialwerte für w

Die asymptotische Laufzeitanalyse des Algorithmus PerzeptonLernen in Abhängigkeit der Eingabegröße n, wobei n der Anzahl der Trainingsdaten entspricht, gestaltet sich als schwierig. Einzig das Best-Case-Szenario, bei dem der initiale Gewichtsvektor so gewählt wird, dass es keiner Anpassung bedarf, lässt sich asymptotisch mit $\Omega(n)$ nach unten abschätzen, da jedes Element aus den Trainingsdaten, also $M_+ \cup M_-$, mindestens einmal überprüft werden muss. Für die Average-Case und Worst-Case Laufzeiten lässt sich so jedoch keine Aussage treffen, da die Wahl von w deutlich größere Auswirkungen auf die Laufzeit des Lernverfahrens hat. Folglich kann die Laufzeit des Algorithmus bei Verwendung eines optimalen intialen Gewichtsvektors w auf $\mathcal{O}(n)$ reduziert werden. Allerdings entspricht das Finden optimaler Gewichtsvektoren genau dem Ausführen des Lernverfahrens, dieses hat schließlich zum Ziel, ein w zu finden, dass die Mengen trennt, was äquivalent dazu ist, dass das w bei Verwendung als initialem Gewichtsvektor im Lernverfahren nicht mehr angepasst werden muss, da es alle Trainingsdaten korrekt klassifiziert. Ein finden optimaler Gewichtsvektoren ist also keine Option zu Verbesserung der Laufzeit.
Dennoch lassen sich durch die Verwendung heuristisch bestimmter „guter“ Gewichtsvektoren w im Durchschnitt Laufzeitverbesserungen erreichen. Eine geeignete Heuristik sieht wie folgt aus:
Alle $x \in M_+$ werden aufaddiert und davon wird die Summer aller $x \in M_-$ abgezogen.

$$w = \sum_{x \in M_+} x - \sum_{x \in M_-} x$$

[Ert08]

4.4 Erweiterung auf linear affine Mengen

Die bisherige Abbildungsvorschrift des Perzeptrons beruht darauf, dass die beiden Mengen durch eine Hyperebene durch den Nullpunkt trennbar sind. Dies widerspricht aber der Definition linear separabler Mengen, die lediglich voraussetzt, dass es eine beliebige Schwelle, insbesondere also auch solche ungleich 0 (vgl. Abschnitt 4.1). Daher stell sich unweigerlich die Frage, ob das Perzeptron tatsächlich alle linear separablen Mengen korrekt klassifizieren kann, oder lediglich solche, die durch eine durch den Ursprung verlaufende Hyperebene trennbar sind. Um zu zeigen, dass das Perzeptron auch solche Mengen, die durch eine durch eine beliebige Schwelle definierte Hyperbene trennbar sind, korrekt klassifizieren kann, greifen wir auf die bekannten Rechenregeln der reellen Zahlen für Ungleichungen zurück. Zunächst erweitern wir den Eingabevektor x um eine $n+1$. Komponente, die Konstant auf -1 gesetzt wird.
Damit ergibt sich für die Abbildungsvorschrift des Perzeptrons

$$P(x) = \begin{cases} 1, & \text{falls } \sum_{i=1}^{n} w_i x_i - w_{i+1} > 0 \\ 0, & \text{falls } \sum_{i=1}^{n} w_i x_i - w_{i+1} \leq 0 \end{cases}$$

und für die Gleichung der trennenden Hyperebene

$$\sum_{i=1}^{n} w_i x_i - w_{i+1} = 0$$

Durch einfache Äquivalenzumformung ergibt sich die folgende Gleichung für die Hyperebene

$$\sum_{i=1}^{n} w_i x_i = w_{i+1}$$

Daraus folgt für die Abbildungsvorschrift des Perzeptron:

$$P(x) = \begin{cases} 1, & \text{falls } \sum_{i=1}^{n} w_i x_i > w_{i+1} \\ 0, & \text{falls } \sum_{i=1}^{n} w_i x_i \leq w_{i+1} \end{cases}$$

Mit w_{i+1} ist also eine neue Schwelle ungleich 0 gefunden, die das Gewünschte liefert. Damit ist gezeigt, dass das Perzeptron auch solche Mengen klassifizieren kann, die durch eine nicht durch den Nullpunkt verlaufende Hyperebene voneinander getrennt werden können. Damit ist das Perzeptron in der Lage tatsächlich alle linear separablen Mengen zu klassifizieren.
Zu beachten ist, dass der Schwellwert w_{i+1} zunächt unbekannt ist, und erst im Verlaufe des Lernverfahrens zusammen mit dem Gewichstsvektor w gelernt wird.

