Visualisierung sowie Auswertung medizinischer Sensordaten unter Zuhilfenahme unterschiedlicher Algorithmen des maschinellen Lernens

Bachelorarbeit zur Erlangung des Bachelor-Grades
Bachelor of Science im Studiengang Medieninformatik
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Krystian Schindler
Matrikel-Nr.:	011 097 598 46
Adresse:	Oststraße 17
	51766 Engelskirchen
	krystian.schindler@smail.th-koeln.de

eingereicht bei:	Prof. Dr. Heide Faeskorn-Woyke
Zweitgutachter/in:	Prof. Dr. Dietlind Zühlke

Gummersbach, 26.10.2020

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum

Rechtsverbindliche Unterschrift

Kurzfassung/Abstract

Aufbauend auf einer vorherigen Arbeit, die sich mit der Implementierung einer Komponente zum Auslesen medizinischer Sensordaten mithilfe eines Arduino und eines Raspberry Pi befasst hat, beschäftigt sich diese Arbeit mit der Visualisierung sowie Auswertung der durch das System gesammelten Daten. Das Ziel dieser Arbeit ist es, über einen Zeitraum von etwa drei Monaten mithilfe der Komponente Daten zu sammeln und diese Daten in einem sinnvollen Kontext visuell darzustellen. Zudem sollen diese Daten mithilfe unterschiedlicher Algorithmen des Maschinellen Lernens ausgewertet werden, um mögliche Muster und Zusammenhänge erkennen zu können. In diesem Kontext konnte die Hypothese aufgestellt werden, dass ein Zusammenhang zwischen der Körpertemperatur und der Sauerstoffsättigung im Blut besteht.

Building on a previous work that dealt with the implementation of a component for reading medical sensor data using an Arduino and a Raspberry Pi, this work deals with the visualization and analysis of the data collected by the system. The goal of this thesis is to collect data over a period of about three months using the component and to visualize this data in a meaningful context. In addition, the data will be evaluated using different machine learning algorithms in order to identify possible patterns and correlations. In this context, it was possible to hypothesize that there is a correlation between body temperature and oxygen saturation in the blood.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Themen künstliche Intelligenz sowie maschinelles Lernen nehmen einen immer höheren Stellenwert im alltäglichen Leben ein. Sei es ein Algorithmus in einem Online-Shop, der einem je nach angesehenen Artikeln weitere vorschlägt, die interessant sein könnten, oder ein intelligenter Chatbot eines Unternehmens, der präzise auf Anfragen eines Nutzers antworten kann. Ein aktuelles Beispiel aus der Gaming-Branche[1] legt die Aktualität des Themas nahe. Anfang des Jahres 2020 wurde Sonys Patent einer künstlichen Intelligenz zur Verbesserung des Spielerlebnisses der neuen Konsolengeneration publiziert. Demnach soll die künstliche Intelligenz vom Spielerverhalten in einem bestimmten Videospiel lernen und falls der Spieler in einer Situation nicht weiter kommt, mögliche Lösungsansätze vorschlagen, die sie aus dem Verhalten derjenigen Spieler gelernt hat, die diese Situation bereits gemeistert haben.

Das grundlegende Prinzip des maschinellen Lernens lässt sich demnach wie folgt beschreiben: Es werden Daten gesammelt und diese Daten sollen mithilfe ein oder mehrerer Algorithmen ausgewertet werden, um daraus relevante Informationen sowie Zusammenhänge innerhalb der Daten zu entnehmen. Dies könnten beispielsweise Daten über zum Kauf stehenden Immobilien sein, um zu untersuchen, ob ein Zusammenhang zwischen dem Preis einer Immobilie und dem Fußbodenbelag einer Küche besteht. Um ein besseres Verständnis für die vorliegenden Daten zu haben, werden diese häufig durch Visualisierungen veranschaulicht. Da das maschinelle Lernen nicht auf eine bestimmte Thematik beschränkt ist, lassen sich damit sehr unterschiedliche Arten von Daten, so auch zum Beispiel medizinische Daten untersuchen.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist das Sammeln medizinischer Sensordaten mithilfe des im vorangegangenen Praxisprojekts implementierten Systems sowie die Visualisierung und Auswertung der gesammelten Daten. Auf das Praxisprojekt soll kurz zusammenfassend im nächsten Unterkapitel eingegangen werden. Im Folgenden soll ein Überblick über die Anforderungen an die Arbeit gegeben werden.

1. Die medizinischen Sensordaten sollen visuell dargestellt werden können.
2. Die Komponente zur Visualisierung soll eigenständig sein, sodass mit ihr auch Daten anderen Ursprungs visuell dargestellt werden können.
3. Es soll eine Auswertung der Sensordaten mithilfe von Algorithmen des Maschinellen Lernens erfolgen.

1.2 Einblick in das vorangegangene Praxisprojekt

Wie bereits erwähnt baut diese Arbeit auf den Ergebnissen des vorangegangenen Praxisprojekts auf, welches sich damit beschäftigte, medizinische Sensordaten mithilfe eines

Raspberry Pi und eines Arduino aufnehmen zu können. Abbildung 1 zeigt das für das Praxisprojekt verwendete Architekturdiagramm. Es sollten Daten von drei unterschiedlichen Sensoren von einem Arduino aufgenommen werden und zur Weiterverarbeitung an einen auf dem Rasperry Pi befindlichen node.js-Server geschickt werden, der die aufgenommenen Daten persistent in einer Datenbank speichert und diese auch abrufen kann. Für diese Arbeit wird davon ausgegangen, dass bereits eine Datenbank mit den zugehörigen Daten existiert, weshalb ein näheres Eingehen auf das Praxisprojekt nicht erforderlich ist.

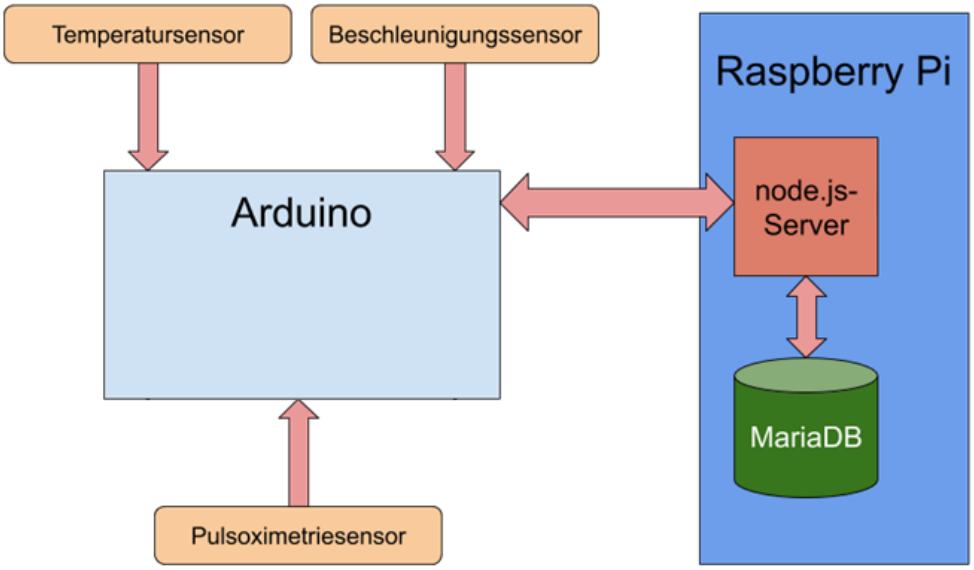


Abbildung 1: Architekturdiagramm des Praxisprojekts

1.3 Struktur der Arbeit

Die folgende Arbeit ist in folgende Kapitel unterteilt:

Kapitel 2 befasst sich mit den Normwerten der gesammelten medizinischen Daten, den für das Projekt relevanten Grundlagen zu den in dieser Arbeit genutzten Visualisierungsformen sowie den genutzten Algorithmen des Maschinellen Lernens. Zudem werden die Funktionalitäten der Algorithmen anhand von Beispielen erklärt und es gibt eine kurze Einführung in die Struktur des Servers.

Kapitel 3 befasst sich mit der Implementierung der Visualisierungskomponente sowie deren zu erfüllenden Anforderungen.

Kapitel 4 zeigt anhand der gesammelten Daten die Anwendung der genutzten Algorithmen und geht auf Besonderheiten der Algorithmen ein.

Kapitel 5 fasst die Ergebnisse der Arbeit kurz zusammen und geht, wenn vorhanden, auf die dabei entstandenen Probleme ein.

2 Grundlagen

2.1 Medizinische Daten und Server

Im Rahmen dieser Arbeit wurden medizinische Sensordaten unter Zuhilfenahme des im Praxisprojekt erstellten Systems gesammelt, um im Folgenden visualisiert und ausgewertet werden zu können. Der derzeitig noch immer anhaltende Pandemiestatus ließ allerdings das Sammeln der Daten nur bedingt zu. Innerhalb von drei Monaten konnten von vier Probanden Daten gesammelt werden, wobei nur von zwei im Haushalt lebenden Personen die Datenaufnahme regelmäßig stattfand. Dies gestaltete sich insbesondere für die Auswertung der Daten als schwierig, da die Vielfältigkeit der Daten nicht gegeben war.

Für diese Arbeit wurden folgende Daten erfasst:

1. Körpertemperatur
2. Körperposition
3. Sauerstoffsättigung im Blut
4. Körperform

2.1.1 Körpertemperatur

Wie in Abbildung 2 erkennbar, unterscheidet man bei der Körpertemperatur zwischen Körperkern- und Körperschalentemperatur. Die Abbildung gibt auch Auskunft über den Normwert in den jeweiligen Körperzonen. Für diese Arbeit ist der Normwert der Körpertemperatur in den Fingern relevant, da die Temperatur über das Umfassen eines Temperatursensors mit den Fingern gemessen wird. Dieser liegt bei ca. 28 ° C.

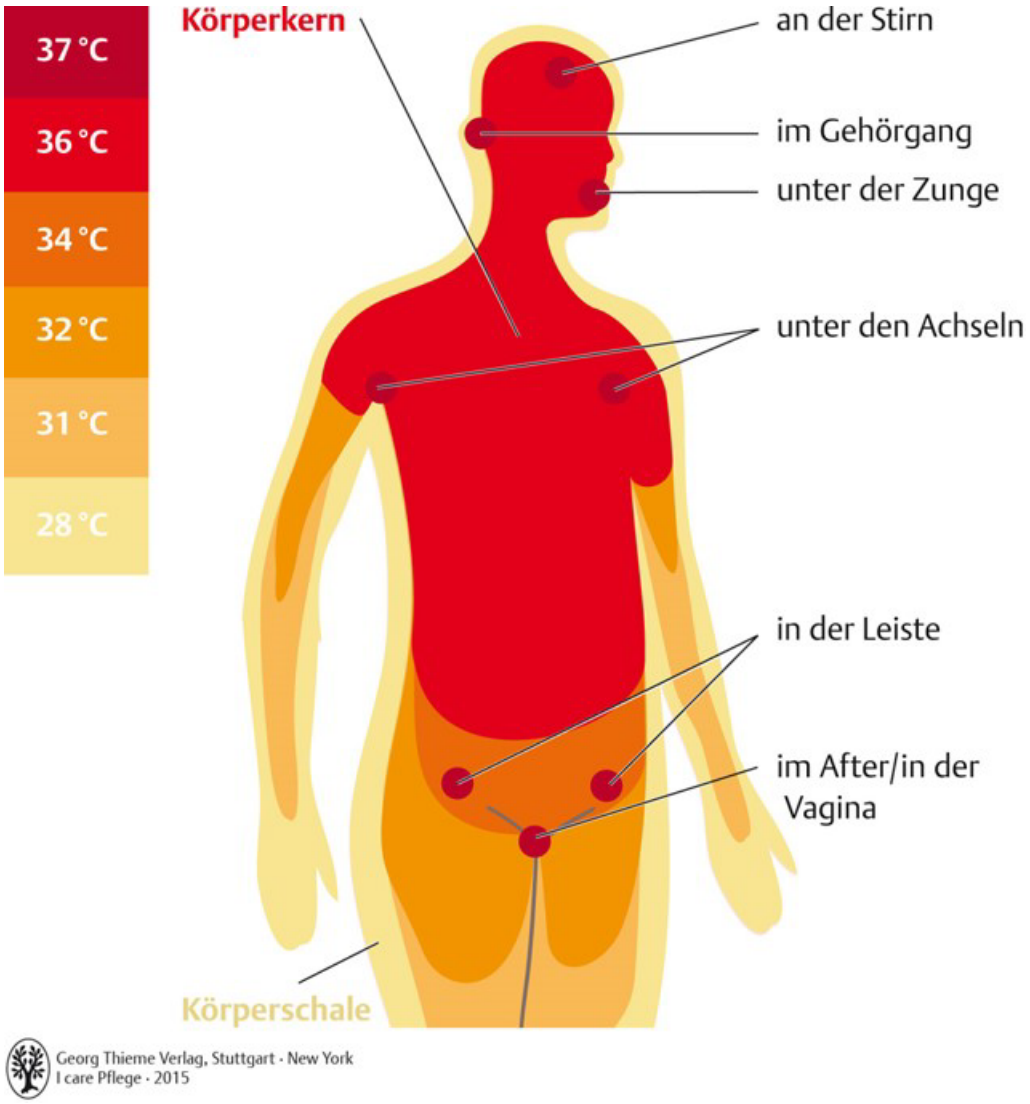


Abbildung 2: Körpertemperaturzonen [3]

2.1.2 Körperposition

Die Körperposition wird mithilfe eines 3-Achsen-Gyroskops erfasst. Dieser Wert gibt Auskunft darüber, in welcher Körperposition die anderen Werte gemessen werden. Dabei wurde zwischen den folgenden Positionen unterschieden:

1. Auf dem Bauch liegend
2. Auf dem Rücken liegend
3. Auf der rechten Seite liegend
4. Auf der linken Seite liegend
5. Sitzend/Stehend
6. Unbekannt

2.1.3 Sauerstoffsättigung im Blut

Die Sauerstoffsättigung im Blut wird mithilfe eines Pulsoximetriesensors erfasst. Der Normalbereich liegt dabei zwischen 0.94 und 0.98, was einem Wertebereich von 94% bis 98% entspricht[2].

2.1.4 node.js Server

Der node.js Server agiert in dieser Arbeit als zentrale Komponente für Datenaufnahme sowie -wiedergabe. Auf die Funktionsweise der Datenaufnahme wird jedoch nicht weiter eingegangen, da dies Teil des Praxisprojekts war und somit irrelevant für diese Arbeit ist. Mit dem node.js Server werden bei einer Anfrage mit einem Browser an die zugehörige URL Daten aus der Datenbank abgerufen, an ein Python-Skript zur Verarbeitung gesendet und dem Browser mit dem Ergebnis auf seine Anfrage geantwortet.

