Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

THILO LARS KRATZER
MASTER THESIS

ENTWICKLUNG EINES INTELLIGENTEN RAUMPLANUNGSSYSTEMS ANHAND SENSOREN FÜR LUFTQUALITÄT UNTER ANWENDUNG VERSCHIEDENER MACHINE UND DEEP LEARNING VERFAHREN

Eingereicht am 4. November 2019

Betreuer: Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 4. November 2019

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see https://creativecommons.org/licenses/by/4.0/

Erlangen, 4. November 2019

Abstract

The thesis is targeting the question to predict a room's occupation status without help of intrinsic sensors such as motion sensors or noise. Based on quantitive information about air composition, several machine learning algorithms are supposed to decide whether a meeting is terminated already, to release the specific resource for another booking. A second aspect aims at qualitative information. Studies showed that concentration decreases dramatically in bad air surroundings. Gathered subjective data and modern scientific approaches will be used to evaluate the indoor air quality and improve people's satisfaction.

Zusammenfassung

Im Fokus dieser Arbeit steht die Fragestellung, ob der Belegungsstatus eines Besprechungsraums auch ohne Zuhilfenahme intrinsischer Sensoren wie Bewegungsmeldern oder Lautstärke zuverlässig bestimmt werden kann. Auf Basis quantitativer Informationen der Luftqualität sollen verschiedene Lernalgorithmen eigenständig entscheiden, ob ein Meeting bereits frühzeitig beendet ist, um entsprechende Ressourcen wieder freizugeben.

Im Zuge der Arbeit werden Recurrent Neural Networks (RNN), Support Vector Maschinen (SVM) und Random Forests ausgewertet. Mithilfe der RNN konnte eine Genauigkeit von 94% erzielt werden. Die Netze tendieren allerdings zur Überanpassung (engl. *Overfitting*) an die Trainingsdaten, weswegen eine Übertragung auf unbekannte Daten oder neue Räume nicht realisiert werden konnte. Die klassischen Ansätze von SVM und Random Forests lieferten ähnlich gute Ergebnisse. Auf den Trainingsdaten konnte eine True positive rate von 89% bzw. bei Random Forests sogar 91% erzielt werden. Eine Vorhersage auf eine unbekannte Testmenge erreichte eine Genauigkeit von 74%.

Ein weiterer Aspekt bezieht qualitative Auswertungen mit ein. Verschiedene Studien haben gezeigt, dass Aufmerksamkeit sowie Produktivität von Menschen in Besprechungen nach einiger Zeit drastisch abnehmen und schon kurze Pausen sowie frische Luft die Leistung sofort wieder anheben. Im Rahmen der Arbeit werden daher Einschätzungen der Teilnehmer über die Qualität der Besprechung gesammelt. Die Daten sollen dazu genutzt werden, in Zukunft rechtzeitig einen Abfall der Konzentrationsfähigkeit oder unproduktive Meetings zu prognostizieren und die Effektivität insgesamt zu steigern. Zur Berechnung der Luftqualität werden aktuelle Verfahren der Wissenschaft genutzt, relevante Schadstoffe und thermische Einflüsse quantitativ auszuwerten.

Inhaltsverzeichnis

1 Anforderungen **1**
1.1 Erweiterung des Zustandsraums 2
1.2 Luftqualität während Besprechungen 7
1.3 Nicht-funktionale Anforderungen 8

2 Architektur und Design **9**
2.1 Systemlandschaft der Sensorik 10
2.2 Messung qualitativer Daten 14
2.3 Erweiterung des bestehenden Systems 15

3 Implementierung **17**
3.1 Messung und Sicherung der Sensordaten 17
3.2 Aufbereitung der gesammelten Daten 19
3.3 Klassische Machine Learning Verfahren 21
3.4 Deep Learning Verfahren 27
3.5 Luftqualität in geschlossenen Räumen 36

4 Auswertung und Ergebnisse **40**
4.1 Klassische Modelle 40
4.2 Neuronale Netze 48
4.3 Luftqualität und Indoor Air Quality Index 59

Anhänge **61**
Anhang A Dashboard der Sensorik 61
Anhang B Verkabelung der Sensoren 62

Literaturverzeichnis **63**

1 Anforderungen

Um den Belegungsstatus von Besprechungsräumen anzuzeigen, gibt es zahlreiche Möglichkeiten. Im Rahmen eines Team-Workshops bei der *Method Park Engineering GmbH* wurde zur Lösung eine digitale Belegungsanzeige mit Hintergrundbeleuchtung entwickelt. Als Programmiersprache wurde *Python* gewählt, da dafür bereits fertige Bibliotheken zur Kommunikation mit der *Microsoft Graph* API (Programmierschnittstelle, engl. *Application Programming Inferface*) bzw. den *Office 365-Diensten* und zum Ansteuern der *WS2812b RGB LEDs* als Open Source zur Verfügung stehen (Archer, Melborn & Casanovas, 2018; Garff, 2018). Die Software läuft auf einem *Raspberry Pi 3 Model B+*, einem kostengünstigen, aber leistungsstarkem Einplatinencomputer mit WLAN Funktion und *Raspbian „Strech“* Betriebssystem, welches alle benötigten Laufzeitumgebungen bereits vorinstalliert hat (Kern, 2018).