4.5 Probleme des Perzeptrons

Obwohl das Perzeptron ein sehr einfaches maschinelles Lernverfahren ist, so hat es in der Praxis, außer als Vorstufe zu neuronalen Netzen, keine Bedeutung. Aufgrund der Einschränkung auf linear seprablen Mengen stößt das Perzeptron schnell an seine Grenzen. Dies wird bereits an einem einfachen Beispiel deutlich.
Gehen wir davon aus, dass das Perzeptron die booleschen Funktionen lernen soll. Das heißt es sollen Abbildungen der Form

$$\{0,1\} \times \{0,1\} \to \{0,1\}$$

gelernt werden. Zunächst betrachten wir die logische UND-Funktion, das heißt die Abbildung

$$f : \{0,1\} \times \{0,1\} \to \{0,1\} : x \mapsto f(x)$$

mit

$$f(x) = \begin{cases} 1, & \text{falls } x = (1,1) \\ 0, & \text{sonst} \end{cases}$$

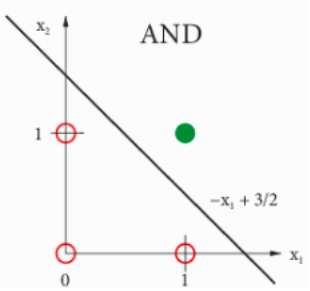

Abbildung 8: Hyperebene, die die mit 1 und 0 klassifizierten Elemente voneinander trennt
[Ert08, S.201, Abb. 8.8(links)]

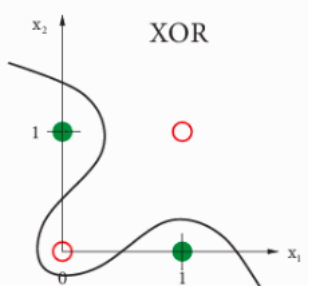

Abbildung 9: Versuch, die Klassen zu trennen
[Ert08, S.201, Abb. 8.8(rechts)]

An Abb. 8 wird deutlich, dass die Menge der Vektoren, die auf 1 abbilden durch eine Hyperebene von der Menge der Vektoren, die auf 0 abbilden.
Anders verhält es sich bei der logischen XOR-Funktion:

$$f : \{0,1\} \times \{0,1\} \to \{0,1\} : x \mapsto f(x)$$

mit

$$f(x) = \begin{cases} 1, & \text{falls } x = (1,0) \vee x = (0,1) \\ 0, & \text{sonst} \end{cases}$$

Auch in diesem Fall wird durch einen Blick auf den Graphen (Abb. 9) deutlich, dass es Möglichkeit gibt, die Menge der Vektoren, die auf 1 abbilden durch eine Hyperebene, die der Gleichung der Form

$$\sum_{i=1}^{n} w_i x_i = \theta$$

genügt, von der Menge der Vektoren, die auf 0 abbilden, zu trennen[Ert08].

5 Grundbegriffe und Abstandsfunktionen

5.1 Was ist Ähnlichkeit?

Als Ähnlichkeit wird häufig bezeichnet, wenn in zwei oder mehr beobachteten Objekten, in zu untersuchenden Ausprägungen, Übereinstimmungen zu finden sind.
Hierzu passend ist das Beispiel eines Arztes, der sich während einer Diagnose eines seiner Patienten daran erinnert, bereits in der Vergangenheit bei anderen Patienten die gleichen Symptome beobachtet zu haben.[Ert08] Durch das Erinnern, an einen Fall in der Vergangenheit und dem enstandenden Vergleich zur Gegenwart, ist es dem Arzt möglich seinen momentanen Patienten zu helfen, da die gleichen Behandlungsmethoden verwenden werden können.
Weil für Menschen Ähnlichkeit intuitiver ist als für Maschinen, stellt sich die Frage, wie man einer Maschine vermitteln kann, wann Objekte zueinander ähnlich sind und vorallem, wie man dieses Konzept in eine für Maschinen verständliche Sprache übersetzt und umgesetzt werden kann.

Zu diesem Zwecke wird die aus der Mathematik bekannten Datenstruktur der Vektoren betrachtet.
Vektoren sind in diesem System Tupel von Daten. Innerhalb dieses Systems werden Vektoren mit Werten der reellen Zahlen verwendet und für Werte aus anderen Räumen wird eine eindeutige Repräsentation gesucht.
Beispielsweise könnte eine Übersetzung von Zeichenketten in den reellen Raum über ein ASCII-System erfolgen.
Für unsere Anwendung wird die Eigenschaft ausgenutzt, dass Vektoren Elemente des Raumes $\mathbb{R}^n$ sind, wobei jede der n Dimensionen für eine konkrete Ausprägung einer Eigenschaft steht. Damit lassen sich Datenmengen als Punkte in einem $\mathbb{R}^n$ dimensionalen Hyperraum darstellen. Dadurch definiert Wolfgang Ertel: "Zwei Beispiele sind umso ähnlicher, je geringer ihr Abstand im Merkmalsraum ist."[Ert08, S. 207]

5.2 Einführung von Abstandsfunktionen

Innerhalb des definierten, metrischen n-dimensionalen Merkmalsraumes kann man infolgedessen die Unterschiedlichkeit zweier betrachteten Objekte mithilfe einer Abstandsfunktion bestimmen. Somit kennen wir die Ähnlichkeit zweier Datenpunkte zueinander, wenn der Abstand zwischen ihnen im Merkmalsraum bekannt ist.
Hierzu bieten sich eine Vielzahl an Metriken an z.B. die euklidische Norm, Manhattan-Abstand als auch der Hamming-Abstand. Weitere erwähnenswerte Metriken lauten: Summe der Abstandsquadrate, Abstand der maximalen Komponente.
Als *metrischen Raum* bezeichnet man eine Menge in der eine Abstandsfunktion $D(x, y)$ mit $x, y \in \mathbb{R}^n$ definiert ist.
Folglich kann man, innerhalb dieser definierten Menge, zwischen Elementen Abstände bestimmen, was sich besonders nützlich für die im Folgenden behandelten Anwendungen erweist.
Ein Entwickler muss sich zusätzlich bei der Auswahl einer Metrik im Klaren sein, welche Abstandsfunktion für seine Anwendung am besten geeignet ist.