Bevor näher auf die URLs eingegangen wird, soll jedoch das Datenbankschema kurz behandelt werden. Die Daten liegen auf der Datenbank in folgendem Schema vor:

- `user (userID, userName, geschlecht, aktKoerperform)`
- `meddata (userID, aufnahmeDatum, koerperform, kTemperatur, kPosition, sauerstoffsaettigung)`

Die `userID` ist der Primärschlüssel der Tabelle `user` und wird der Tabelle `meddata` als Fremdschlüssel übergeben. Die Körperform entspricht dem nach der WHO klassifiziertem Body Mass Index[4] und unterteilt sich in sechs mögliche Gruppen. Diese wären: Untergewicht, Normalgewicht, Übergewicht, Adipositas Grad I, Adipositas Grad II und Adipositas Grad III. Da sich die Körperform ändern kann, wird die aktuelle Körperform in der Tabelle `user` festgehalten. Diese kann jederzeit geändert werden. Bei der Aufnahme der Daten wird die Körperform einer Person aus dem zugehörigen Datensatz der Nutzerdaten ermittelt und dem Datensatz hinzugefügt. Damit wird erreicht, dass Personen bei einer Änderung ihrer Körperform ihren Nutzerdatensatz ändern können, ohne die Körperform in den bereits aufgenommenen Datensätzen zu ändern, sodass Verfälschungen in den Daten vermieden werden.

Der Server reagiert auf folgende URLs:

1. `raspithkoeln:8000/plotUser?name=xxx`
2. `raspithkoeln:8000/plotMultipleUser`

Dabei entspricht `raspithkoeln` dem Hostnamen des verwendeten Raspberry Pi, wenn die URL von einem externen System aufgerufen wird. Wenn das Skript des Servers intern aufgerufen wird, muss `raspithkoeln` durch `localhost` ersetzt werden. Die Zahl `8000` entspricht der Nummer des Ports. Mit der URL `raspithkoeln:8000/plotUser?name=xxx` wird die visuelle Darstellung eines Nutzers angefordert, wobei der Query-Parameter `name` dem in

der Datenbank hinterlegten `userName` entspricht. Eine visuelle Darstellung der Sensordaten aller Nutzer kann über die URL `raspithkoeln:8000/plotMultipleUser` aufgerufen werden, wobei die Datenpunkte der verschiedenen Nutzer durch unterschiedliche Farben voneinander abgegrenzt werden können.

2.2 Visualisierung

Es gibt unterschiedliche Möglichkeiten, Daten visuell darzustellen.[5] Im Nachfolgenden sollen die in der Arbeit verwendeten Darstellungsformen vorgestellt und begründet werden.

1. Histogramm - Zur Darstellung der Häufigkeit der Daten gruppiert nach der Körperposition
2. Barchart - Zur Darstellung der Häusigkeit der Daten gruppiert nach dem Geschlecht
3. Scatterplot - Verwendete Möglichkeit in der Visualisierungskomponente
4. Boxplot - Zum Vergleich und Ermittlung der der Daten unterschiedlicher Probanden

2.2.1 Histogramm

Bei einem Histogramm handelt es sich um eine Möglichkeit der grafischen Darstellung für Häufigkeitsverteilungen. Die x-Achse entspricht dabei einem quantitativen Merkmal. Auf der y-Achse hingegen steht die absolute oder relative Häufigkeit. Der Flächeninhalt der Balken eines Histogramms gibt Auskunft über dessen absolute/relative Häufigkeit und die Balken des Histogramms berühren sich. Das Beispiel in Abbildung 3 zeigt dabei die absolute Häufigkeit von erreichten Punkten in einer Klausur. Dabei wurden die Klausurpunkte in Klassen unterteilt, die jeweils in einem Balken zu sehen sind. Eine Klasse deckt in diesem Beispiel ein Intervall von 20 Punkten ab.

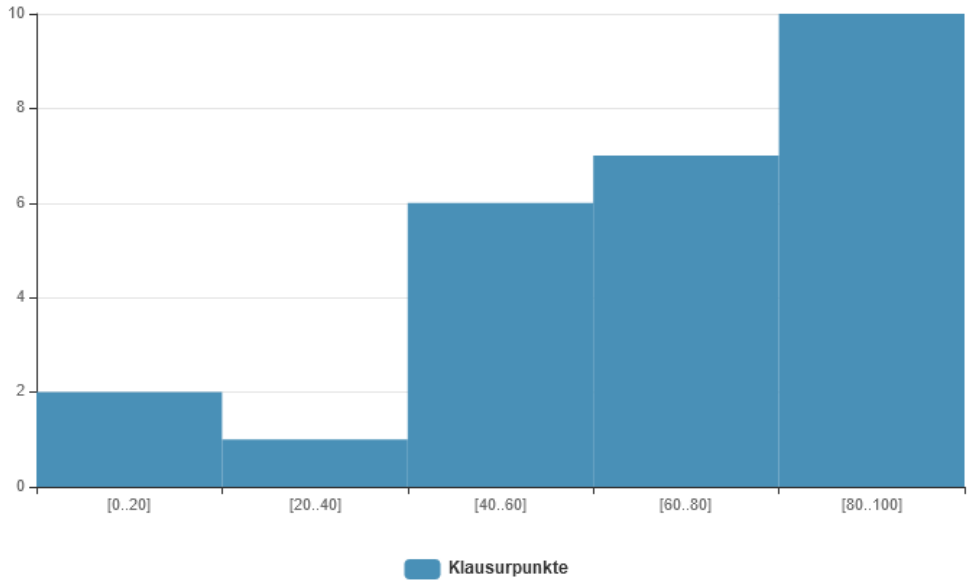


Abbildung 3: Beispiel für ein Histogramm

2.2.2 Barchart

Ein Barchart ist dem Histogramm sehr ähnlich. Der wesentliche Unterschied besteht darin, dass bei einem Barchart die x-Achse einem kategorischen Merkmal entspricht und sich die Balken nicht berühren. Das Beispiel in Abbildung 4 stellt die monatlichen Ausgaben (in €) von fünf Personen und den Zweck der Ausgaben gegenüber. Durch die Visualisierung lässt sich somit nicht nur ermitteln, wie viel eine Person für Lebensmittel oder Luxusgüter ausgegeben hat, man kann auch die einzelnen Personen auf einen Blick miteinander vergleichen und mögliche Rückschlüsse daraus ziehen.

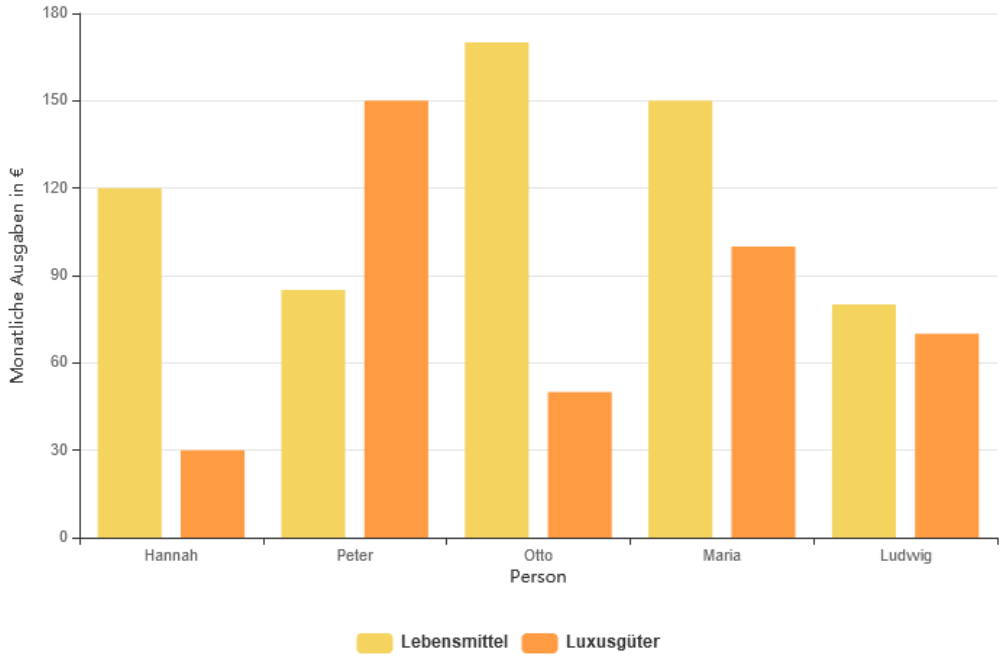


Abbildung 4: Beispiel für ein Barchart

2.2.3 Scatterplot

Bei einem Scatterplot werden Wertepaare zweier statistischer Merkmale gegenübergestellt. Dabei repräsentiert die x-Achse eine unabhängige und die y-Achse die abhängige Variable. Die einzelnen Datenpunkte werden als Punkte in einem Koordinatensystem dargestellt. Das Beispiel in Abbildung 5 stellt das Verhältnis von Alter und Körpergröße einer Person in Abhängigkeit von dessen Geschlecht dar. Im Gegensatz zum Line Chart, bei dem die Punkte durch Linien verbunden sind, hat beispielsweise die Änderung einer Körpertemperatur innerhalb eines Zeitintervalls keinen großen Zusammenhang und wird daher als separater Datenpunkt visuell dargestellt und deshalb bei der Visualisierungskomponente verwendet.

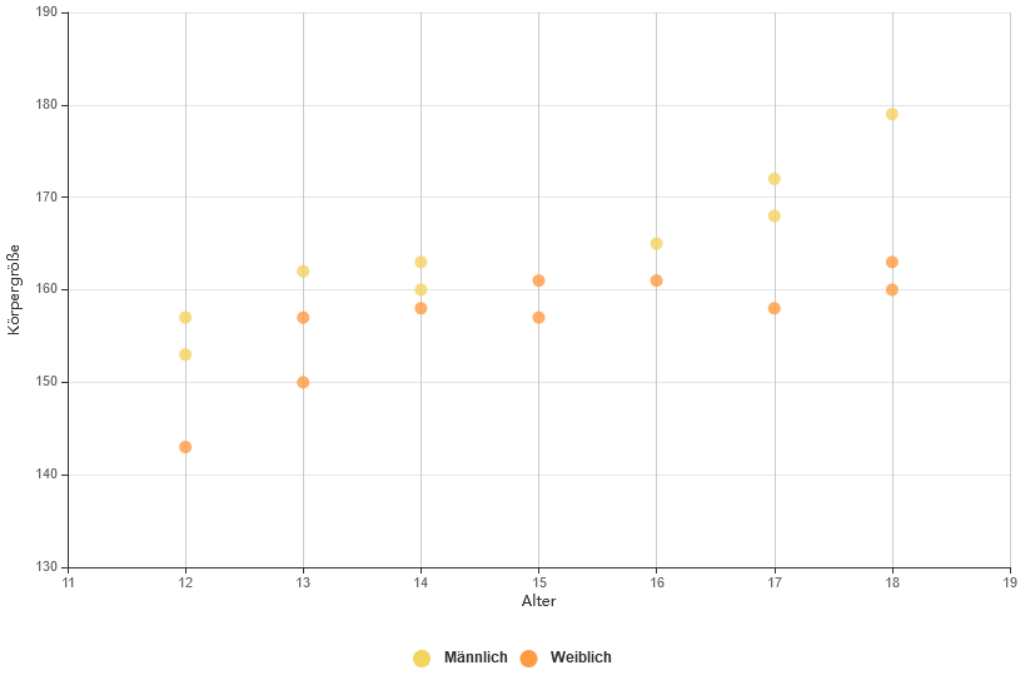


Abbildung 5: Beispiel für einen Scatterplot

2.2.4 Boxplot

Ein Boxplot fasst die wichtigsten Lage- und Streuungsmaße in einem Diagramm übersichtlich zusammen. Dabei markiert die untere beziehungsweise obere Grenze der Box das untere beziehungsweise obere Quartil. Die T-förmigen Linien unterhalb und oberhalb der Box werden als unterer beziehungsweise oberer Whisker bezeichnet. Oftmals markieren die Linien den Wert bis zur 1,5-fachen Länge des IQR (Interquartile Range) der Box. Die Länge des IQR wird berechnet, indem man vom oberen Quartil das untere Quartil subtrahiert. Wenn es weiterhin Punkte gibt, die außerhalb des Boxplots existieren, werden sie als Ausreißer bezeichnet und mit Kreisen außerhalb des Boxplots auf dem Diagramm dargestellt. Ein Boxplot kann sowohl vertikal als auch horizontal gezeichnet werden. Das Beispiel in Abbildung 6 zeigt das Einkommen einer Personengruppen in Abhängigkeit von deren Schulbildung.

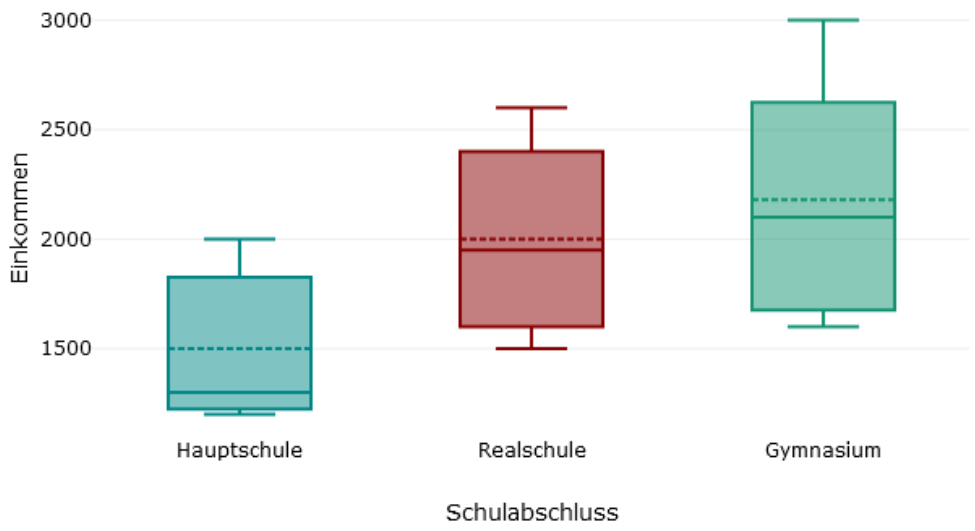


Abbildung 6: Beispiel für einen Boxplot

Es sollte zudem erwähnt werden, dass die in diesem Unterkapitel aufgeführten Beispiele nicht zwingend der Realität entsprechen und nur zur Veranschaulichung der Diagramme dienen sollen.

2.3 Maschinelles Lernen

Um das maschinelle Lernen etwas verständlicher zu machen, soll zunächst grundlegend geklärt werden, was maschinelles Lernen überhaupt ist, und im Anschluss die Funktionsweise von einigen Algorithmen erläutert werden. Dabei wird jedoch auf die mathematische Herkunft verzichtet, da dies den Rahmen dieser Arbeit sprengen würde.