Art und Umfang der Anforderungen an Softwaresysteme sind vielseitig. Das Institute of Electrical and Electronics Engineers (IEEE) definiert Anforderungen als Bedingung oder Fähigkeit, die eine System oder eine Systemkomponente besitzen muss, um einen Vertrag, eine Norm oder ein anderes Dokument zu erfüllen (IEEE, 1990). Typischerweise unterscheidet man zwischen funktionalen und nicht-funktionalen Anforderungen.

Funktionale Anforderungen spezifizieren dabei eine Aufgabe, die das System in der Lage sein muss, auszuführen. Im Gegensatz dazu beschreiben nicht-funktionale Anforderungen gewisse Eigenschaften oder Attribute des Systems, beispielsweise bezüglich der Qualität oder Sicherheit. Softwarequalität wiederum umfasst Funktionalität, Effizienz, Wartbarkeit, Benutzbarkeit, Zuverlässigkeit, Übertragbarkeit sowie Sicherheit und Kompatibilität des Systems (ISO/IEC 25010, 2011). In diesem Kapitel soll der Fokus vor allem auf den funktionalen Anforderungen liegen.

1.1 Erweiterung des Zustandsraums

Die aktuell erreichbaren Zustände sind in Abbildung 1.1 dargestellt. Nach erfolgreicher Herstellung einer Netzwerkverbindung fragt die Software des *Raspberry Pi* alle zehn Sekunden den aktuellen Belegungsstatus entsprechend des Buchungskalenders vom *Microsoft Exchange Server* ab. Dieser liefert den aktuell laufenden oder als nächstes geplanten Termin, welcher schließlich auf dem Display entsprechend dargestellt wird. Weiter ist zu sehen, dass eine spontane Buchung des Raums (*Ad-hoc*) nur möglich ist, wenn die Ressource gemäß Serverantwort nicht bereits belegt und noch mindestens 15 Minuten frei ist. Ad-hoc-Buchungen dürfen standardmäßig ohne weitere Einschränkung direkt am Display beendet werden. Sie werden im Microsoft Office Kalender gespeichert und bei Beendigung mit dem Server synchronisiert.

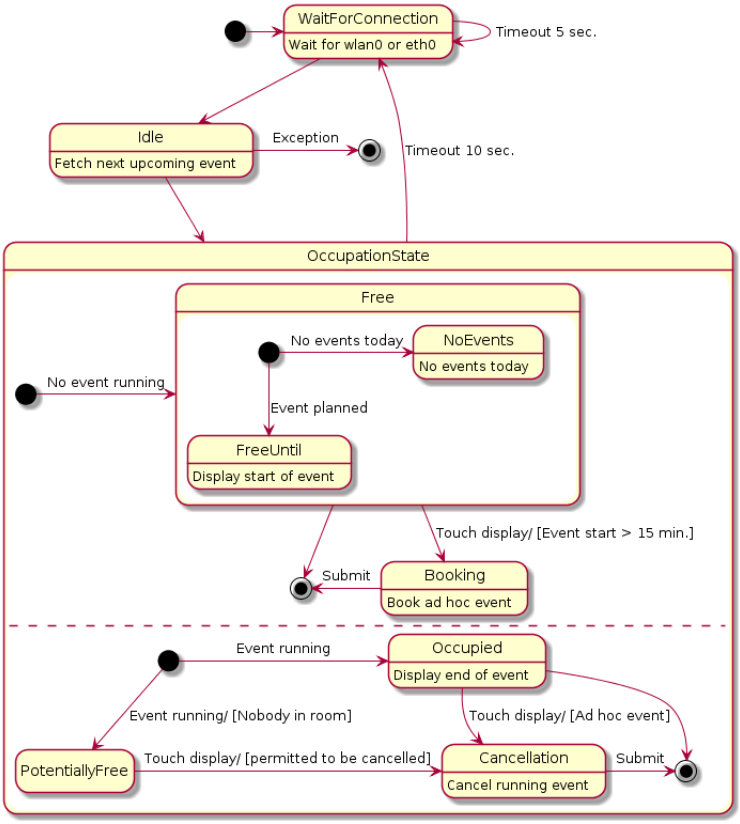


Abbildung 1.1: Zustandsdiagramm der Belegungsanzeige

Als gängiger Fall zeigte sich in der Vergangenheit, dass Besprechungen häufig vorzeitig beendet sind, der Raum durch den Kalendereintrag allerdings weiter blockiert ist. Nach Abwägung verschiedener Szenarien wurde entschieden, eine Beendigung gewöhnlicher Termine am Display direkt nicht zu implementieren, um einen möglichen Missbrauch dieser Möglichkeit zu vermeiden. Als Erweiterung des oben beschriebenen Softwaresystems soll in dieser Arbeit stattdessen die Frage geklärt werden, ob der Belegungsstatus auch intrinsisch, also ohne direkte Kommunikation mit dem Kalender, bestimmt werden kann, um den Raum darauf basierend freigeben oder reservieren zu können.