5.3 Euklidische Norm

Eine namenhafte Abstandsfunktion, die meist mit dem Satz des Pythagoras eingeführt und berechnet werden kann, ist die der euklidischen Norm und ist wie folgt definiert:

$$x, y \in \mathbb{R}^n$$

$$d_e(x, y) := |x - y| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

In unterschiedlichen Anwendungsfällen können bestimmte Merkmale eine wichtigere Rolle spielen als andere.[Ert08] Dazu wird die euklidische Norm um einen Gewichtsvektor w erweitert, wobei dieser die Gewichtung jeder einzelnen Komponente bestimmt.
Eine gewichtete Variante der euklidischen Norm sieht wie folgt aus:

$$w, x, y \in \mathbb{R}^n$$

$$d_w(x, y) := |x - y| = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}$$

5.4 Manhattan-Abstand

Hierbei werden nur Geraden, die parallel zu einer Ursprungsachsen verlaufen betrachtet, um zwei Punkte zu verbinden. Die Distanz ergibt sich aus der aufsummierten Gesamtlänge jeder der dabei verwendeten Geraden.
Formal beschreibt der *Manhattan-Abstand*, dass die Distanz zwischen zwei Elementen aus einem mehrdimensionalen Raum den Differenzen jeder Komponente als absoluten Wert aufsummiert entspricht.
In Formelnotation sieht der *Manhattan-Abstand* wie folgt aus:

$$x, y \in \mathbb{R}^n$$

$$d_m(x, y) := \sum_{i=1}^{n} |x_i - y_i|$$

Ein Beispiel hierfür ist in Abbildung 10 zu sehen, wobei die folgenden Daten gegeben sind:
$x, y \in \mathbb{R}^2$ mit $x = (2, 2)$ und $y = (5, 4)$

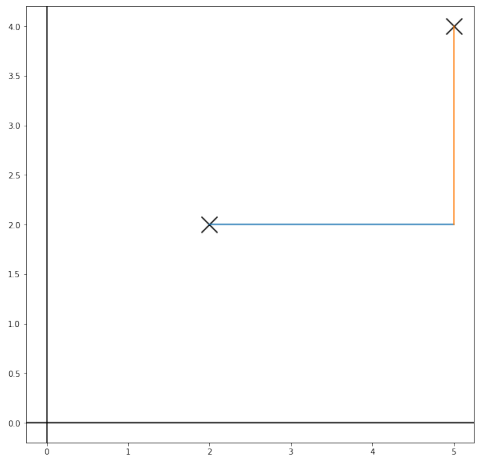

Abbildung 10: Beispiel Manhattan-Abstand

Für die Distanz ergibt sich dadurch: $d_m(x, y) = |2 - 5| + |2 - 4| = | - 3| + | - 2| = 5$

5.5 Hamming-Abstand

Ist ein Abstandsmaß, dass bei Zeichenketten häufig Verwendung findet und gibt die Unterschiedlichkeit zwischen zwei endlichen Elementen aus einem Zeichenraum Σ als Wert wieder.

$$x, y \in \Sigma^n$$

$$d_h(x, y) := |x_i \neq y_i|$$

Es folgt ein Beispiel, wie der *Hamming-Abstand* zwischen zwei Zeichenketten (hier: Binärzahlen) aussehen würde.
$\sum = \{0, 1\}$ als Alphabet.
$x = 01101$
$y = 10001$
$D_h(x, y) = 3$, da wir nur an den Stellen $i = \{1, 2, 3\}$ also in x_1, x_2, x_3 Unterschiede zu y_1, y_2, y_3 vorfinden.

5.6 Lazy Learning Lernverfahren

Stehen im Gegensatz zu den *Eager Learning Lernverfahren*. Das *Lazy Learning* heißt übersetzt faules Lernen und ist eine weitere Kategorie von Lernverfahren. Diese Art von Verfahren sind gekennzeichnet dadurch, dass es genügt alle vorhandenen Trainingsdaten in eine vom System verwendete Datenstruktur zu sichern. Es wird keine Klassifikationsabbildung ermittelt, wie bei dem *Eager Learning*.