Das maschinelle Lernen ist ein Teilgebiet der künstlichen Intelligenz und ermöglicht einem System, selbstständig Muster und Zusammenhänge aus großen Datenmengen zu erkennen und sich zu verbessern, ohne dafür explizit programmiert worden zu sein. Zu dem Aufgabenbereich des maschinellen Lernens gehören unter anderem die Vorhersage von Werten auf Basis der analysierten Daten, die Berechnung von Wahrscheinlichkeiten bestimmter Ereignisse und die Klassifizierung der Daten in bestimmte Bereiche.[6]

Das maschinelle Lernen wird im Wesentlichen in drei Bereiche unterteilt: Supervised Learning, Unsupervised Learning und Reinforcement Learning.[7]

Das Supervised Learning - oder auch überwachtes Lernen - nimmt sich die Vorhersage einer bestimmten Variable zum Ziel. Man könnte als Beispiel die Klassifizierung von Bildern in Mensch, Tier oder Haus nehmen. Als Input bekommt das System viele Bilder von Menschen, Tieren und Häusern, die bereits von Menschenhand klassifiziert sind, und lernt aufgrund der bisherigen Klassifizierung auch bei neuen Bildern, zwischen Mensch, Tier und Haus zu unterscheiden. Der Mensch als Komponente bringt also dem System vorher bei,

dass das Bild eines Hundes beispielsweise der Klasse Tier angehört. Nachdem das System gelernt hat, welche Merkmale die unterschiedlichen Klassen besitzen, versucht es selbst aufgrund neuer Bilder die Vorhersage zu treffen, ob es sich dabei um Mensch, Tier oder Haus handelt.

Das Unsupervised Learning - oder auch unüberwachtes Lernen - nimmt sich zum Ziel, (eventuell versteckte) Muster und Gruppen in einer Datenmenge zu erkennen. Dabei wird im Gegensatz zum Supervised Learning nicht mitgeteilt, was der Algorithmus konkret herausfinden soll. Die Datenmenge wird übergeben und es wird selbstständig ermittelt, wie die Daten strukturiert werden könnten. Als Beispiel zum besseren Verständnis sollen erneut die Bilder von Menschen, Tieren und Häusern dienen. Da beim Unsupervised Learning eine Gruppierung der Daten nicht initial vorhanden ist und die Daten somit nicht klassifiziert sind, könnte ein mögliches Ergebnis die Unterteilung der Bilder nach der am häufigsten aufgetretenen Farbe sein.

Das Reinforcement Learning - oder auch verstärkendes Lernen - ist eine besondere Form des maschinelles Lernens. Dabei versucht der Algorithmus ein Problem selbstständig zu lösen und wird bei einer richtigen Ausführung belohnt, um selbstständig eine Strategie zur Lösung des Problems zu erlernen und die Belohnung zu maximieren. Als Beispiel soll ein Hund betrachtet werden, der das Kommando 'Sitz' beim Heben der Hand ausführen soll. Anfangs wird es einige Zeit dauern, bis der Hund überhaupt versteht, was von ihm verlangt wird. Bei einer richtigen Ausführung des Kommandos belohnt man ihn mit einem positiven Impuls, wie zum Beispiel einem Leckerchen. Mit der Zeit wird der Hund selbstständig lernen, dass er sich beim Heben der Hand hinsetzen soll, um belohnt zu werden. Da im Rahmen dieser Arbeit jedoch keine künstliche Intelligenz entwickelt werden soll, wird nicht weiter auf das Reinforcement Learning eingegangen.

Um zu bestimmen, welche Algorithmen im Rahmen dieser Arbeit eingesetzt werden sollen, wurden zunächst drei Fragen formuliert, auf die man hoffte, eine Antwort zu finden:

1. Besteht ein Zusammenhang zwischen der Körpertemperatur und der Sauerstoffsättigung im Blut?
2. Kann mithilfe der Werte von Körpertemperatur und Sauerstoffsättigung im Blut das Geschlecht vorhergesagt werden?
3. Welcher Zusammenhang kann aus Körpertemperatur und Sauerstoffsättigung im Blut ermittelt werden, wenn man keine konkrete Fragestellung formuliert?

Um die erste Fragestellung zu untersuchen, wurden mithilfe des Supervised Learning Algorithmus Lineare Regression zwei Zielvariablen, in diesem Fall Körpertemperatur und Sauerstoffsättigung im Blut, auf deren Zusammenhang untersucht.

Für die zweite Fragestellung eignete sich die Untersuchung mithilfe der Supervised Learning Algorithmen k-Nearest Neighbor und Support Vector Machine, da beide Algorithmen zur Klassifizierung von Daten mit einer bekannten Zielvariable angewendet werden können. Durch die Anwendung zwei unterschiedlicher Algorithmen lässt sich der Algorithmus mit der höheren Genauigkeit bestimmen.

Für die dritte Fragestellung wurde der Unsupervised Learning Algorithmus k-Means ausgewählt. Es wurde versucht, anhand einer durch den Algorithmus selbstständigen Gruppierung der Daten einen Zusammenhang zu erkennen.

Im Folgenden sollen die oben genannten Algorithmen des maschinellen Lernens vorgestellt werden. Bei den Beispielen wird zum Verständnis eine Darstellung in einem 2D-Koordinatensystem verwendet. Das soll aber nicht bedeuten, dass die echten Daten ausschließlich in zwei Dimensionen vorliegen können. Um die Funktionsweise der Algorithmen anschaulicher zu gestalten, zeigt Abbildung 7 die Darstellung von Beispieldaten in einem Koordinatensystem.

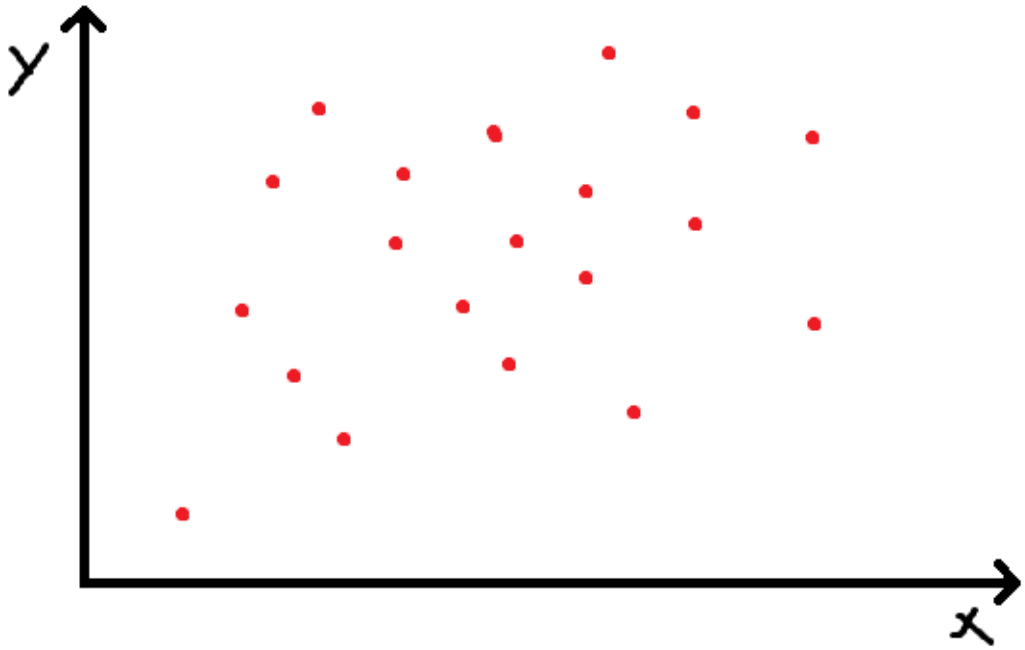


Abbildung 7: Darstellung von Beispieldaten in einem Koordinatensystem

2.3.1 Lineare Regression

Bei der Linearen Regression handelt es sich um einen Supervised Learning Algorithmus, dessen Ziel es ist, bei einer Menge von Datenpunkten eine Gerade zu ermitteln, welche die Datenpunkte möglichst gut annähert. Hierfür wird die Summe der quadrierten Abweichungen der tatsächlichen und vorhergesagten y-Werte der Datenpunkte minimiert. Eine Methode, dies mit maschinellem Lernen umzusetzen, ist es, zunächst mit einer zufälligen Parametrisierung der Geraden zu beginnen und diese iterativ zu verbessern. Aus Basis der gefundenen Parameter für die Regressionsgerade können dann Vorhersagen für die Zielvariable gettroffen werden.[8] Abbildung 8 zeigt ein Beispiel für eine mögliche Anwendung der Linearen Regression.

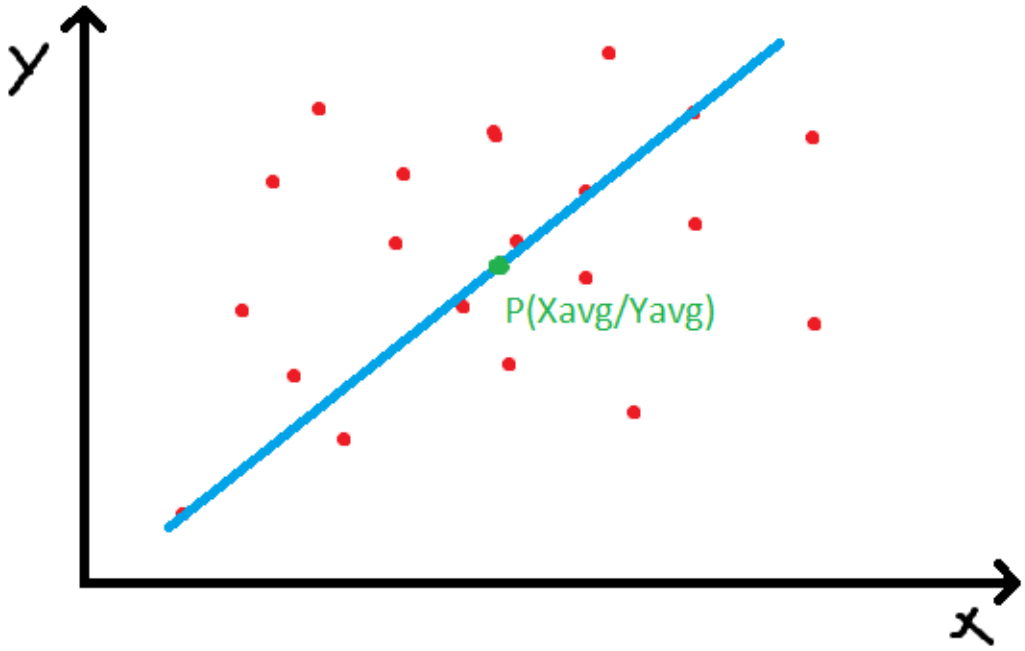


Abbildung 8: Beispiel Lineare Regression

2.3.2 k-Nearest Neighbor

Beim k-Nearest Neighbor handelt es sich um einen Supervised Learning Algorithmus, dessen Ziel es ist, einen neuen Datenpunkt einer bereits vorhandenen Klasse an Datenpunkten zuzuordnen.[8] In Abbildung 9 werden die Datenpunkte in unterschiedlichen Farben dargestellt und jede Farbe entspricht einer Klasse.

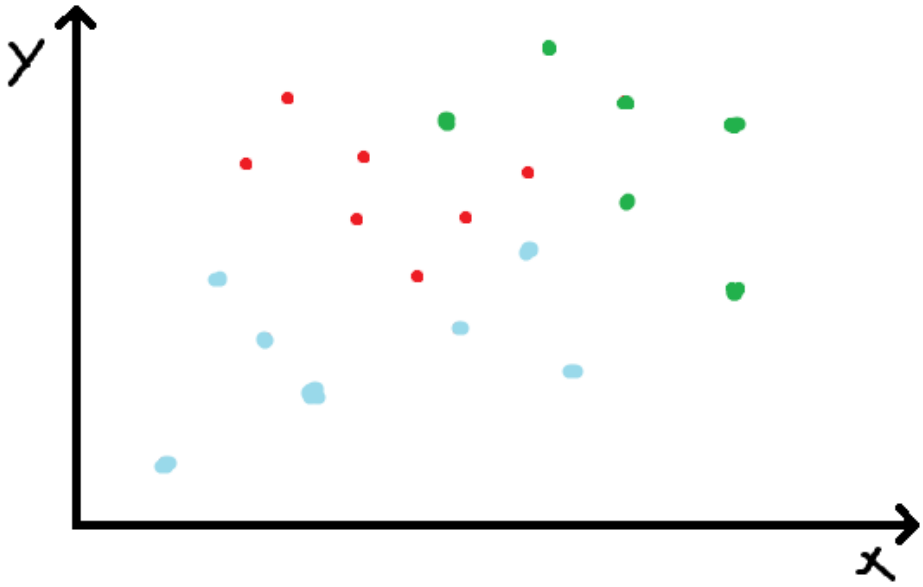


Abbildung 9: Klassifizierte Datenpunkte als Grundlage für k-Nearest-Neighbor

Nun soll ein neuer Datenpunkt eingefügt werden, der automatisiert durch den Algorithmus einer bereits vorhandenen Klasse zugeordnet wird. Dabei wird noch die Variable k festgelegt, wobei k der Anzahl der am nächsten gelegenen Datenpunkte entspricht, die für die Klassifizierung untersucht werden sollen. Für das Beispiel in Abbildung 10 wurde k = 5 gesetzt und der rosa markierte Punkt entspricht dem Punkt, der klassifiziert werden soll. Von den fünf am nächsten gelegenen Datenpunkten entsprechen vier der Klasse rot, während einer der Klasse grün entspricht. Da bei k = 5 die meisten Nachbarn der Klasse rot entsprechen, wird der neue Datenpunkt als rot klassifiziert. Wenn die am häufigsten vorkommende Klasse nicht eindeutig bestimmt werden kann (zum Beispiel zwei mal rot, zwei mal blau und ein mal grün), wird die Klasse des Datenpunktes entweder zufällig bestimmt oder, wenn möglich, bekommt der Datenpunkt beide Klassen.