Durch Messung der Luftqualität soll die Software neben den binären Zuständen „`Free`“ und „`Occupied`“ einen dritten Zustand „`PotentiallyFree`“ einnehmen können. In diesem Zustand soll das erneute Buchen des Raumes unter bestimmten Umständen erlaubt sein. Grundsätzlich nicht überschrieben werden dürfen ganztägige Buchungen. Abhängig von der Dauer des Kalendereintrags soll die Freigabe erst nach einer vordefinierten Zeit erfolgen. Ad-hoc-Termine dürfen jederzeit automatisch beendet werden. Je länger die Zeitspanne des Termins, desto länger soll eine Freigabe hinausgezögert werden. Bei Seminaren mit einer Dauer von vier oder acht Stunden sind beispielsweise Pausen üblich, in denen sich keine Personen mehr im Raum befinden. Insgesamt ergibt sich daraus folgende funktionale Anforderung R1:

> R1: Das Buchen eines Besprechungsraumes soll möglich sein, wenn ein anderer Termin frühzeitig beendet wurde.

Auf Basis quantitativer Informationen soll das System eigenständig entscheiden, ob ein Meeting bereits frühzeitig beendet ist, um die entsprechende Ressource wieder freizugeben. Für den Zustandsübergang werden Messwerte der Temperatur, der Luftfeuchtigkeit, des Luftdrucks, des CO_2-Gehalts sowie Konzentrationen weiterer Gase abgefragt. Die Informationen werden von entsprechenden Sensoren gemessen und anschließend via Message Queue Telemetry Transport (MQTT) Protokoll im Netzwerk veröffentlicht. Die interne Weiterverarbeitung im System erfolgt mittels eines zuvor trainierten Lernalgorithmus, der den Zustand an das Display weitergibt. Eine detaillierte Beschreibung der Architektur findet sich später in Kapitel 2.

		Ground truth	
		`Free`	`Occupied`
Prediction	`Free`	True positive (TP)	False positive (FP)
	`Occupied`	False negative (FN)	True negative (TN)

Tabelle 1.1: Wahrheitsmatrix der erreichbaren Zustände

Die Zuverlässigkeit der Algorithmen soll anhand der *Precision*, auch Positive predictive value (PPV), gemessen werden. Dieser Term gibt den Anteil der True positives an der Gesamtheit aller als positiv (`Free`) klassifizierten Beispiele gemäß Wahrheitsmatrix (engl. *Confusion matrix*) in Tabelle 1.1 an. Die *Ground truth* umfasst dabei den tatsächlichen Zustand des Raums, unabhängig vom vorliegenden Kalendereintrag. Als *Prediction* werden die Vorhersagen eines spezifischen Algorithmus bezeichnet.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Dieses Verhältnis ist von besonderer Bedeutung, da für ein zuverlässiges System vor allem die Anzahl der False positives (Fehler 1. Art bzw. α-Fehler) gering sein soll, was ebenso eine hohe Precision impliziert. Es muss vermieden werden, dass ein Raum als frei klassifiziert wird, obwohl sich Personen darin befinden. Im Gegensatz dazu kann zunächst vernachlässigt werden, dass der Raum als belegt klassifiziert wird, wenn sich keine Personen darin befinden. Es ist davon auszugehen, dass andernfalls ein Kalendereintrag vorliegt.

Der Zusammenhang aller korrekten Klassifizierungen kann über die sog. Receiver Operating Characteristic (ROC) Kurve dargestellt werden. Als weitere wichtige Kennzahl müssen dazu auch die False negatives (Fehler 2. Art bzw. β-Fehler) betrachtet werden. Das Verhältnis der False negatives zu allen tatsächlich positiven Datensätzen wird als *False negative rate* oder *Miss rate* bezeichnet (Powers, 2008).

$$\text{False negative rate (FNR)} = \frac{\text{FN}}{\text{FN} + \text{TP}}$$

Es impliziert einen kleinen Fehler 2. Art, wenn auch das Verhältnis klein ist. Eine inkorrekte Klassifizierung als belegt soll vermieden werden, wenn sich tatsächlich keine Personen im Raum befinden. Zur Bestimmung eines idealen Klassifizierers mithilfe der ROC muss schließlich abgewogen werden zwischen *Sensitivity* (True positives) und *Specificity* (True negatives). Ein Klassifizierer ist also besser, je genauer er die Zustände „`Free`“ und „`Occupied`“ differenziert.

$$\text{Sensitivity} = \text{True positive rate} = 1 - \text{FNR}$$

$$\text{Specificity} = \text{True negative rate} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

$$\text{Fall-out} = \text{False positive rate} = 1 - \text{Specificity}$$

Das Konzept reicht zurück bis zum 2. Weltkrieg und wurde ursprünglich verwendet, um Radarsignale zu analysieren und Rauschen zu erkennen (Gonçalves, Subtil, Rosário & De Zea Bermudez, 2014). In den 50er Jahren wurden ROC Kurven in der Psychologie und Medizin dazu genutzt, menschliche Sensorik zu bewerten (O'Mahony & Hautus, 2008). Auch wenn Entscheidungsmodelle mithilfe von ROC Kurven trainiert werden können, so werden sie gewöhnlicherweise erst zur Evaluierung verwendet. Diese Art der Auswertung ist zwar nicht nur für binäre Probleme geeignet, dennoch soll der Fokus nachfolgend darauf gesetzt werden (Fan, Upadhye & Worster, 2006).