6 Nearest-Neighbour Lernverfahren

Gehört zu den faulen Lernverfahren, da initial außer dem Abspeichern der Trainingsdaten kein zusätzlicher Rechenaufwand betrieben wird. Als Ziel des Systems kann man bezeichnen, dass mittels Trainingsdaten versucht wird neue Daten zu klassifizieren bzw. einordnen zu können. (Vgl. Kapitel 3.2.1, lernender Agent auf S. 14)

6.1 Nearest-Neighbour Methode

Einführend werden zur Anschaulichkeit nur zwei mögliche Klassen, denen ein Datenpunkt zugeordnet werden kann, betrachtet. Jedoch kann die Methode des *nearest Neighbour* problemlos auf mehr Klassen erweitert werden.
Unsere Datenpunkte sind Vektoren des $\mathbb{R}^n$, wobei jede Komponente einer konkreten Ausprägung entspricht.
Beispielsweise könnte der Versuch einen Patienten mittels eines Vektors darzustellen, wie folgt aussehen. Indem man jeder Komponente des Vektors eine konkrete Ausprägung des Patienten zuordnet, wie z.B. einer Komponente das Alter oder die Größe der Person, kann auf diese Weise ein Vektor einen Patienten repräsentieren und alle für unser System benötigten Informationen enthalten.
Zu Beginn werden die Mengen M_+ und M_- als Trainingsdaten, s als den zu klassifizierenden Datenpunkt und t als den nächsten Nachbarn von s definiert. Die Funktion *argmin{}* erhält eine endliche Menge als Eingabe und gibt die Stelle an dem ein Minimum auftritt als Rückgabewert zurück. Analog würde für *argmax{}* gelten, dass die Stelle an der ein Maximum auftritt zurückgegeben wird.
Die Idee des *nearest Neighbour* Verfahrens ist, wenn der nächste Nachbarn von s erfolgreich bestimmt werden konnte, kann aufgrund der Ähnlichkeit zwischen beiden Daten-

punkten die Aussage getroffen werden, dass s vermutlich der gleichen Klasse angehört wie t. Dadurch ist unsere Klassifizierung erfolgt.

6.2 Nearest-Neighbour Algorithmus

Der Algorithmus ist wie folgt definiert:

Data: $M_+, M_-, s \in \mathbb{R}^n$
Result: Klassifikation von s
for $x \in M_+ \cup M_-$ **do**
  $d(s, x)$;
end
$t = argmin\{d(s, x)\}$;
if $t \in M_+$ **then**
  return $+$;
else
  return -;
end

Algorithm 2: Nearest Neighbour Algorithm

Als Nächstes wird das *nearest Neighbour* Verfahren anhand eines Beispiels ausgeführt. In Abb. 11 sind zwei Klassen {rot, grün} sind als Trainingsdaten vorgegeben und dazu ein schwarzer Punkt dessen Klasse ermittelt werden soll. Alle Daten sind Teil des Raumes $\mathbb{R}^2$.

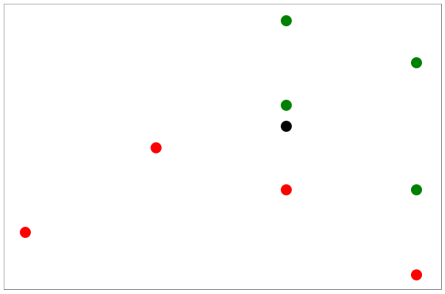

Abbildung 11: Ausgangssituation

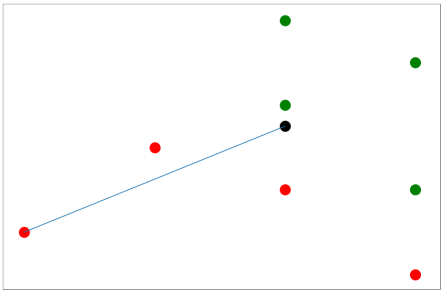

Abbildung 12: Bestimmen des ersten Abstands

Im nächsten Schritt wird die Distanz mithilfe der gewählten Abstandsfunktion (hier: euklidische Norm) zwischen Ausgangspunkt und jedem Punkt innerhalb des Merkmalsraumes bestimmt. In den Abbildungen 12, 13 und 14 als blau gekennzeichnet.

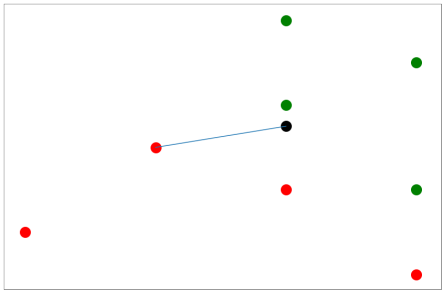

Abbildung 13: Bestimmen des zweiten Abstands

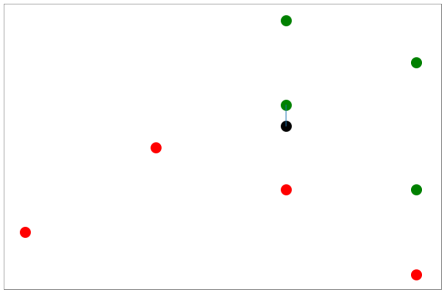

Abbildung 14: Bestimmen des n-ten Abstands

Der gefundene Wert der *argmin{}* Funktion verändert sich nicht mehr, nachdem die minimale Distanz ermittelt wurde. Nach Durchlauf aller n Trainingsdaten kann die *argmin{}*-Funktion dadurch mit Sicherheit den kleinsten Wert zurückliefern.
Im letzten Schritt kann die Klasse des neuen Punktes gesetzt werden, wie in Abb. 14 zu sehen, gehört der nächste Nachbar zur grünen Klasse. Daraus kann man schließen, dass der schwarze Punkt aufgrund seiner Ähnlichkeit zum grünen Punkt auch zur selben Klasse gehört.