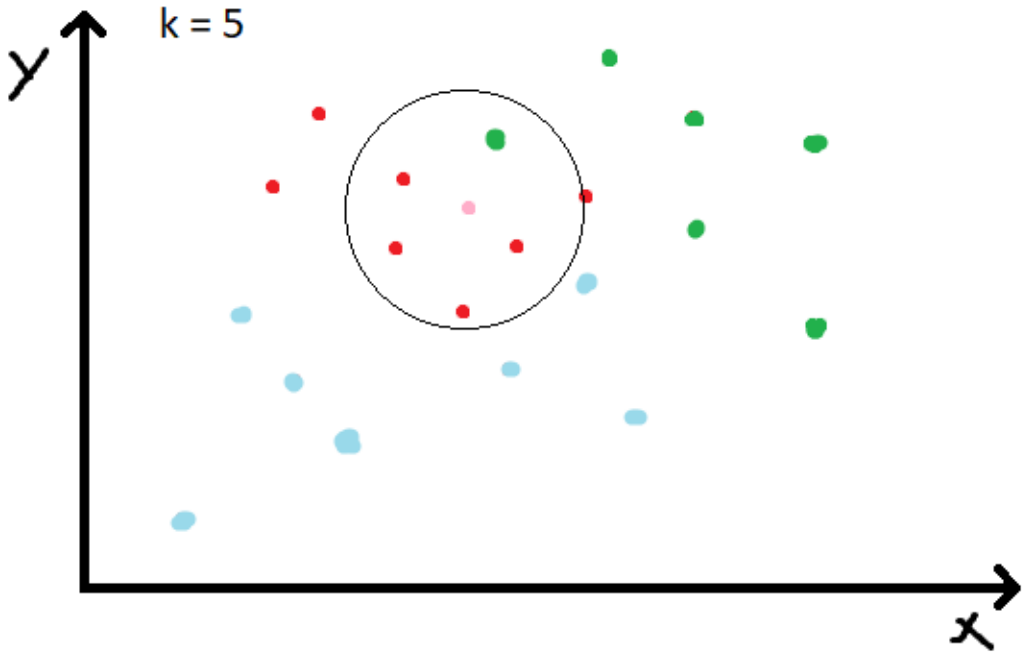


Abbildung 10: Beispiel für k-Nearest-Neighbor

2.3.3 SVM (Support Vector Machine)

Bei einer SVM handelt es sich um einen Supervised Learning Algorithmus, dessen Ziel es ist, mittels einer Trennlinie (sogenannter Hyper-Plane) Daten in bestehende Klassen zu klassifizieren.[8] Als Ausgangspunkt dient dazu eine Menge an Datenpunkten, die wie in Abbildung 11 beispielsweise in rot und grün klassifiziert wurden. Zwischen den beiden Klassen soll nun eine Hyper-Plane existieren, die künftige Daten möglichst präzise in die beiden Klassen einteilen soll.

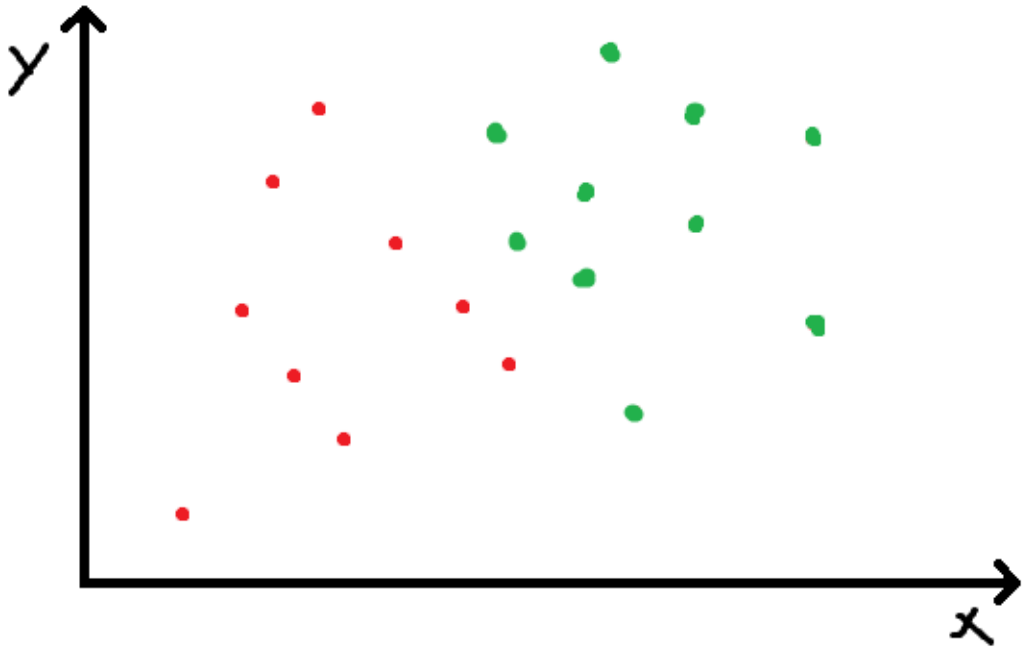


Abbildung 11: Klassifizierte Datenpunkte als Grundlage für SVM

Dabei stößt man – wie in Abbildung 12 zu sehen – auf ein Problem, denn es können unterschiedliche Hyper-Planes existieren, womit eine Klassifizierung der als blau dargestellten Datenpunkte sich als schwer gestaltet. Um eine sicherere Möglichkeit zur Klassifizierung einzelner Datenpunkte zu ermöglichen, wird daher der maximale Abstand zwischen den Datenpunkten beider Klassen als zusätzliche Maßnahme festgelegt. Dieser Abstand wird auch als Margin bezeichnet.

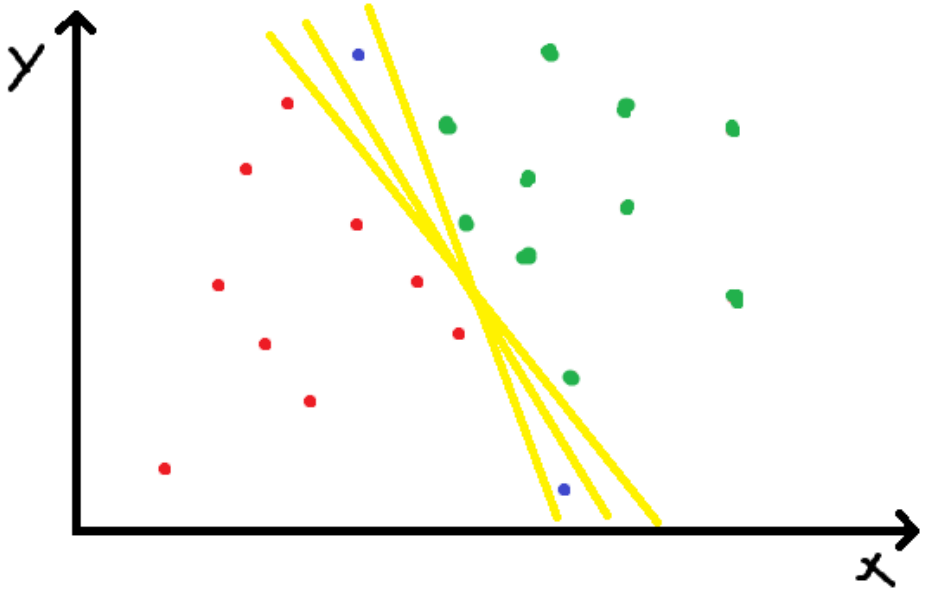


Abbildung 12: Möglichkeiten für Hyper-Planes

In Abbildung 13 soll ein Beispiel aufgeführt werden, bei dem der Abstand zwischen den Datenpunkten maximiert wurde, um somit eine optimale Hyper-Plane zu ermitteln. An dieser Stelle sei gesagt, dass dies nur zum Verständnis dienen soll und bei der erstellten Abbildung möglicherweise nicht die optimale Hyper-Plane gewählt wurde. Zukünftige Datenpunkte, die rechts von der Hyper-Plane liegen, werden demnach als grün klassifiziert, die Datenpunkte links hingegen als rot. Für die blauen Datenpunkte in der Abbildung würde das bedeuten, dass der obere blaue Punkt als grün und der untere Datenpunkt als rot klassifiziert wird.

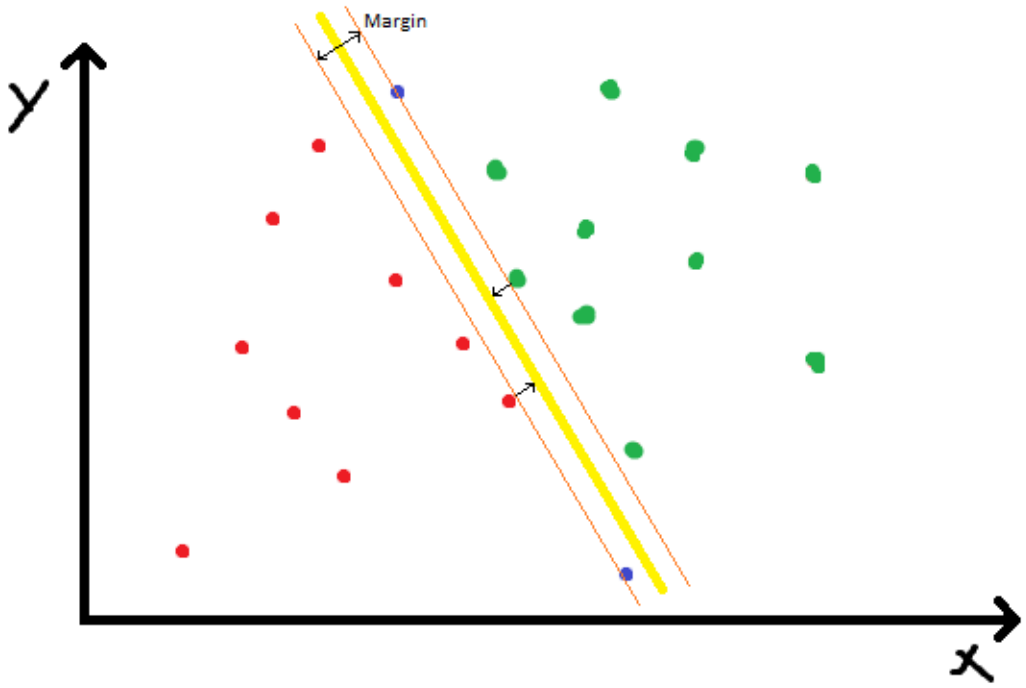


Abbildung 13: Hyper-Plane mit maximaler Margin

Mit einer SVM lassen sich auch 2-dimensionale Daten klassifizieren, deren Struktur wie in Abbildung 14 nicht linear ist. Solche Daten können mithilfe eines sogenannten Kernels klassifiziert werden, indem die Datenpunkte in eine höhere Dimension gebracht werden. Demnach wird als Kernel eine Funktion bezeichnet, die einen 2D-Datenpunkt in eine höhere Dimension, in dem Fall in einen 3D-Datenpunkt, bringt. Ein Beispiel für eine Kernelfunktion könnte bei einem Punkt P(X,Y) sein: $(X^2)+(Y^2)=Z$ für $P_2(X,Y,Z)$. Eine Hyper-Plane im 3-dimensionalen Raum kann man sich statt einer Trennlinie als eine Trennwand vorstellen.

Im Optimalfall lassen sich somit n-dimensionale Datenpunkte wie in Abbildung 15 zu sehen in einem (n+1)-dimensionalen Raum mit einer Hyper-Plane klassifizieren.

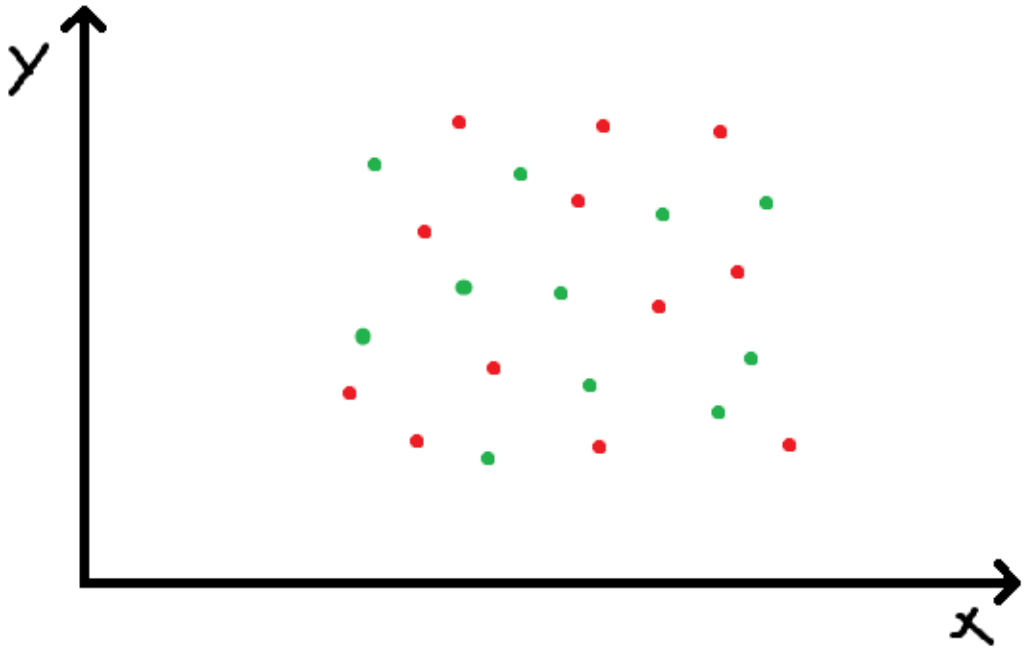


Abbildung 14: Willkürliche Datenpunkte in einem 2D-Koordinatensystem

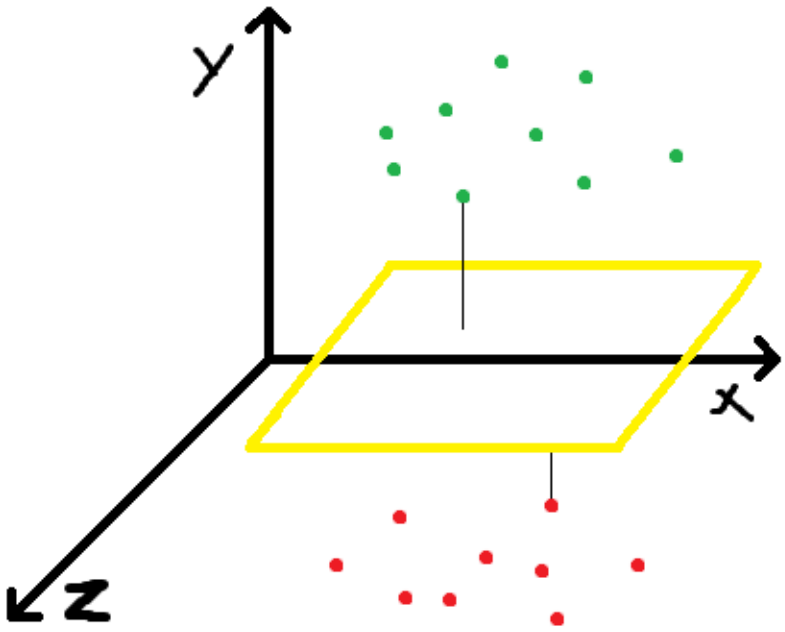


Abbildung 15: Klassifizierung in einem 3-dimensionalen Raum

2.3.4 k-Means

Bei k-Means handelt es sich um einen Unsupervised Learning Algorithmus, dessen Ziel es ist, Daten ohne bereits vorhandener Gruppierung eigenständig zu klassifizieren und so möglicherweise Zusammenhänge oder Muster in Daten zu erkennen, die sonst nicht

erkennbar wären.[9] Dazu wird eine Anzahl k an sogenannten Centroids festgelegt, also der Anzahl der Gruppen, in die die Datenpunkte unterteilt werden sollen. Die Centroids werden vorerst willkürlich platziert. Abbildung 16 zeigt eine Visualisierung von einer Datenmenge mit k = 3 Centroids, die als unterschiedlich farbige Kreise mit dem Buchstaben C darin dargestellt wurden.