Wie die nachfolgende Abbildung 1.2 zeigt, wird der ROC Space durch Sensitivity und Fall-out aufgespannt, stellt also den Zusammenhang von True positives („Nutzen“) und False positives („Kosten“) dar. Auf der horizontalen Achse wird dabei die False positive rate und auf der vertikalen Achse die True positive rate aufgetragen. Je höher die Sensitivity, desto zuverlässiger werden positive Beispiele klassifiziert. Andersherum, je geringer der Fall-out, also je größer die Specificity, desto genauer können negative Beispiele erkannt werden.

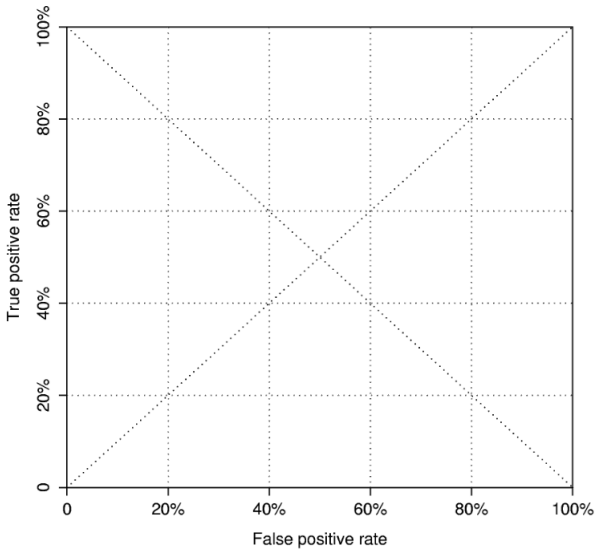


Abbildung 1.2: Receiver Operating Characteristic Space

Die Punkte $(0, 0)$ und $(1, 1)$ repräsentieren daher die untrainierten Klassifizierer „immer negativ“ und „immer positiv“, die Punkte $(0, 1)$ und $(1, 0)$ hingegen den idealen Klassifizierer bzw. einen, der für jedes Beispiel falsch liegt. Alle Punkte (p, p) auf der winkelhalbierenden Diagonale beziehen sich auf zufällige Klassifizierer mit Wahrscheinlichkeit p für positiv (TP) und $1 - p$ für negativ (FP). Der ROC Space erlaubt es, verschiedene Klassifizierer miteinander zu vergleichen und denjenigen zu wählen, der dem Punkt $(1, 0)$ am nähsten ist. Für trainierte Klassifizierer erstrebenswert sind konvexe Kurven oberhalb der Winkelhalbierenden (Flach, 2003). Für das fertige System sind insgesamt Klassifizierer mit geringem Fall-out solchen mit hoher Sensitivity vorzuziehen, da vor allem die zuverlässige Freigabe von Ressourcen das zentrale Problem umfasst.

1.2 Luftqualität während Besprechungen

Verschiedene Studien haben gezeigt, dass Aufmerksamkeit sowie Produktivität von Menschen in geschlossenen Räumen nach einiger Zeit drastisch abnehmen und schon kurze Pausen sowie frische Luft die Leistung sofort wieder anheben. Auch wenn es mit insgesamt etwas weniger als 0,05% (= 500 ppm) der Umgebungsluft vermeintlich obsolet wirkt, trägt vor allem der CO_2-Level einen großen Teil zur Konzentrationsfähigkeit bei. Zwar sorgen erst andauernde CO_2-Konzentrationen von über 20.000 ppm (Millionstel, engl. *Parts per million*) für ernsthafte Komplikationen wie Atembeschwerden oder Kopfschmerzen, für einen 8-Stunden-Arbeitstag wird allerdings dennoch empfohlen, einen Durchschnitt von maximal 5.000 ppm nicht zu überschreiten (OSHA, 1990). Es wurde nachgewiesen, dass bei Konzentrationen ab 1.000 ppm Schwierigkeiten bei der Entscheidungsfindung bereits moderat aber merklich zunehmen. Bei 2.500 ppm ist diese bereits erheblich gehemmt (Allen et al., 2015; Satish et al., 2012).

Eine intelligente Anzeige soll frühzeitig auf einen solchen Anstieg des CO_2-Gehalts hinweisen und damit insgesamt zur Produktivität im Unternehmen und zur Gesundheit und Zufriedenheit der Angestellten beitragen.

R2: Die Qualität der Besprechung soll anhand der gemessenen Luftqualität bewertet werden.

Zur Erweiterung des Systems sollen im Besprechungsraum außerdem drei LEDs installiert werden, die durch entsprechende intuitive Farbgebung (rot – gelb – grün) auf den Zustand und die Qualität der Luft hinweisen. Das Display soll um eine äquivalente, vereinfachte Darstellung der Messwerte erweitert werden. Zusätzlich sollen weitere Details wie der Verlauf bzw. die Entwicklung seit Beginn des Meetings auf einem separaten Screen darstellbar sein.

R3: Der Verlauf verschiedener Kennzahlen soll dokumentiert und angezeigt werden können.