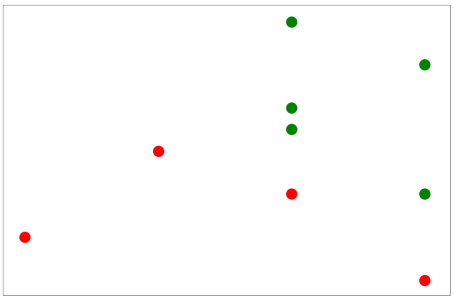

Abbildung 15: Nach Durchlauf des Algorithmus

6.3 Kritik an der Nearest-Neighbour Methode

Das *nearest Neighbour* Verfahren ist ein mächtiges Werkzeug zum Klassifizieren von Daten, da der Algorithmus selbst anhand nicht linear separable Mengen eine Zuordnung treffen kann.[Ert08] Wenn das Lernverfahren jedoch mit dem bereits behandelten *Perzeptron* aus Kapitel 4 verglichen wird, fällt auf, dass der *nearest Neighbour* Algorithmus im $\mathbb{R}^2$ keine lineare Trennlinie bzw. im Raum des $\mathbb{R}^n$ keine trennende Hyperfläche erzeugt. Dies ist insoweit ungünstig, da anhand dieser Trennfunktion für jeden weiteren zu klassifizierenden Punkt weniger Aufwand betrieben werden muss, die zugehörige Klasse zu ermitteln.[Ert08]
Weiterer Kritikpunkt des nearest Neighbour ist, dass im Falle von statistischen Ausreißern in den Trainingsdaten Datenpunkte möglicherweise falsch zugeordnet werden. Ein statistischer Ausreißer bedeutet im Falle des *nearest Neighbour* Algorithmus, dass sich innerhalb der Trainingsdaten ein oder mehr falsch klassifizierte Datenpunkte befinden. Dieses Problem besteht selbst dann, wenn in unmittelbarer Nähe zum Betrachtungspunkt weitere korrekt klassifizierte Trainingsdaten zur Verfügung stehen. Das liegt daran, dass bei dieser Variante des *nearest Neighbours* nur der erste nächste Nachbarn zur Klassifizierung betrachtet wird.[Ert08]
In Abbildung 16 wird diese Ausgangssituation mit fehlerhaften Trainingsdaten beschrieben, um zu verdeutlichen, wie ein einzelner Irrtum das gesamte Verfahren verfälschen kann.

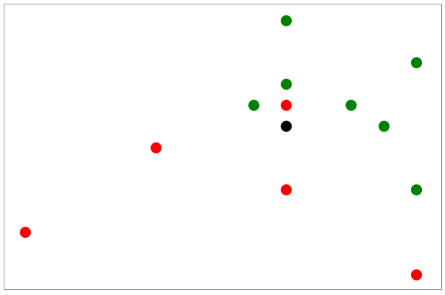

Abbildung 16: Ausgangssituation mit Ausreißer

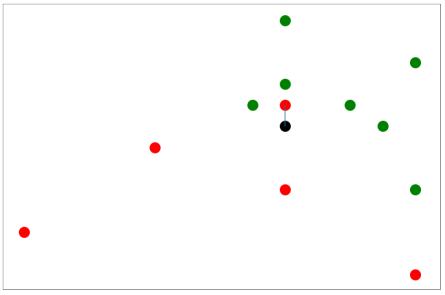

Abbildung 17: Bestimmen des minimalen Abstands

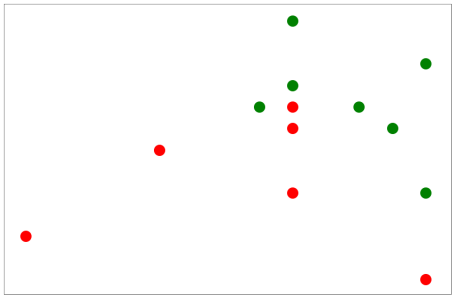

Abbildung 18: Fehlklassifizierung nach Durchlauf

Man erkennt, dass sich ein rot klassifizierter Punkt in unmittelbarer Nähe von mindestens drei Punkten der grünen Klasse befindet. Auf diese Daten wird nun analog zu Kapitel 6.2 das Lernverfahren des *nearest Neighbour* angewendet.
Es werden erneut alle Datenpunkte auf minimalen Abstand zum (schwarzen) Ausgangspunkt überprüft. Die *argmin{}*-Funktion liefert für diesen Fall den statistischen Ausreißer als Referenzpunkt zurück (siehe Abb. 17). Abschließend erfolgt die Klassifizierung zur Klasse rot(siehe Abb. 18).
Demnach ist es dem System nicht möglich zu wissen, ob ein Punkt aus den Trainingsdaten fehlerhaft oder korrekt ist und kann auch für diesen Fall nicht auf andere verfügbare Daten zurückgreifen. Für dieses Problem ist die *nearest Neighbour* Methode, wie sie in diesem Kapitel beschrieben ist, nicht mächtig genug.