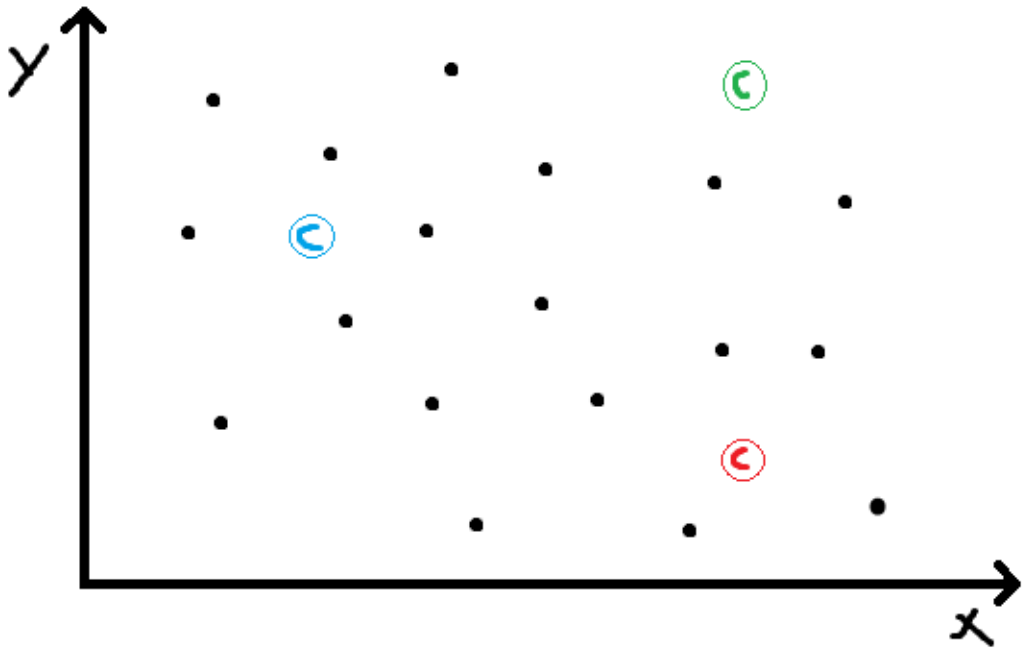


Abbildung 16: Beispiel für Platzierung der Centroids in einem 2D-Koordinatensystem

Nun werden die Datenpunkte dem Centroid zugeordnet, die ihnen am nächsten ist, und die Centroids werden so verschoben, dass sie sich in der Mitte der zugehörigen Datenpunkte befinden. Dies wird in Abbildung 17 illustriert.

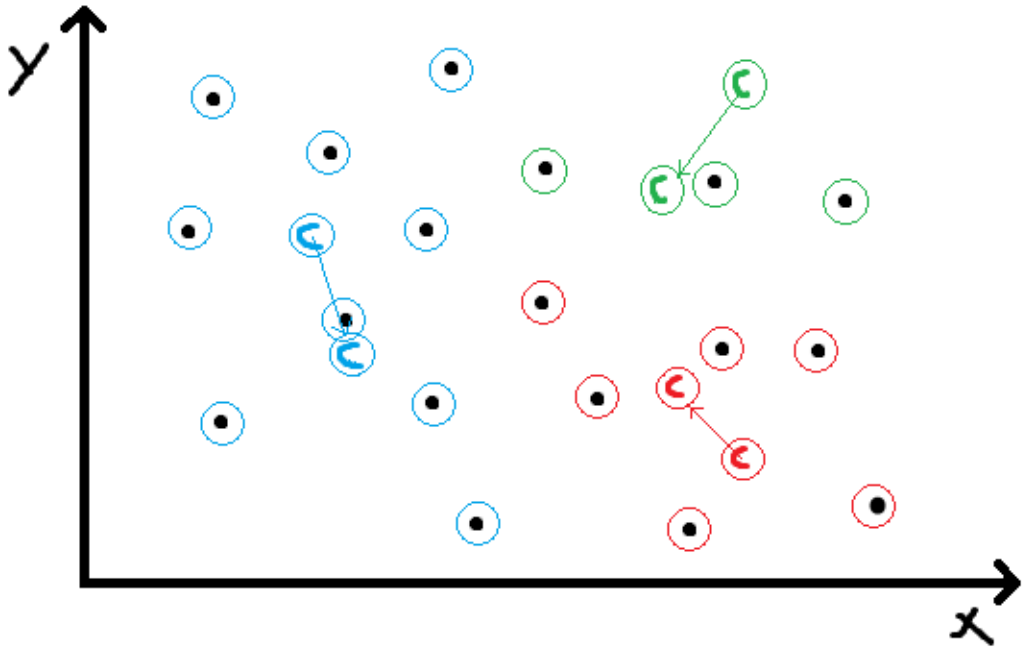


Abbildung 17: Zuordnung der Datenpunkte an einen Centroid sowie Verschiebung der Centroids

Als nächstes findet erneut eine Zuordnung der Datenpunkte an den nächstgelegenen Centroid statt. Wie in Abbildung 18 zu sehen ist, verändern dadurch im Beispiel zwei Datenpunkte ihre Zuordnung. Dieser Prozess findet iterativ statt, bis es keine Datenpunkte mehr gibt, die nach einer Verschiebung der Centroids ihre Zuordnung ändern.

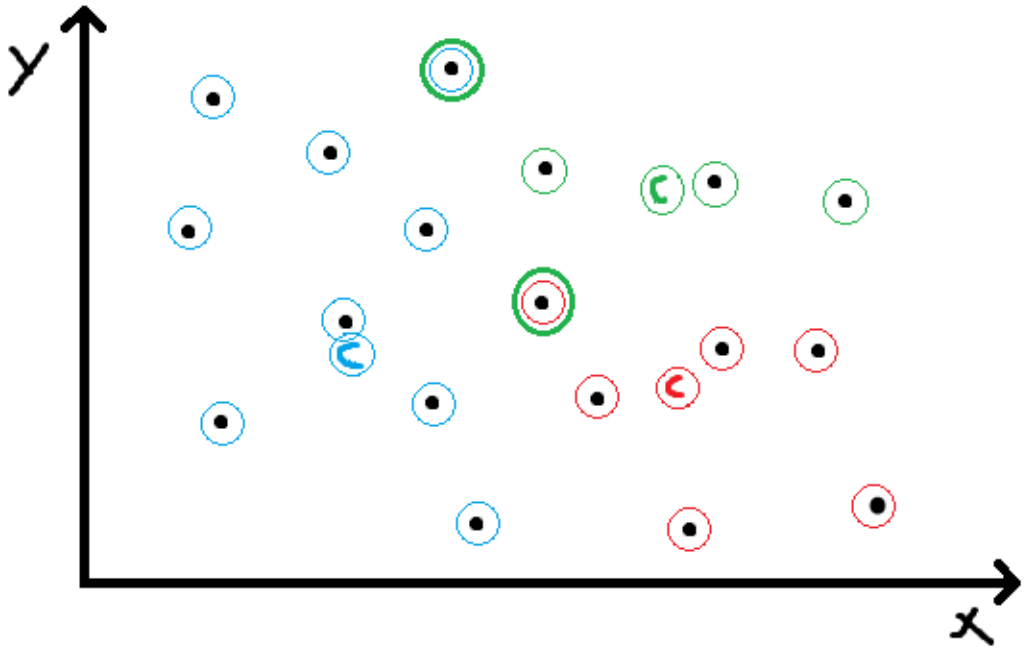


Abbildung 18: Erneute Zuordnung der Datenpunkte an den nächstgelegenen Centroid

3 Visualisierung medizinischer Sensordaten

Wie bereits im vorherigen Kapitel erläutert, agiert der node.js-Server als zentrale Komponente für Datenaufnahme sowie -wiedergabe. Mittels Python und der Bibliothek Plotly sollen nun die in der `meddata`-Tabelle der Datenbank vorhandenen Daten visuell in Graphen dargestellt werden können. Dabei wurden folgende Anforderungen identifiziert:

1. Die Komponente zur Visualisierung soll eigenständig und unabhängig vom eingesetzten Server sein.
2. Es soll für jede übergebene Spalte der Datenbank ein eigener Graph erstellt werden können.
    2.1. Es soll möglich sein, mittels einer Navigationsleiste durch die übergebenen Spalten navigieren zu können.
    2.2. Es soll möglich sein, für einzelne Nutzer eine Visualisierung erstellen zu können.
    2.3. Es soll möglich sein, alle Nutzer in einer gemeinsamen Visualisierung darstellen zu können.
3. Beim Bewegen des Mauszeigers über die einzelnen Wertepunkte des Graphen sollen alle Daten des zugehörigen Datensatzes aufgelistet werden. Dieses Vorgehen bezeichnet man auch als Hovern.

3.1 Zur Visualisierung

Die erste ermittelte Anforderung war es, die Visualisierung als eine eigenständige und unabhängige Komponente umzusetzen. Daraus ergeben sich die Vorteile, dass die Visualisierungskomponente zum einen nach wie vor eigenständig funktioniert, selbst wenn beispielsweise der Server ausfallen sollte, und zum anderen, dass die Visualisierungskomponente unabhängig vom Zweck eingesetzt werden kann, solange die Daten einer bestimmten Struktur entsprechen.

Die Visualisierung wurde mithilfe der Programmiersprache Python und der Bibliothek Plotly umgesetzt, da Plotly zum einen mit dem Umfang der Bibliothek alle nötigen Werkzeuge bietet, um die identifizierten Anforderungen erfüllen zu können, und man zum anderen mit über 50 Millionen Benutzern weltweit[10] bei Fragen auf eine große Community zurückgreifen kann. Mit der Implementierung eines separaten Python-Skripts zur Visualisierung übergebener Daten wird Anforderung 1 bereits erfüllt.

3.2 Struktur der Daten

Wie bereits erwähnt müssen Daten einer bestimmten Struktur entsprechen, damit sie dargestellt werden können. Auf diese Anforderungen soll im Folgenden kurz eingegangen werden:

1. Das Objekt mit den Daten muss in einem für das Python-Skript lesbarem Format vorliegen, was in dieser Arbeit einem JSON-String entspricht.
2. Alle auf der y-Achse darzustellenden Daten müssen von einem numerischen Datentyp sein.
3. Die Daten müssen ein Attribut vom Typ `datetime` beinhalten, um chronologisch auf der x-Achse dargestellt werden zu können.
4. Wenn Daten von mehreren Nutzern dargestellt werden sollen, müssen sie ebenfalls ein Attribut mit einem Nutzernamen beinhalten, um Daten mehrerer Nutzer in einem Graphen nicht zu vermischen

Damit ergeben sich die Parameter, die beim Aufruf des Python-Skripts übergeben werden müssen.

```
1 spawn('python', ['plotUser.py', JSON.stringify(rows), '
    aufnahmeDatum']);
```

Listing 1: Auszug aus `server.js`: Beispiel für einen Aufruf des Python-Skripts mithilfe der node.js Bibliothek spawn

In Listing 1 wird das Python-Skript `plotUser.py` mithilfe der spawn Bibliothek von node.js aufgerufen und bekommt als Parameter das Objekt mit den gewünschten zu visualisierenden Daten als Variable `rows`. Außerdem muss zusätzlich das Datumsfeld angegeben werden. In diesem Fall handelt es sich um das aufnahmeDatum. `JSON.stringify(rows)` konvertiert dabei das von der Datenbank erhaltene Javascript-Objekt `rows` in einen für das Python-Script lesbaren JSON-String. In Listing 2 wird zusätzlich der Spaltenname des Benutzernamens übergeben, um für Anforderung 2.3 Datensätze einzelnen Benutzern zuordnen zu können, um diese korrekt in einem Graphen darzustellen.

```
1 spawn('python', ['plotUser.py', JSON.stringify(rows), '
    aufnahmeDatum', 'userName']);
```

Listing 2: Auszug aus server.js: Beispiel für einen Aufruf des Python-Skripts mit zusätzlichem Parameter userName

3.3 Implementierung der Visualisierungskomponente

Damit das Python Skript überhaupt ausgeführt werden kann, muss neben Python, das standardmäßig mit dem package installer pip ausgestattet ist, die plotly Bibliothek installiert werden. Dies kann beispielsweise mit dem bereits genannten pip package installer in der Kommandozeile mit dem Befehl `pip install plotly==4.10.0` geschehen, wobei 4.10.0 der zum Zeitpunkt der Erstellung dieser Arbeit neuesten Version von plotly entspricht. Die aktuellste Version der Bibliothek kann auf der offiziellen Seite[11] ermittelt werden. Bevor konkret auf die Implementierung eingegangen wird, sollen die benötigten Bibliotheken anhand Listing 3 kurz erläutert werden.

```
#!/usr/bin/python3
import sys
import json
import numbers

import plotly.graph_objects as go
from datetime import datetime, timedelta

#sys.argv[1] = data (all database data)
#sys.argv[2] = xaxis (name of datetime column in database)
#sys.argv[3] = userName (name of userName column in database)
```

Listing 3: Auszug aus plotly.py: Benötigte Bibliotheken für plotUser.py sowie Bedeutung der zu übergebenden Parameter

Der Import des `sys`-Moduls ermöglicht es, auf die mit dem Aufruf des Skripts übergebenen Parameter zuzugreifen. `sys.argv[1]` entspricht dabei den übergebenen Daten als JSON-String, `sys.argv[2]` dem Namen der Datumsspalte in der Datenbank und `sys.argv[3]` dem Namen der Spalte des Nutzernamens der Datenbank. Um den JSON-String interpretieren und später konvertieren zu können, wird das Modul `json` benötigt. Mit dem Import von `plotly.graph_objects` wird die Visualisierung mithilfe der Plotly-Bibliothek ermöglicht. Die Klasse `datetime` wird zur Konvertierung eines Strings in ein `datetime`-Objekt benötigt und `timedelta` ermöglicht die Addition und Subtraktion von Zeitintervallen zu einem `datetime`-Objekt. Das Modul `numbers` ermöglicht später die Abfrage, ob es sich bei den Spaltenwerten um numerische Werte handelt.