1.3 Nicht-funktionale Anforderungen

Eine Reihe weiterer Anforderungen ergibt sich aus den oben beschriebenen Punkten bereits. So folgt aus Abschnitt 1.1 beispielsweise unmittelbar, dass das System zuverlässig sein soll. Zuverlässig meint dabei in erster Linie eine möglichst hohe Sensitivity neben ebenso hoher Specificity. Weiter soll die Ausfallzeit (engl. *Downtime*) so gering wie möglich gehalten werden, um zu jeder Zeit auf die Funktionalität des Systems zurückgreifen zu können.

Die Implementierung soll außerdem den Ansprüchen an *Clean Code* Genüge leisten. Martin (2002) beschreibt in seinem Buch fünf Prinzipien für gutes Software Design und Architektur.

R4: Das System soll einfach erweiterbar und nutzerfreundlich übertragbar sein.

Ein wesentlicher Fokus soll auf das *Open-Closed Principle* gelegt werden. Das System soll offen für Erweiterungen, das grundsätzliche Verhalten aber nicht extern beeinflussbar sein. Neue Sensoren sollen einfach ergänzt und weitere Räume schnell in Betrieb genommen werden können. Auch verschiedene Algorithmen und Modelle sollen einfach ersetzt und erweitert werden können.

2 Architektur und Design

Um die verschiedenen Komponenten zu realisieren, wird die bestehende Systemlandschaft erweitert. Neben den Sensoren spielt dabei vor allem eine Datenbank eine wichtige Rolle, um die essentielle Datenbasis für die Lernalgorithmen zu bieten. Das nachfolgende Kapitel soll Einblick in die Funktionsweise der einzelnen Bestandteile bieten und somit zum Verständnis für den Gesamtzusammenhang der Software beitragen.

Name	**Größe**	**Kapazität**	**Türen**	**Fenster**	**Klimaanlage**
Speaker's Corner	60 m^2	16	2	4	ja
Round Table	30 m^2	6	1	3	ja
Europa	20 m^2	4	1	1	nein

Abbildung 2.1: Abmessungen der ausgestatteten Besprechungsräume

Insgesamt werden drei verschiedene Besprechungsräume mit Messstationen ausgestattet, um eine höhere Varianz an geeigneten Trainingsdaten zu erhalten. Die Abmessungen der einzelnen Räume können Abbildung 2.1 entnommen werden. Die Räume *Round Table* und *Europa* haben dabei etwa die gleiche Größe, wobei in ersterem zusätzlich eine Klimaanlage installiert ist. Bei *Speaker's Corner* handelt es sich um einen Seminarraum für bis zu 16 Personen, ebenso mit Klimaanlage sowie zwei Türen und mehreren Fenstern.

2.1 Systemlandschaft der Sensorik

Für die Auswertung des Belegungsstatus gemäß Anforderung R2 werden Temperatur, Luftfeuchtigkeit, Luftdruck, CO_2-Gehalt sowie Konzentrationen weiterer Gase gemessen. Dafür wird zum einen ein *BME680 Sensor* der Bosch Sensortec GmbH (2019) sowie ein *K30 FR CO_2 Sensor* der Firma SenseAir AB (2015) verwendet. Der *BME680* ist ein Sensor, der basierend auf bewährten Messtechniken die Messung von Gaswiderstand, Luftfeuchtigkeit, Druck und Temperatur vereint und typischerweise in kleineren Geräten wie Handys, Tablets oder Smart Watches verbaut ist. Zu beachten gilt, dass der Sensor einen Widerstandswert mit Gesamt-VOC-Gehalt misst, also den Anteil flüchtiger organischer Verbindungen (engl. *volatile organic compounds*) in der Luft, wobei sich keine einzelnen Gase oder Alkohole unterscheiden lassen. Grundsätzlich gilt, dass der Gaswiderstand mit nachlassender Luftqualität ebenso sinkt, wobei der korrekte Bezugswert sehr individuell sein kann und der Sensor zunächst einige Minuten benötigt, um sich richtig zu kalibrieren.

Beim *K30 FR* handelt es sich um ein einfaches, vorkalibriertes CO_2-Transmitter-Modul, welches sich durch seine kompakte Bauweise sowie den geringen Stromverbrauch ebenso für mikroprozessor-basierte Überwachungen eignet. Beide Sensoren werden an einen *Raspberry Pi Zero W* angeschlossen. Durch seine offene Bauweise und die integrierte WLAN Funktion ermöglicht der schlanke Single Core Rechner mit 40-Pin GPIO (General Purpose Input Output) den Aufbau von WLAN-gesteuerten Aktoren und Sensoren. Aufgrund der großen Community ist außerdem eine Vielzahl an Add-Ons und Programmbibliotheken zur hardwarenahen Programmierung als Open Source Software verfügbar.

Neben dem *Raspberry Pi* eignete sich beispielsweise ebenso ein *ESP8266 Microcontroller* der Firma Espressif Systems (2015). Dieser unterstützt Arduino/C++-basierte, hardwarenahe Programmierung. Sowohl das Auslesen der Sensordaten über den seriellen I^2C Datenbus, als auch die Kommunikation über MQTT führten allerdings zum regelmäßigen Absturz. Aufgrund diverser Probleme bei der konstanten und zuverlässigen Inbetriebnahme wurde dieser Ansatz daher nicht weiter verfolgt.