7 k-nearest Neighbour Methode

Ist eine Variante des *nearest Neighbour*. Bei der Verwendung des *k-nearest Neighbour* Algorithmus wird statt einem nächsten Nachbarn eine k-große Gruppe an nächsten Nachbarn betrachtet. Die Entscheidung zu welcher Klasse ein neuer Datenpunkt gehören soll, erfolgt durch einen Mehrheitsentscheid innerhalb dieser ermittelten k-nächsten Nachbarn.[Ert08]
Es werden die gleichen Bezeichner für den Algorithmus verwendet, die auch für den *nearest Neighbour* angewandt wurden und um die Variable V erweitert.
V bezeichnet die k-nächsten Nachbarn von s innerhalb der Trainingsdaten.

7.1 k-nearest Neighbour Algorithmus

Der Algorithmus ist wie folgt definiert:

Data: $M_+, M_-, s \in \mathbb{R}^n$
Result: Klassifikation von s
V ={k-nächste Nachbarn von s};
if $|M_+ \cap V| > |M_- \cap V|$ **then**
  return $+$;
end
if $|M_+ \cap V| < |M_- \cap V|$ **then**
  return -;
else
  return *Random(+,-)*;
end

Algorithm 3: k-Nearest Neighbour Algorithm

Im Folgenden wird ein Beispiel für den *k-nearest Neighbour* behandelt. Es stehen in Abb. 19 rot und grün klassifizierte Punkte als Trainingsdaten zur Verfügung und zu bestimmen ist die Klasse des schwarzen Punktes.

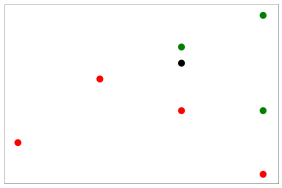

Abbildung 19: Ausgangssituation

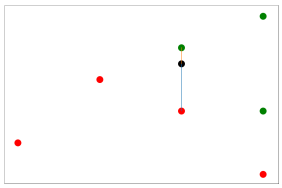

Abbildung 20: Fall $k = 2$

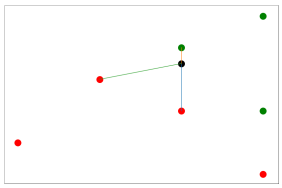

Abbildung 21: Fall $k = 3$

Für den Fall $k = 1$, also die Betrachtung nur eines einzelnen nächsten Nachbars, kann auf das Ergebnis aus Kapitel 6.2 verwiesen werden.
Der Fall $k = 2$ kann aus Abb. 20 entnommen werden. Hierbei wird die Menge betrachteter nächster Nachbarn auf zwei erhöht und führt dazu, dass bei Anwendung des Algorithmus die zwei kleinsten Abstände über die Abstandsfunktion zu ermitteln sind. Die Variable V enthält nach Bestimmen aller Distanzen jeweils einen Punkt aus der roten und grünen Klasse.

$$V = \{\text{rot,grün}\}$$

Als Nächstes werden die Schnittmengen von V mit den jeweiligen Klassen ausgewertet. Es folgt also:

$$|V \cap \text{Rot}| = 1 \text{ und } |V \cap \text{Grün}| = 1$$

Für die Klasse des schwarzen Punktes bedeutet dies, dass seine Klasse zufällig aus der Menge {Rot, Grün} ausgewählt werden muss.
In Abb. 21 wird die Anzahl nächster Nachbarn mit Stimmrecht erneut um eins erhöht. Die Verbindungen zwischen dem schwarzen Punkt und allen anderen zeigen, welche der Punkte im Merkmalsraum die kleinsten Abstände aufweisen. Dadurch enthält V = {rot, rot, grün} und daraus folgt demnach für die Schnittmengen:

$$|V \cap \text{Rot}| = 2 \text{ und } |V \cap \text{Grün}| = 1$$

Für die Zugehörigkeit des schwarzen Punktes ergibt sich somit eine Klassifizierung in Richtung der roten Klasse.
Auf die gleiche Weise kann $k \to n$ weitergeführt werden.

7.2 Anwendung bei mehr als zwei Klassen

Es ist ebenfalls möglich den *k-nearest Neighbour* nicht nur für zwei Klassen anzuwenden, sondern auch auf endlich viele Klassen zu erweitern. Hier steht man jedoch vor der Herausforderung, falls die Anzahl Klassen ansteigt, wächst im gleichen Zug die Anzahl der benötigten Trainingsdaten stark an.[Ert08]
Noch weiter verschärft wird dieses Szenario, wenn nicht genug Trainingsdaten oder Speicherressourcen zur Verfügung stehen. Steigt die Anzahl der Klassen weiter, ist eine diskrete Klassifikation nicht mehr ideal und eine stetige Abbildungsfunktion ist für die Anwendung zweckmäßiger.[Ert08]
Auf die Betrachtung der Laufzeit wird in Kapitel 9 eingegangen.