Um bei fehlerhafter Datenübergabe an die Visualisierungskomponente auf das System zugeschnittene Fehlermeldungen anzuzeigen, mussten zuerst die möglichen Fehlerquellen identifiziert werden. Dabei wurden folgende mögliche Fehlerquellen betrachtet:

1. Datenstring wird in einem falschen Format übergeben.
2. Übergabe einer ungültigen Anzahl an Argumenten.
3. Übergabe eines Spaltennamens, der nicht dem Namen der beispielsweise zugehörigen Spalte in der Datenbank entspricht.
4. Das Datum wird nicht im gewünschten Format übergeben.
5. Die auf der y-Achse darzustellenden Spalten bestehen nicht aus numerischen Werten.

Nachdem mögliche Fehlerquellen identifiziert wurden, werden die Standardfehlermeldungen der zugehörigen Fehler abgefangen, um diese zu modifizieren und dem Benutzer auf die Visualisierungskomponente zugeschnittene Fehlermeldungen ausgeben zu können.

```
try:
  userData = json.loads(sys.argv[1])
  dataRowNames = userData[0].keys()
  dateRowName = sys.argv[2]
except IndexError:
  sys.stderr.write("Wrong amount of arguments in sys.argv")
  exit(1)
except json.JSONDecodeError:
  sys.stderr.write("Data is not a JSON-String")
  exit(2)
```

Listing 4: Auszug aus `plotly.py`: Beispiele für Abfangen der Fehlerquellen

Dabei wurde – wie in Listing 4 zu sehen – die Fehlernachricht in `stderr` geschrieben. Dies ist wichtig, damit nicht nur die Konsole beim Starten des `plotUser.py`-Skripts, sondern auch der externe Server die Fehlernachricht ausgeben kann. Beim zugehörigen node.js-Server kann wie in Listing 5 beispielsweise über die Funktion `stderr.on` der spawn-Bibliothek auf die Fehlernachricht zugegriffen werden. Mittels `json.loads` werden die als JSON-String übergebenen Daten in ein Python Dictionary konvertiert, um zum einen die Weiterverarbeitung der Daten möglich zu machen und zum anderen die Daten wieder in ein objektorientiertes Format zu bringen. Damit lässt sich mittels Keys und Values auf die einzelnen Spalten sowie deren zugehörigen Werte zugreifen.

```
python.stderr.on('data', function(err) {
  console.log(err.toString());
});
```

Listing 5: Auszug aus `server.js`: Zugriff und Ausgabe der Fehlernachricht des Python-Skripts durch den Server

Um über die Keys des Python Dictionary navigieren zu können, wird bei jedem Aufruf des Python-Skripts dynamisch der HTML-Code für eine Navigationsleiste mit veränderbaren Parametern erzeugt. Dabei werden für die Navigation nur die Spalten berücksichtigt, die nicht dem Spaltennamen des Datums oder des Nutzernamens entsprechen. Da an dieser Stelle nicht bekannt ist, ob eine Spalte für den Nutzernamen existiert, da diese nur beim Plotten mehrerer Nutzer gebraucht wird, wird `dataRowName != sys.argv[3]` – wie in Listing 6 zu sehen – geprüft. Wenn der Benutzername nicht existiert, ist `len(sys.argv) < 4` immer `True` und verhindert somit die Abfrage, ob `sys.argv[3]` überhaupt existiert, was sonst eine Fehlermeldung zur Folge hätte. Die `showPlot`-Funktion regelt dabei beim Wechseln der einzelnen Tabs auf der Navigationsleiste, ob und welcher Graph gerade angezeigt wird. Wichtig sei dabei zu erwähnen, dass es sich bei `showPlot` um eine Javascript-Funktion handelt, die im Browser ausgeführt wird. Insgesamt wurde damit Anforderung 2.1 erfüllt.

```
dynamicNav = ''
for dataRowName in dataRowNames:
  if dataRowName != dateRowName and (len(sys.argv) < 4 or
      dataRowName!= sys.argv[3]):
    dynamicNav += """<li onclick="showPlot('"""+dataRowName+"""')
       ;">"""+dataRowName+"""</li>"""
```

Listing 6: Auszug aus plotly.py: Erstellung der Navigationsleiste

Da es sich bei dem Datum in `userData` (dem Python Dictionary) zum derzeitigen Zeitpunkt um einen String handelt, muss dieser in ein `datetime`-Objekt konvertiert werden. Um den Programmcode übersichtlicher zu halten, wird das gewünschte Format in der Variable `dateFormat` festgehalten. Zudem werden in der Liste `dateList` die einzelnen Datumswerte gespeichert, um später das minimale und maximale Datum des im Graphen abzubildenden Wertebereiches zu ermitteln.

Die Visualisierung selbst erfolgt jedoch erst in den jeweiligen Funktionen `plotOneUser` und `plotMultipleUser`, wobei der wesentliche Unterschied darin besteht, dass in `plotOneUser` direkt ein Graph mit der zugehörigen Linie gezeichnet wird, bei `plotMultipleUser` jedoch zuerst ein leerer Graph erzeugt wird und erst danach dieser mit Linien, die den jeweiligen Nutzernamen und deren zugehörigen Werte repräsentieren, befüllt wird. Im Folgenden soll auf die `plotOneUser`-Funktion, wie in Listing 7 dargestellt, eingegangen und bei Unterschieden zur `plotMultipleUser`-Funktion diese erläutert werden.

```
def plotOneUser(userData, dataRowNames, dateRowName, dateFormat,
    dateList):

for dataRowName in dataRowNames:
  if dataRowName != dateRowName:
    x_axisData = []
    y_axisData = []
    hoverData = []

    for data in userData:
      date = data[dateRowName]

      try:
        x_axisData.append(datetime.strptime(date, dateFormat))
      except ValueError:
        sys.stderr.write("Given dates from date row doesnt match
            the format YYYY-MM-DD HH:MM:SS")
        exit(3)

      if not isinstance(data[dataRowName], numbers.Number) and
          data[dataRowName] is not None:
        sys.stderr.write("Cannot plot non-numerical data.")
        exit(6)
      y_axisData.append(data[dataRowName])

      hoverString = ""

      for key, value in data.items():
        hoverString += str(key) + ": " + str(value) + "<br>"

      hoverData.append(hoverString + "<extra></extra>")

    layout = createLayout(dateRowName, dataRowName)

    fig = go.Figure(data=[go.Scatter(x=x_axisData, y=y_axisData,
        mode="markers", hovertemplate=hoverData)], layout=layout)

    fig.update_layout(xaxis_range=[datetime.strptime(min(dateList
        ), dateFormat) - timedelta(days=1), datetime.strptime(max(
        dateList), dateFormat) + timedelta(days=1)])

    print('<div id="'+dataRowName+'" class="visibleElements"
        style="display: none;">')
    print(fig.to_html())
    print("</div>")
```

Listing 7: Auszug aus `plotly.py`: plotOneUser-Funktion

Zuerst wird über `dataRowNames` iteriert, um für die Erstellung des Graphen die jeweils zu den Spaltennamen zugehörigen Werte darzustellen. Mit der if-Abfrage `if dataRowName != dateRowName` wird sichergestellt, dass kein Graph für die Spalte mit dem Datum erstellt wird. Bei der `plotMultipleUser`-Funktion wird zusätzlich die Spalte mit dem Benutzernamen aussortiert. Die Listen `x_axisData` und `y_axisData` werden benötigt, um diese dann mit den jeweils zu den `dataRowNames` zugehörigen Werten zu befüllen. Die Liste `hoverData` wird für Anforderung 3 benötigt, um die gewünschten Daten beim Hovern über einzelne Wertepunkte des Graphen anzuzeigen. Mit `x_axisData.append(datetime.strptime(date,`

`dateFormat))` wird der String in der Variable `date` in ein `datetime`-Objekt mit dem zuvor festgelegten `dateFormat` konvertiert und der Liste `x_axisData` hinzugefügt. Dasselbe geschieht für `y_axisData` mit der Ausnahme, dass es nicht in ein `datetime`-Objekt konvertiert werden muss, da es sich bei den Werten auf der y-Achse um numerische Werte handelt. Im `hoverString` werden die jeweils zu den Wertepunkten des Graphen zugehörigen Daten gesammelt als String hinzugefügt und mit `hoverData.append` der Liste hinzugefügt. Standardmäßig wird beim Hovern über die Linie neben den Informationen über den entsprechenden Datenpunkt der Titel der jeweiligen Linie angezeigt. Dieses Verhalten ist nicht gewünscht, da insbesondere bei `plotOneUser` die Linie ohnehin nicht benannt wird und daher einen automatisch generierten Namen erhält. Durch den leeren `<extra>`-Tag wird diese ungewünschte Zusatzinformation nicht mehr angezeigt. Mit `go.figure` wird dann der Graph erstellt und als Variable `fig` zwischengespeichert, um diese dann später als HTML-Code ausgeben zu können. Mit `update_layout` wird der Wertebereich des Graphen festgelegt, der aus dem kleinsten Wert der `dateList` und dem größten Wert der `dateList` besteht. `timedelta(days=1)` sorgt für eine optisch schönere Darstellung des Graphen, indem das Intervall des angezeigten Wertebereichs vergrößert wird. Zum Abschluss des Kapitels werden zur optischen Veranschaulichung in den Abbildungen 19 bis 21 die Elemente des in diesem Kapitel erläuterten Codes dargestellt.

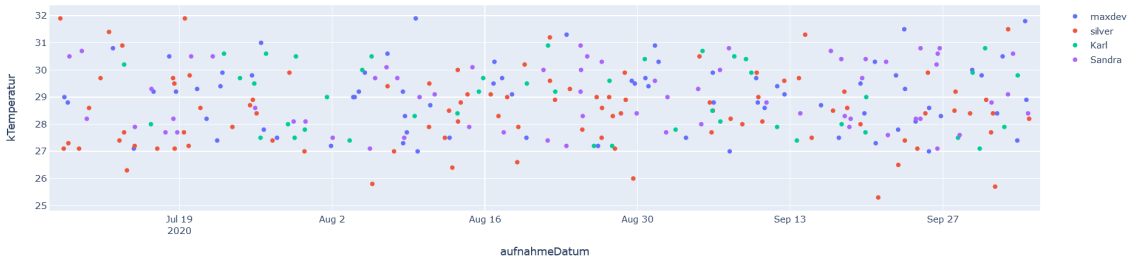


Abbildung 19: Beispiel Visualisierung von plotMultipleUser

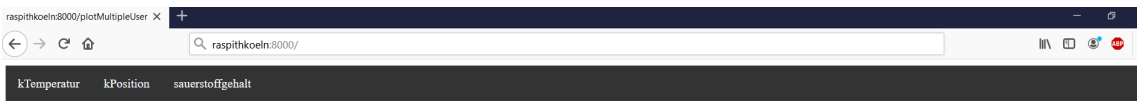


Abbildung 20: Beispiel Visualisierung der Navigationsleiste

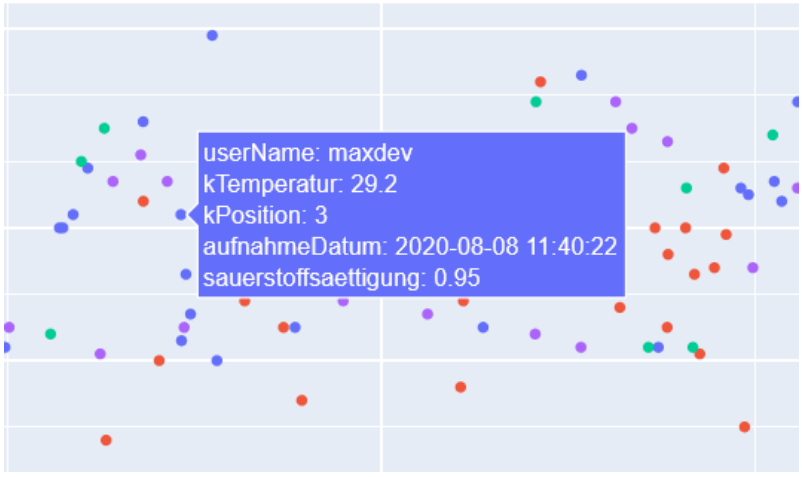


Abbildung 21: Beispiel für Anzeige der Daten beim Hovern über einen Wertepunkt des Graphen in plotMultipleUser

4 Datenanalyse mithilfe Machine Learning Algorithmen

Nachfolgend sollen die gesammelten Daten untersucht werden. Um einen besseren Überblick über die Daten zu schaffen, sollen die Daten zunächst mit den in Kapitel 2 vorgestellten visuellen Darstellungsformen näher betrachtet und schließlich mit den im selben Kapitel vorgestellten Algorithmen näher untersucht werden. Die gesammelten Daten wurden zu Untersuchungszwecken als csv-Datei exportiert. Die Programmcodes wurden in einem Jupyter-Notebook geschrieben.

4.1 Einblick in die Daten

Zunächst soll ein Überblick über die Daten gegeben werden. Dies geschieht mithilfe der Bibliothek Pandas. NumPy wird in dieser Arbeit für die Manipulation der Daten genutzt, während Pandas dem Zweck der Datenfilterung dient[12].

```
import pandas as pd
data = pd.read_csv('data.csv')
data.head()
```

	userID	userName	geschlecht	gebDatum	aktKoerperform	aufnahmeDatum	koerperform	kTemperatur	kPosition	sauerstoffsaettigung
0	8001	silver	m	1993-12-09	Uebergewicht	2020-07-08 01:40:14	Uebergewicht	31.9	4	0.92
1	8001	silver	m	1993-12-09	Uebergewicht	2020-07-08 09:04:25	Uebergewicht	27.1	5	0.93
2	8002	maxdev	m	1996-07-01	Normalgewicht	2020-07-08 10:38:24	Normalgewicht	29.0	3	0.97
3	8002	maxdev	m	1996-07-01	Normalgewicht	2020-07-08 18:04:53	Normalgewicht	28.8	2	0.95
4	8001	silver	m	1993-12-09	Uebergewicht	2020-07-08 19:21:04	Uebergewicht	27.3	1	0.96

Abbildung 22: Import der csv-Datei

Abbildung 22 zeigt den Import der csv-Datei als Variable `data`. Mithilfe von `data.head()` lassen sich die ersten fünf Datensätze anzeigen, wobei die Anzahl der Datensätze auch einstellbar ist. Der Default Wert ist fünf. Da sich die Körperformen der Probanden während des Aufnahmezeitraums nicht verändert haben, sind die Werte der Attribute `aktKoerperform` und `koeperform` für jeden Datensatz identisch. Daher wird das Attribut `aktKoeperform` gelöscht. Da `userID` nur für die interne Speicherung genutzt wird und jedem `userName` eindeutig zugeordnet wird (und umgekehrt), wird das Attribut `userID` ebenfalls nicht für die Datananalyse benötigt und entfernt. Wie in Abbildung 23 zu sehen, sind die Attribute `gebDatum` und `aufnahmeDatum` vom Typ Object, was in Pandas dem Datentyp einer Zeichenkette gleichzusetzen ist. Diese sollen zu einem Datum konvertiert werden. Das erste `print(dtypes)` zeigt dabei die Datentypen der Ursprungsdaten, während das zweite `print(dtypes)` die Datentypen nach der Modifizierung anzeigt. Dabei wird sichtbar, dass sich die Attribute `gebDatum` und `aufnahmeDatum` jetzt im gewünschten Format befinden.

```
print(data.dtypes)
data = data.drop(columns=['aktKoerperform', 'userID'])
data['gebDatum'] = pd.to_datetime(data['gebDatum'], format='%Y-%m-%d')
data['aufnahmeDatum'] = pd.to_datetime(data['aufnahmeDatum'], format='%Y-%m-%d %H:%M:%S')
print(data.dtypes)
```

```
userID                  int64
userName               object
geschlecht             object
gebDatum               object
aktKoerperform         object
aufnahmeDatum          object
koerperform            object
kTemperatur           float64
kPosition               int64
sauerstoffsaettigung  float64
dtype: object
userName                      object
geschlecht                    object
gebDatum                datetime64[ns]
aufnahmeDatum           datetime64[ns]
koerperform                   object
kTemperatur                  float64
kPosition                      int64
sauerstoffsaettigung         float64
dtype: object
```

Abbildung 23: Manipulation der Daten

Für die visuelle Darstellung der Daten zu Analysezwecken soll die Bibliothek seaborn verwendet werden. Der wesentliche Unterschied zur plotly Bibliothek ist, dass der Aufwand zur Erstellung eines Diagramms geringer ausfällt, dafür jedoch mit plotly interaktive Graphen erstellt werden können. So lässt sich beispielsweise über bestimmte Wertepunkte hovern, um weitere Informationen zu diesem Datensatz zu erhalten. Bei seaborn ist eine Interaktion mit dem Graphen hingegen nicht vorgesehen.