Die zentrale Einheit bildet ein virtueller Server unter dem lokalen Hostnamen `mrd-server.er.methodpark.de`. Der Server ist vor allem für den Empfang und die persistente Verarbeitung der Trainingsdaten zuständig. Die Daten werden dazu mittels SQLite in eine relationale Datenbank gespeichert. Die C-basierte und daher systemunabhängige SQLite-Bibliothek bietet den Vorteil, keinen separaten Server Prozess zu benötigen, sondern über entsprechende Lese- und Schreibvorgänge direkt mit der Datei auf der Festplatte zu interagieren. Da derartige Zugriffe bis zu 35% schneller als gewöhnliche Dateizugriffe sein können, eignet sich SQLite vor allem für den Einsatz in Systemen mit hohem Umsatz (Hipp, Kennedy & Mistachkin, 2019).

Zusätzlich wird über Port 1880 ein *Node-RED* Dashboard angeboten, um über das lokale Netzwerk jederzeit aktuelle Messwerte ablesen zu können. Node-RED ist ein ursprünglich von IBM programmiertes Entwicklungswerkzeug, basierend auf *Node.js*. Die Software ermöglicht es, Anwendungsfälle im Bereich des Internet of Things (IoT) einfach in JavaScript umzusetzen und grafisch darzustellen (Lewis, 2016). Im Dashboard (siehe Anhang A) werden neben den aktuellen Messungen auch Tagesverlauf sowie die Kurven der letzten sieben Tage dargestellt. Die Lösung mittels des Services bildet eine Vorstufe zur Erfüllung der Anforderung R3, der Darstellung aktueller Messwerte.

Für die Netzwerkkommunikation wird das ebenfalls von IBM entwickelte, offene MQTT Nachrichtenprotokoll verwendet, welches sich aufgrund seiner einfachen Anwendung vor allem für Machine-to-Machine (M2M) Kommunikation im IoT eigenet (Banks, Briggs, Borgendale & Gupta, 2019). MQTT ist ein Client-Server basiertes publish-and-subscribe Protokoll, bei dem kompakte Binärpakete über einen zentralen Verteiler (engl. *Broker*) an interessierte Teilnehmer versendet werden. Zu seinen Vorteilen zählt vor allem die einfache Einbindung in IoT-Anwendungen sowie die geringe Belastung des Netzwerkverkehrs und dadurch gute Eignung für ressourcenarme Geräte mit schlechter Internetverbindung (Gerstl, 2018).

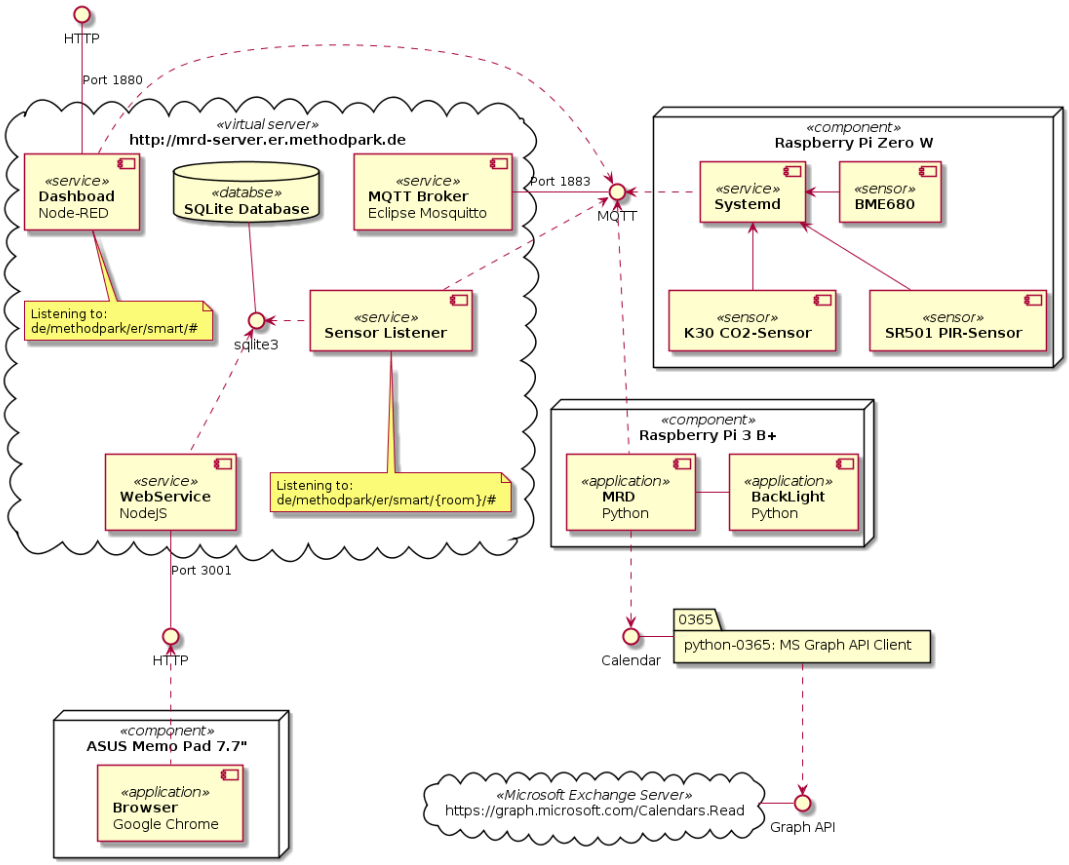


Abbildung 2.2: Komponentendiagram der Sensorik

Wie in Abbildung 2.2 zu sehen, verbindet sich der *Raspberry Pi* mit dem MQTT Broker über Port 1883 und veröffentlicht die gemessenen Werte unter dem Topic `de/methodpark/er/smart/{room}/{sensor}`. Der Wert für `sensor` ergibt sich abhängig vom jeweiligen Sensor als `carbon_dioxide`, `temperature`, `humidity`, `pressure` oder `gas_resistance`.