7.3 Kritik an der k-nearest Neighbour Methode

Analog gilt ebenfalls für die *k-nearest Neighbour* Methode, dass die Anwendung des Algorithmus allein keine Trennebene erzeugen kann. Andererseits kann festgehalten werden, dass der Algorithmus eine schwächere Anfälligkeit gegenüber statistischen Ausreißern aufweist. Grund dafür ist die Betrachtung einer größeren Stichprobe der Trainingsdaten.
Ein weiterer Kritikpunkt lautet, dass bei wachsendem k, also bei Betrachtung eine größer werdenden Gruppe nächster Nachbarn, die Anzahl weiter entfernter Nachbarn einen nicht unwesentlichen Einfluss auf den Mehrheitsentscheid und somit auf die Klassifizierung eines Datenpunktes haben. Dies steht im Widerspruch zu dem zentralen Ähnlichkeitsbegriff (Vgl. Kapitel 5.1), welcher geringe Abstände als Ähnlichkeitsmaß definierte.
Eine Möglichkeit zur Verbesserung nennt Wolfgang Ertel, die Stimmen der k-nächsten Nachbarn mit Gewichten zu versehen. Die Stimmgewichte der nächsten Nachbarn sollen mit steigendem Abstand zum zu klassifizierenden Punkt quadratisch abnehmen, um dadurch weniger Einfluss ausüben zu können.[Ert08]
Die Bestimmung der Stimmgewichte w ist wie folgt definiert:
x ist der zu klassifizierende Punkt und x_i ist der i-te nächste Nachbar von x.

$$x, x_i \in \mathbb{R}^n$$

$$w_i := \frac{1}{1 + \alpha d(x, x_i)^2}$$

α ist ein Konstante und bestimmt, wie schnell Gewichte mit größer werdendem Abstand zum Ausgangspunkt abnehmen sollen.[Ert08] Anhand dieser Formel erkennt man, dass mit wachsendem Abstand der Einfluss bzw. das Stimmgewicht weit entfernter Punkte asymptotisch gegen null geht und ist somit ein zielführender Lösungsansatz.[Ert08]

8 Voronoi-Diagramme

Im Kapitel 7.3 (Titel: Kritik an der *nearest Neighbour* Methode) wurde bereits erwähnt, dass der *nearest Neighbour* Algorithmus im Vergleich zum *Perzeptron* keine (lineare) Trennebene erzeugen kann. Es ist jedoch möglich durch Hinzunahme eines weiteren Verfahrens nicht nur eine Trennung vorzunehmen, sondern auch eine wesentlich komplexere Trennebene aus den gegebenen Trainingsdaten zu schaffen, um so die benötigte Rechenzeit zur Klassifizierung zu vermindern.[Ert08] Wie bereits bei den *Eager Learning* Algorithmen erklärt wurde, benötigt die Bestimmung dieser Abbildungsfunktion den Großteil der Rechenzeit. Ein Verfahren aus der Kategorie des *Lazy Learnings* wird auf diese Weise um ein Verfahren, dass man dem *Eager Learning* zuordnen würde, erweitert.
Die Bestimmung der Trennebene ist daher mit einem Zusatzaufwand verbunden, jedoch wird der Merkmalsraum um eine nützliche Eigenschaft ergänzt. Denn mithilfe des *Voronoi-Diagramms* wird um jeden Punkt ein beliebig komplexes Polygon erzeugt. Innerhalb jedes Polygons gilt, dass der nächste Nachbar genau jener Punkt ist, um den das Polygon anfangs gebildet wurde.[Ert08]
Auf die konkrete Bestimmung eines *Voronoi-Diagramms* für eine endliche Datenmenge wird nur beschränkt eingegangen, da lediglich das Ergebnis für unsere Anwendung von Interesse ist.

8.1 Voronoi-Diagramm Beispiel

Als Beispiel wird die folgende Ausgangssituation (siehe Abb. 22) betrachtet. Zwei Klassen als grüne Plus und rote Minus gekennzeichnet dienen als Menge von Trainingsdaten. Für diese Menge soll nun als Nächstes das zugehörige *Voronoi-Diagramm* genauer untersucht werden.
Auf den ersten Blick erkennt man, dass die Trennung in keinem Falle durch eine lineare Gerade erzeugt werden kann, da sich links und rechts von grünen Klasse jeweils Punkte der roten Klasse befinden. Dadurch kann man die Aussage treffen, dass die vorgegebene Trainingsdatenmenge nicht Teil der linear separablen Mengen ist und demnach zu keiner Lösung bei Verwendung der *Perzeptron*-Lernregel führen würden.