Mithilfe eines Histogramms soll sich nun die absolute Häufigkeit der Daten verteilt auf die unterschiedlichen Körperpositionen angesehen werden, wie in Abbildung 24 zu erkennen ist. Der Befehl `%matplotlib inline` bewirkt dabei, dass die Graphen innerhalb einer Zelle des Jupyter-Notebooks dargestellt werden. Das zugehörige Histogramm wird mit dem Funktionsaufruf `sns.distplot(data["kPosition"], bins = 5, kde = False)` erstellt, wobei `bins` die Anzahl der Gruppen festlegt. `kde` steht für kernel density estimation und zeichnet in den Graphen zusätzlich eine Schätzung der Kerndichte. Dies ist jedoch nicht gewünscht und wird deshalb auf `False` gesetzt. Anhand des Histogramms kann erkannt werden, dass die Verteilung der Daten recht gleichmäßig ist. Lediglich in der Körperposition Nummer drei (auf der rechten Seite liegend) ist die Anzahl der dazugehörigen Datensätze höher.

```
import seaborn as sns
%matplotlib inline

sns.distplot(data['kPosition'], bins = 5, kde = False)
```

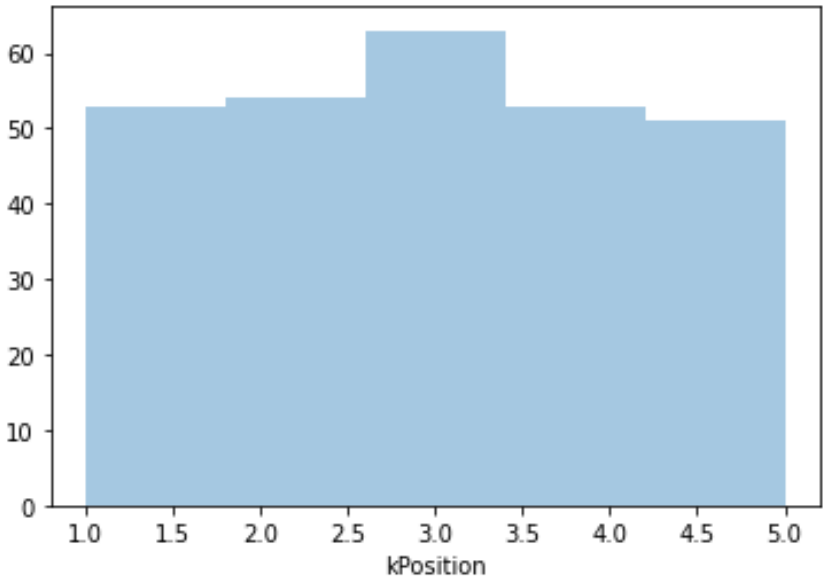


Abbildung 24: Histogramm: Anzahl der Daten gruppiert nach Körperposition

In Abbildung 25 wurde ermittelt, wie viele Daten verteilt auf das Geschlecht gesammelt wurden. Dabei kann beobachtet werden, dass die Anzahl der Datensätze männlicher Probanden wesentlich höher ausfällt als die der weiblichen Probanden.

```
sns.countplot(x='geschlecht', data=data)
```

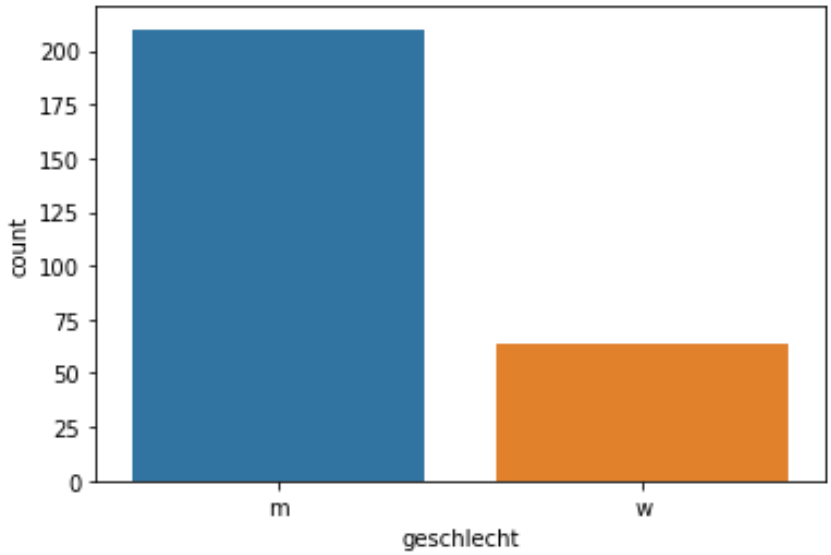
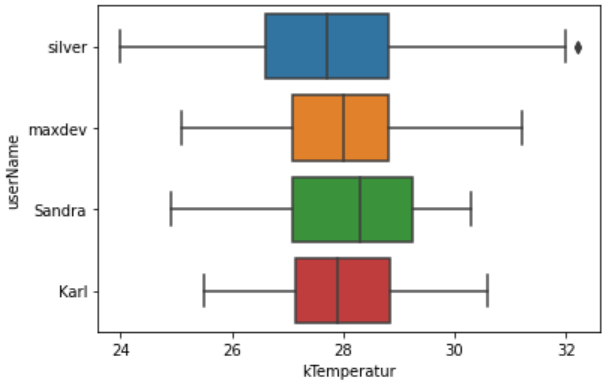


Abbildung 25: Barchart: Anzahl der Daten gruppiert nach Geschlecht

Zuletzt soll noch in Abbildung 26 in einem Boxplot beobachtet werden, wie die Verteilung der Daten hinsichtlich der Körpertemperatur und der Sauerstoffsättigung auf die einzelnen Probanden aussieht. Hierbei fällt auf, dass die meisten Messungen der Körperkontakttemperatur um 28 ° C liegen und die Sauerstoffsättigung der Probanden überwiegend zwischen 93% und 97% lag.

```
sns.boxplot(x=data['kTemperatur'], y=data['userName'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee2c706820>
```




```
sns.boxplot(x=data['sauerstoffsaettigung'], y=data['userName'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee2c9dd9a0>
```

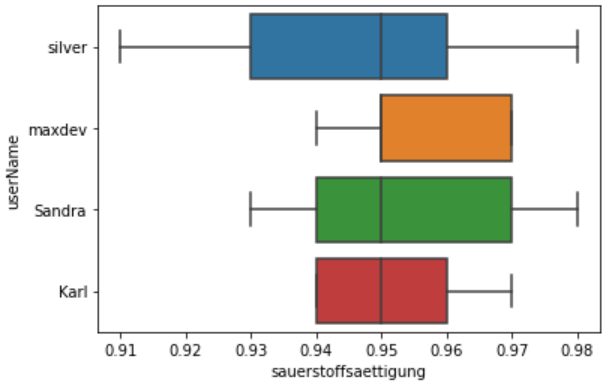


Abbildung 26: Boxplot: Vergleich der Werte einzelner Probanden anhand der Körpertemperatur und Sauerstoffsättigung

4.2 Anwendung der Machine Learning Algorithmen

Nachfolgend werden die in Kapitel 2.3 vorgestellten Algorithmen angewendet und die Resultate erläutert.

4.2.1 Lineare Regression

Abbildung 27 zeigt die Untersuchung, ob ein Zusammenhang zwischen der Körpertemperatur und der Sauerstoffsättigung im Blut besteht.

```python
import numpy as np
from sklearn.linear_model import LinearRegression

x = np.array(data['kTemperatur']).reshape((-1, 1))
y = np.array(data['sauerstoffsaettigung'])

model = LinearRegression().fit(x, y)
r_sq = model.score(x, y)

print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)

y_pred = model.predict(x)

sns.regplot(x=x, y=y)
```

```
coefficient of determination: 0.16800234676788794
intercept: 1.0702591405173778
slope: [-0.00424455]
```

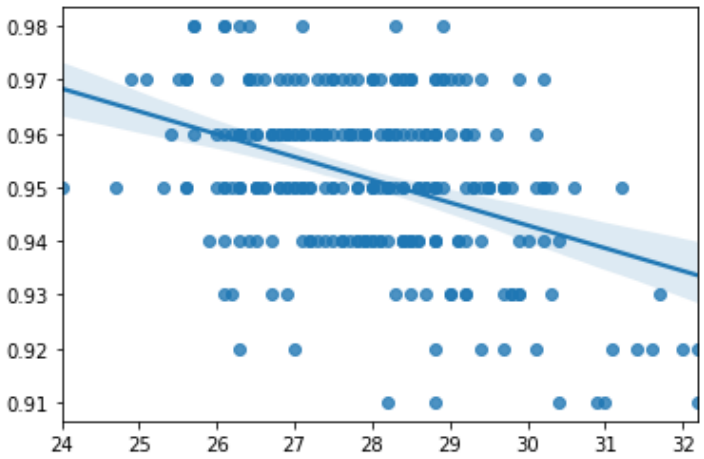


Abbildung 27: Lineare Regression: Bestimmung des Zusammenhangs zwischen Körpertemperatur und Sauerstoffsättigung im Blut

Da Python nicht über einen Array Datentyp verfügt, werden zuerst mithilfe der Python Bibliothek numpy die x- und y-Achse bestimmt. Der Befehl `reshape((-1, 1))` wird benutzt, um `x` als 2-dimensionales Array abzubilden, da ein 2-dimensionales Array erforderlich ist. Als nächstes wird ein Regressionsmodell `model` erzeugt und mit `x` und `y` befüllt. Der Determinationskoeffizient `r_sq` gibt Auskunft darüber, wie gut die Messwerte zu dem Modell passen. Dies entspricht in diesem Fall 16%. `model.intercept_` sagt aus, welchen Wert `y` im Ursprung hätte, also wenn $x = 0$. `model.coef_` bestimmt die Steigung der Best-Fit-Line. Mit einem Determinationskoeffizienten von 16% lässt sich daraus schließen, dass die Vorhersage der Sauerstoffsättigung im Blut durch Körpertemperatur nicht möglich ist. Jedoch kann ein Trend erkannt und somit eine Hypothese aufgestellt werden, dass bei erhöhter Körpertemperatur die Sauerstoffsättigung im Blut abnimmt. Dies könnte dadurch erklärt werden, dass bei körperlicher Anstrengung die Körpertemperatur zwar steigt, aber vom Körper gleichzeitig mehr Sauerstoff benötigt wird, da beispielsweise die Stoffwechselprozesse schneller ablaufen und somit die Sauerstoffsättigung im Blut abnimmt.

4.2.2 k-Nearest Neighbor und SVM

Da sich die beiden Algorithmen kaum in der Syntax unterscheiden und beide die Fragestellung behandeln, ob sich durch die Körpertemperatur und die Sauerstoffsättigung im Blut das Geschlecht bestimmen lässt, soll beides in diesem Unterkapitel behandelt werden. Abbildungen 28 (k-Nearest Neighbor) und 29 (SVM) zeigen den Aufbau sowie die Vorhersagegenauigkeit der jeweiligen Algorithmen.

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report


x_new = data.loc[:, ['kTemperatur', 'sauerstoffsaettigung']].values

y_new = data.iloc[:, 1].values


X_train, X_test, y_train, y_test = train_test_split(x_new, y_new, test_size=0.20)


scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)


classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)


print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           m       0.76      0.88      0.81        40
           w       0.44      0.27      0.33        15

    accuracy                           0.71        55
   macro avg       0.60      0.57      0.57        55
weighted avg       0.67      0.71      0.68        55
```

Abbildung 28: k-Nearest Neighbor: Versuch, das Geschlechts anhand der Körpertemperatur und der Sauerstoffsättigung zu bestimmen

In `x_new` werden die Körpertemperatur und die Sauerstoffsättigung im Blut als unabhängige Variablen extrahiert. Das gleiche geschieht mit y_new als abhängige Variable. `X_train, X_test, y_train, y_test = train_test_split(x_new, y_new, test_size=0.20)` unterteilt die Daten in Trainingsdaten und Testdaten, wobei Trainingsdaten 80% und Testdaten 20% der Daten entsprechen. `X_train = scaler.transform(X_train)` und `X_test = scaler.transform(X_test)` skaliert die Merkmale für eine schnellere Konvergenz. Durch `n_neighbors=5` bei der Erstellung des Klassifizierers wird die Anzahl der zu untersuchenden nächsten Nachbarn als fünf angelegt. Mit `y_pred = classifier.predict(X_test)` erfolgt eine Klassifizierung der Testdaten durch den Klassifizierer. Wie man dem Report

entnehmen kann, wird die richtige Klasse mit einer Wahrscheinlichkeit von 75% vorhergesagt. Wenn man sich allerdings die Anzahl der Datensätze ansieht, gibt es von männlichen Probanden über 200 Datensätze, während es von weiblichen Probanden unter 75 sind. Somit ist die grundsätzliche Wahrscheinlichkeit, dass ein Datensatz zu einer Person des männlichen Geschlechts gehört, höher. Dies fällt insbesondere bei SVM ins Gewicht, da dieser Algorithmus keine Vorhersage über Datensätze von Probanden des weiblichen Geschlechts treffen kann. Somit lässt sich schlussfolgern, dass bei den gegebenen Daten keine Vorhersage über das Geschlecht getroffen werden kann.

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report


x_new2 = data.loc[:, ['kTemperatur', 'sauerstoffsaettigung']].values

y_new2 = data['geschlecht']


X_train, X_test, y_train, y_test = train_test_split(x_new2, y_new2, test_size = 0.20)


svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)


y_pred = svclassifier.predict(X_test)


print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           m       0.76      1.00      0.87        42
           w       0.00      0.00      0.00        13

    accuracy                           0.76        55
   macro avg       0.38      0.50      0.43        55
weighted avg       0.58      0.76      0.66        55
```

Abbildung 29: SVM: Versuch, das Geschlechts anhand der Körpertemperatur und der Sauerstoffsättigung zu bestimmen

4.2.3 k-Means

Mit dem Unsupervised Learning Algorithmus soll beobachtet werden, ob sich sinnvolle Zusammenhänge erkennen lassen, wenn man an den Algorithmus keine konkrete Fragestellung formuliert. Abbildung 30 zeigt die daraus resultierenden Ergebnisse. Dabei sollen sich zwei Szenarien angesehen werden.