Neben Broker und Datenbank läuft ein Python-Skript auf dem Server (Listener), welches für den Empfang der Sensordaten zuständig ist und über das Paket `sqlite3` (Häring, 2015) direkt mit der Datenbank in Verbindung steht. Das Skript abonniert die entsprechenden Subtopics und erwartet die übermittelten Daten als String der Form `'{"Key":"Value", ...}'`. Um die Werte eines Messwert-Datensatzes eindeutig zuordnen zu können, werden außerdem Raum- sowie Sensorname und entsprechende -identifier gesendet. Außerdem werden der gemessene Wert sowie dessen Einheit übermittelt. Der Listener ergänzt vor dem Ablegen in

die Datenbank den Zeitstempel des Empfangs und die ID des MQTT Clients.

Um die gemessenen Trainingsdaten mit der Zielvariable Raumbelegung verknüpfen zu können, wird auf jedem *Meeting Room Display* ein zusätzlicher MQTT Client implementiert. Dieser ist über die *Microsoft Graph API* mit dem Exchange Server und dem Raumkalender verbunden und fragt in regelmäßigen Abständen den Belegungsstatus ab. Um die Daten an den Server zu übertragen, veröffentlicht der Client die Werte unter dem Subtopic `occupation_state`. Um dem Datenbankschema treu zu bleiben, wird dabei als Wert 0 für frei und 1 für belegt abgespeichert. Zusätzlich wird die Anzahl der anwesenden Personen anhand der Terminzusagen als Einheit vermerkt.

Um, bezogen auf die tatsächliche Belegung, eine höhere Genauigkeit zu erhalten, wird an die Sensorstation zusätzlich ein *HC-SR501 Passive Infra Red Motion Sensor* angeschlossen, welcher anhand der gemessenen Strahlung Bewegungen erkennt, um daraus ein digitales Signal zu berechnen. Der messbare Bereich des Sensors umfasst $110° \times 70°$ bei einer Reichweite von etwa sechs Metern (Fried, 2018). Die Bewegungen werden im Abstand von Zehntelsekunden erfasst und über die Dauer von einer Minute gemittelt. Der so errechnete relative Wert der Bewegungen pro Minute wird über MQTT auf dem Subtopic `motion` veröffentlicht und in der Datenbank gespeichert. Die Bewegungsdaten können später mit den Kalenderdaten abgeglichen werden, um ungenaue Trainingsdaten zu relativieren.

Allgemein hat eine Lösung mittels MQTT den Vorteil, dass die Sensordaten nicht nur an die Datenbank geleitet werden. Auch das aktuelle System kann die Nachrichten empfangen und entsprechend weiterverarbeiten, um diese am Display anzuzeigen. Eine Erweiterung um einen trainierten Lernalgorithmus bereitet dahingehend keine Probleme. Durch entsprechende Kapselung der einzelnen Systeme sind diese problemlos zur Laufzeit ersetzbar. Die interne Realisierung im Programmcode sowie die Erweiterung des bestehenden Systems wird später in Abschnitt 2.3 genauer erläutert.

2.2 Messung qualitativer Daten

Um adäquate Aussagen über die Qualität von Besprechungen treffen zu können, werden umfassende Trainingsdaten benötigt. Zu diesem Zweck wird auf dem virtuellen Server ein einfacher *Node.js* Webservice implementiert. Als asynchrone, Event-basierte Laufzeitumgebung eignet sich Node speziell für die Entwicklung von skalierbaren Netzwerkanwendungen (OpenJS Foundation, 2019). Der Service wird mithilfe von *Express*, einem einfachen und flexiblen Web-Framework für *Node.js* Anwendungen, realisiert und wartet auf Anfragen über Port 3001.

Abbildung 2.3: HTML Index-Seite des Node.js Services

Für jeden Besprechungsraum wird ein Tablet installiert, über welches die entsprechende Bewertung abgegeben wird. Abbildung 2.3 zeigt die Index-Seite des Services. Durch Klick auf einen der Smileys wird ein entsprechender Datensatz in der Datenbank angelegt. Dabei wird als Wert die Farbe auf eine der Ganzzahlen $\{0, 1, 2\}$ abgebildet und die textuelle Repräsentation $\{\texttt{green}, \texttt{yellow}, \texttt{red}\}$ als Einheit abgespeichert. Verschiedene Räume können über ein passwortgeschütztes Dropdown-Feld ausgewählt werden. Die Webseite wird anschließen über die *KioWare Lockdown Kiosk Mode* App geöffnet. Die App hindert den Nutzer daran, weitere Seiten aufzurufen oder eine andere Funktionalität des Tablets zu nutzen.