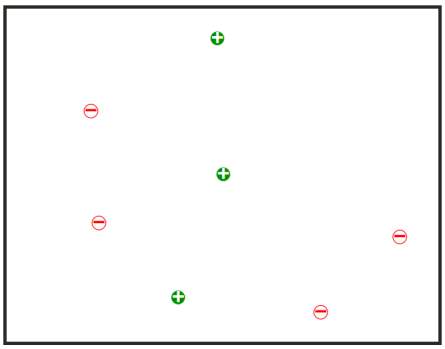

Abbildung 22: Zu klassifizierende Punkte

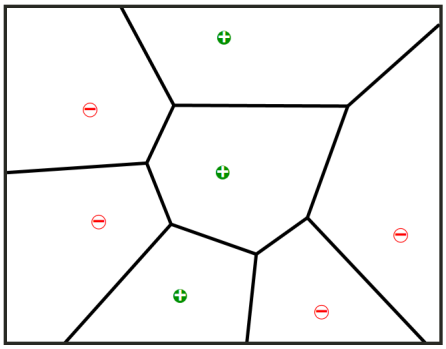

Abbildung 23: Eintragen der Polygone

In Abbildung 23 fällt auf, dass unabhängig von der Klasse eines Punktes eine Gerade orthogonal auf der Mitte einer imaginären Verbindung zwischen dem betrachteten Punkt

und seinen umliegenden Nachbarn liegt. Diese Geraden bilden, wenn für alle näheren Nachbar eingezeichnet, das Polygon in welchem er sich befindet. Dies lässt sich auf die Eigenschaft des *Voronoi-Diagramms* zurückführen, dass innerhalb des Polygons jeder Punkt näher am zentralen Punkt gelegen ist, als an jedem anderen Punkt innerhalb des Merkmalsraums.

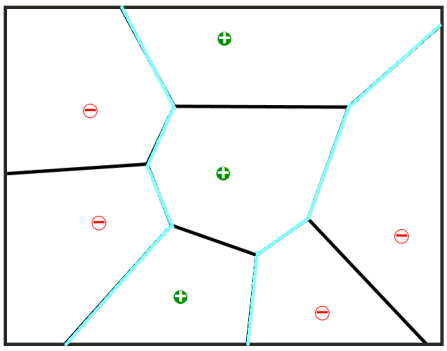

Abbildung 24: Trennung von zwei Klassen

Nachdem das *Voronoi-Diagramm* für eine Menge von Trainingsdaten ermitteln wurde, kann eine Trennebene eingezeichnet werden (siehe Abb. 24). Die Trennung hat die Eigenschaft, dass sich innerhalb Trennebene nur Datenpunkte einer Klasse befinden. Wird anschließend der Versuch unternommen eine Klassifizierung durchzuführen, kann man dies anhand der Trennlinie mit geringem Aufwand unternehmen, indem geprüft auf welcher Seite der Trennung sich der zu bestimmende Punkt befindet. Infolgedessen wird die Berechnungszeit gestrichen, die man ohne *Voronoi-Diagramm* für das Ermitteln aller Abstände zum Betrachtungspunkt genutzt hätte.
Die Diskussion in welchem Verhältnis die Berechnung des *Voronoi-Diagrammms* zu der Zeit, die man für jede Ausführung einspart, steht, würde den Rahmen dieser Ausarbeitung übersteigen und wird nicht weiter ausgeführt.
Man kann folglich mithilfe dieses Beispiels den Beweis erbringen, dass die *nearest Neighbour* Methode ein mächtigeres Klassifizierungsverfahren ist als das *Perzpetron*.

9 Laufzeiten der nearest-Neighbour Methoden

Laut Wolfgang Ertel: „Gibt es kein [maschinelles] Lernverfahren, dass so schnell lernt"[Ert08, S. 213], wie die *nearest Neighbour* Methode, da in diesem Fall nur Daten gesichert werden müssen.
Wie der Rechenaufwand für beide vorgestellten Methoden aussieht wird im Folgenden gezeigt.

9.1 Laufzeit nearest-Neighbour Algorithmus

Bei der Betrachtung des *nearest Neighbour* Algorithmus fällt auf, dass jeder enthaltene Punkt der Trainingsdaten auf eine minimale Distanz zum Betrachtungspunkt untersucht werden muss. Somit gilt für die Laufzeit:

$$\mathcal{O}(n)$$

9.2 Laufzeit k-nearest-Neighbour Algorithmus

Der *k-nearest Neighbour* Algorithmus hat ähnlich, wie der *nearest Neighbour*, eine Grundlaufzeit von n, jedoch entsteht bei der Iteration über alle ermittelten nächsten Nachbarn ein zusätzlicher Rechenaufwand der Größe k. Dadurch ergibt sich für die Laufzeit:

$$\mathcal{O}(n + k)$$

Bei den *nearest Neighbour* Methoden gibt es keine Fallunterscheidung in bessere oder schlechtere Ausgangssituationen, da bei keiner Konstellation der Trainingsdaten Unterschiede in Ablauf entstehen.
Nicht außer Acht gelassen werden darf die ist Tatsache, dass mit n nur die Anzahl verfügbarer Trainingsdaten gemeint ist und nicht verwechselt werden darf mit der Anzahl Dimensionen aus denen ein Punkt aus dem Merkmalsraum geformt ist.

Literatur

[Ert08] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz*. Springer Fachmedien, Wiesbaden, 2016 edition, 2008.

[Koh05] Wolfgang Kohn. *Statistik: Datenanalyse und Wahrscheinlichkeitsrechnung*. Springer-Verlag, 2005.

[MS17] Vishal Maini and Samer Sabri. Machine Learning for Humans. `https://medium.com/machine-learning-for-humans/why-machine-learning-matters-6164faf1df12`, 2017. [Online; Abgerufen am 13.08.2019].