1. Was passiert, wenn man dem Algorithmus nur die Körpertemperatur und die Sauerstoffsättigung im Blut übergibt?

2. Was passiert, wenn zusätzlich die Körperposition mit übergeben wird?

Zunächst sollen die Parameter in `KMeans(init="random", n_clusters=3, n_init=10, max_iter=300)` erläutert werden. Mit `init="random"` werden die Centroids zufällig verteilt. `n_clusters=3` legt die Anzahl an Centroids fest. `n_init=10` legt die Anzahl der

Durchläufe fest, wie oft die Centroids anfänglich zufällig angeordnet werden sollen. In diesem Beispiel werden anfänglich zehn mal die Positionen der Centroids neu angeordnet, ehe die Anordnung mit der geringsten Fehlerquadratsumme (SSE) ausgewählt wird. `max_iter=300` legt die maximale Anzahl an Iterationen fest. Das bedeutet, es werden in diesem Fall maximal 300 mal die Centroids in die Mitte der zugehörigen Datenpunkte verschoben, wenn der Datenpunkt nach einer Verschiebung des Centroids seine Gruppe ändert. Dabei kann man erkennen, dass unter Umständen der Algorithmus vergleichsweise sehr lange brauchen könnte, um ein Ergebnis zu liefern.

```
from pandas import DataFrame
from kneed import KneeLocator
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

x_new3 = data.loc[:, ['kTemperatur', 'sauerstoffsaettigung']]
x_new4 = data.loc[:, ['kTemperatur', 'sauerstoffsaettigung', 'kPosition']]

kmeans = KMeans(init="random", n_clusters=3, n_init=10, max_iter=300).fit(x_new3)
kmeans2 = KMeans(init="random", n_clusters=4, n_init=10, max_iter=300).fit(x_new4)
centroids = kmeans.cluster_centers_
centroids2 = kmeans2.cluster_centers_

plt.scatter(x_new3['kTemperatur'], x_new3['sauerstoffsaettigung'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()

plt.scatter(x_new4['kTemperatur'], x_new4['sauerstoffsaettigung'], c= kmeans2.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroids2[:, 0], centroids2[:, 1], c='red', s=50)
plt.show()
```

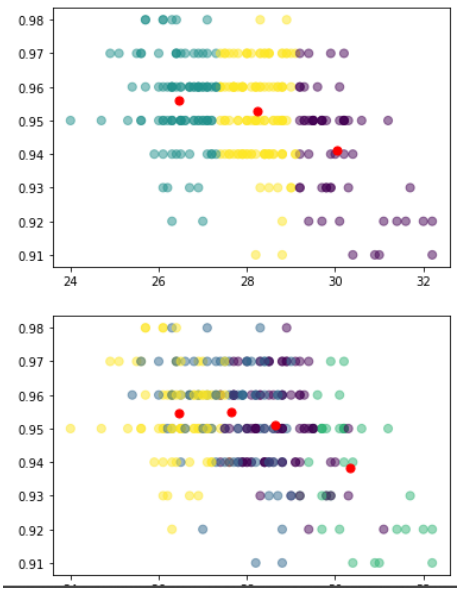


Abbildung 30: k-Means: Zusammenhänge in den Daten finden

Der untere Teil von Abbildung 30 zeigt zwei Graphen. Der obere Graph zeigt die Gruppierung der Datenpunkte mit Körpertemperatur und Sauerstoffsättigung im Blut als übergebene Parameter. Daraus lässt sich erkennen, dass der Algorithmus die Datenpunkte nach der Körpertemperatur gruppiert hat. Die Daten des unteren Graphen, dem als zusätzlicher Parameter die Körperposition übergeben wurde, lassen sich nicht interpretieren. Bei einer Darstellung in 3-dimensionalem Raum, wie in Abbildung 31 sichtbar, kann man hingegen erkennen, dass die Daten erneut nach Körpertemperatur gruppiert werden, jedoch abhängig von der Körperposition und mittig geteilt.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')


x_vals = x_new4['kTemperatur']
y_vals = x_new4['sauerstoffsaettigung']
z_vals = x_new4['kPosition']


ax.scatter(x_vals, y_vals, z_vals, c= kmeans2.labels_.astype(float), marker='o')
ax.set_xlabel('kTemperatur')
ax.set_ylabel('sauerstoffsaettigung')
ax.set_zlabel('kPosition')

plt.show()
```

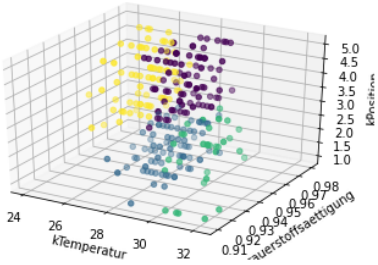


Abbildung 31: k-Means: 3D-Plot

Zur Ermittelung einer Anzahl an Centroids gibt es zwei Möglichkeiten. Die Ellenbogenmethode und den Silhouettenkoeffizienten. Für die Untersuchung von `x_new3` wurde die Ellenbogenmethode und für `x_new4` der Silhouettenkoeffizient zur Ermittlung der Anzahl an Centroids genutzt.

In Abbildung 32 sieht man die Anwendung der Ellenbogenmethode. Dabei wird die Summe der Fehlerquadrate einem Intervall der Anzahl an möglichen Centroids gegenübergestellt. Dabei entsteht ein sogenannter Ëllenbogen". Es wird versucht ein Kompromiss zwischen der Anzahl an Centroids und der Summe der Fehlerquadrate zu finden. In diesem Fall wäre es drei.

```
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300
}


sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(x_new3)
    sse.append(kmeans.inertia_)


plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()


kl = KneeLocator(
    range(1, 11), sse, curve="convex", direction="decreasing"
)
kl.elbow
```

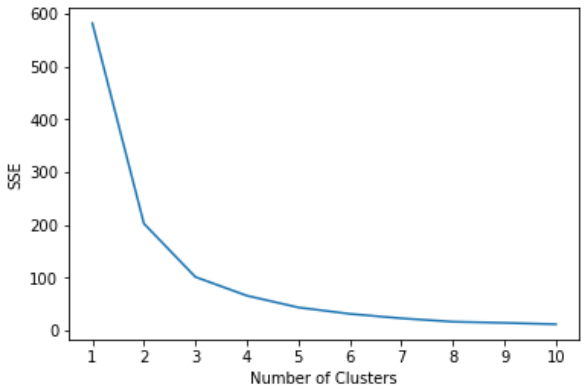


Abbildung 32: Anwendung der Ellenbogenmethode

In Abbildung 33 sieht man die Bestimmung des Silhouettenkoeffizienten. Dabei wird die Qualität des Clusterings für ein Intervall der Anzahl an möglichen Centroids als Silhouettenkoeffizient ausgegeben. Man sollte die Anzahl an Centroids mit dem größten Silhouettenkoeffizienten nehmen. In diesem Fall wäre das vier.

```
silhouette_coefficients = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(x_new4)
    score = silhouette_score(x_new4, kmeans.labels_)
    silhouette_coefficients.append(score)


plt.plot(range(2, 11), silhouette_coefficients)
plt.xticks(range(2, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.show()
```

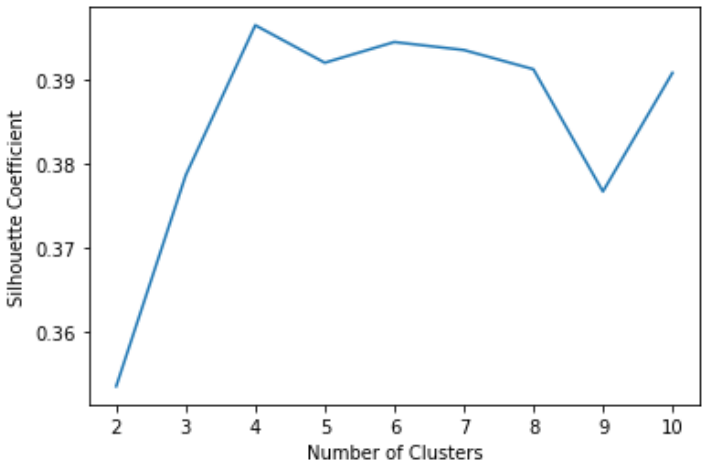


Abbildung 33: Bestimmung der Silhouettenkoeffizienten

5 Fazit

Im Wesentlichen unterteilte sich die Bachelorarbeit in drei unterschiedliche Bereiche.

1. Sammeln der Daten
2. Visuelle Darstellung der Daten
3. Auswertung der Daten

Nachfolgend soll auf die Bereiche und, falls vorhanden, über die Probleme bei der Ausführung eingegangen werden.

5.1 Sammeln der Daten

Es konnten erfolgreich die für die Arbeit benötigten Daten gesammelt werden. Allerdings fehlen in diesem Zusammenhang quantitative Daten. Zum einen war es nicht möglich, enorme Datenmengen für die Arbeit zu sammeln, was unter anderem auch dem aktuellen Pandemiestatus zuzuschreiben ist. Es konnten in näherem Umkreis nur vier Probanden gefunden werden, die mit der Sammlung medizinischer Daten einverstanden waren. Zwar gab es im weiteren Umkreis noch einige Interessenten, aber da die Messung häufig durchgeführt werden musste, um relevante Informationen zu bekommen, und das Sammeln der Daten nicht die einzige Aufgabe in dieser Arbeit war, stellte es sich aus zeitlichen Gründen als nicht möglich heraus. Durch die begrenzte Menge an Probanden konnten auch nicht besonders vielseitige Daten gesammelt werden. Sowohl die Körperform als auch das Alter konnten für Analysezwecke nicht in Betracht gezogen werden. Alle Probanden wiesen eine ähnliche Körperform auf und das Alter lag im Intervall von 20 bis 30 Jahren. Innerhalb von drei Monaten gab es auch bei niemandem eine Änderung der Körperform, was sich ebenfalls gut hätte analysieren lassen können. Anfangs wurde mit dem Gedanken gespielt, zusätzliche Eigenschaften mit in die Daten zu integrieren, wie beispielsweise, wenn sich jemand fiebrig fühlt oder an Bauch-/Kopfschmerzen leidet. Jedoch kann man bei einem Probanden eine Krankheit nicht erzwingen und selbst wenn eine Datenaufnahme bei Krankheitssymptomen möglich gewesen wäre, hätte man erneut das Problem, dass die Daten nicht quantitativ genug wären. Mit 274 Datensätzen konnte zwar immerhin im Rahmen einer 1-Personen-Arbeit eine beachtliche Anzahl an Daten gesammelt werden, aber für das maschinelle Lernen schienen es dennoch viel zu wenig gewesen zu sein, um aussagekräftige Muster und Zusammenhänge zu erkennen.

5.2 Visuelle Darstellung der Daten

Mit der implementierten Komponente lassen sich nicht nur die gesammelten Daten plotten, sondern auch generell alle Daten, die einem bestimmten Format entsprechen. Im Wesentlichen setzt dies voraus, dass die auf der y-Achse darzustellenden Daten numerische Werte beinhalten und es ein Attribut mit einem Datum gibt, um es auf der x-Achse darstellen zu können.

5.3 Auswertung der Daten

Eine Auswertung der Daten funktionierte aus den in Kapitel 5.1 dargestellten Gründen nur bedingt. Es konnte die Hypothese aufgestellt werden, dass bei einer erhöhten Körpertemperatur eine Tendenz zu einer niedrigeren Sauerstoffsättigung vorliegt.

Literatur

[1] Llc, S. I. E. (2020, Januar 30). WO2020023321 IN-GAME RESOURCE SURFACING PLATFORM. patentscope. https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2020023321, letzter Abruf am 25.10.2020

[2] I care Pflege (1. Aufl.). (2015). Thieme Georg Verlag.

[3] Guder, W. G. & Nolte, J. (2009). Das Laborbuch: für Klinik und Praxis - mit Zugang zum Elsevier-Portal (2. Aufl.). Urban & Fischer/Elsevier.

[4] World Health Organization. (o. J.). Body mass index - BMI. WHO. https://www.euro.who.int/en/health-topics/disease-prevention/nutrition/a-healthy-lifestyle/body-mass-index-bmi, letzter Abruf am 25.10.2020

[5] Grundlagen der Wahrscheinlichkeitsrechnung und Statistik. Eduard Cramer und Udo Kamps. 3. Auflage.

[6] Maschinelles Lernen - Eine Analyse zu Kompetenzen, Forschung und Anwendung. Fraunhofer-Gesellschaft. https://www.bigdata.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/Fraunhofer_Studie_ML_201809.pdf, letzter Abruf am 26.10.2020

[7] An introduction to machine learning. Pierre Lison, Language Technology Group (LTG), Department of Informatics, HiOA, October 3 2012 http://heim.ifi.uio.no/plison/pdfs/talks/machinelearning.pdf, letzter Abruf am 26.10.2020

[8] Machine Learning - A Probabilistic Perspective. Kevin P. Murphy. https://doc.lagout.org/science/Artificial%20Intelligence/Machine%20learning/Machine%20Learning_%20A%20Probabilistic%20Perspective%20%5BMurphy%202012-08-24%5D.pdf, letzter Abruf am 26.10.2020

[9] K Means. Chris Piech. Based on a handout by Andrew Ng. https://stanford.edu/ cpiech/cs221/handouts/kmeans.html, letzter Abruf am 26.10.2020

[10] Plotly: The front end for ML and data science models. (o. J.). plotly. https://plotly.com/, letzter Abruf am 25.10.2020

[11] Getting Started with Plotly. (o. J.). Python | Plotly. https://plotly.com/python/getting-started/, letzter Abruf am 25.10.2020

[12] McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (2. Aufl.). O'Reilly Media.