2.3 Erweiterung des bestehenden Systems

Die bestehende Software der *Meeting Room Displays* wurde in einer hexagonalen Architektur programmiert, auch bekannt als Ports and Adapters, um möglichen Erweiterungen und neuen Komponenten offen gegenüber zu sein. Es handelt sich dabei um ein modernes Software Design Pattern, welches von Alistair Cockburn (2005) entwickelt wurde. Wesentlicher Bestandteil bei Ports and Adapters ist, dass die Anwendung die zentrale Einheit des Systems ist, ohne dabei die Programmlogik zu beeinflussen. Jegliche Kommunikation der angeschlossenen Komponenten findet ausschließlich mit der Anwendung, aber niemals untereinander statt. Aufgabe der Anwendung ist die entsprechende Weiterleitung der Ein- und Ausgaben über sogenannte Ports, in den meisten Fällen realisiert durch Schnittstellen. Dabei sollte die Anwendung kein Wissen über die angeschlossenen Adapter haben, von denen sie benutzt wird, sogenannte primäre Adapter, oder solche die sie benutzt, auch sekundäre Adapter (Vernon, 2013).

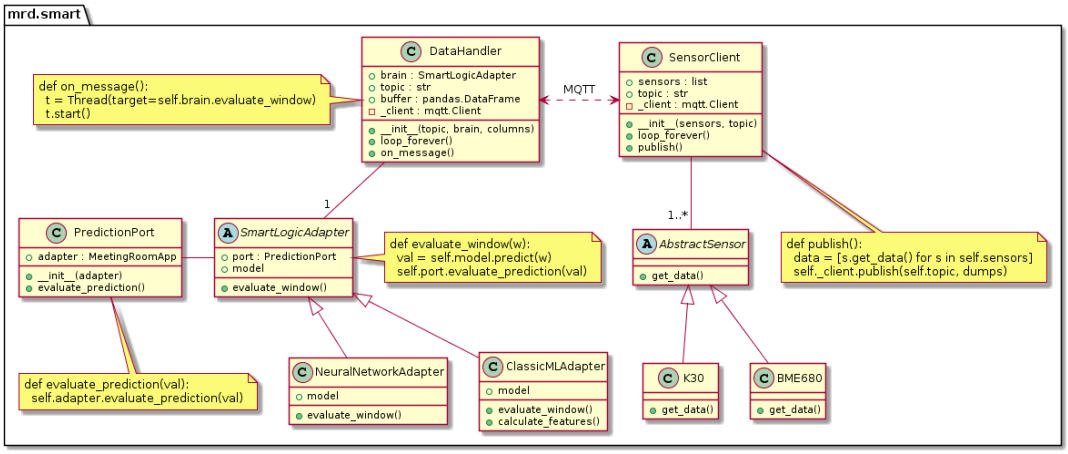


Abbildung 2.4: Klassendiagramm der neuen Komponenten

Das *Meeting Room Display* als zentraler Baustein besitzt aktuell nachfolgende Ports. Zunächst nutzt die Anwendung eine Methode zum Abfragen der Kalenderdaten beim Backend (Microsoft Exchange Kalender) und reicht die Information schließlich über einen definierten Port weiter an die angeschlossenen primären Adapter, das User Interface (UI) sowie ein über GPIO angeschlossenes LED-Ambilight. Weitere Aktionen gehen vom UI aus, beispielsweise bei der Buchung

bzw. beim Beenden eines Ad-hoc-Termins. Diese werden wiederum über einen Port zum Backend weitergereicht.

Abbildung 2.4 zeigt die neue Komponente `mrd.smart`. Über die Klasse `SensorClient` können verschiedene Sensoren angebunden werden. Um einheitliche Methoden sicherzustellen, wurde dazu die abstrakte Klasse `AbstractSensor` eingeführt. Die Werte werden in definierten Zeitintervallen abgefragt und anschließend via MQTT veröffentlicht. Die Kommunikation über MQTT ist an dieser Stelle sinnvoll, da die Sensoren nicht am gleichen Gerät angeschlossen sind, auf dem die Hauptanwendung läuft. Die Daten können auf diese Weise einfach und sicher über das lokale Netzwerk übertragen werden. Der Datenverkehr läuft dabei über einen zentralen Broker. Einzelne Anwendungen bzw. Räume werden dabei über individuell eindeutige Topics unterschieden.

Neben der Hauptanwendung wird eine Instanz der Klasse `DataHandler` gestartet. Sie abonniert die entsprechenden MQTT Topics und buffert die Daten für den benötigten Zeitraum. Liegen genug Daten für die gesetzte Fenstergröße vor, werden die Werte bzw. definierten Features an das trainierte Machine Learning Modell in der Klasse `SmartLogicAdapter` weitergereicht. In einem separaten Thread wird die Vorhersage für die übergebenen Werte getroffen und die Information über den entsprechenden Port `PredictionPort` an die App übermittelt.

3 Implementierung

3.1 Messung und Sicherung der Sensordaten

Wie in Abschnitt 2.1 bereits dargestellt, werden die verwendeten Sensoren an einen *Raspberry Pi Zero W* angeschlossen. Wie in Anhang B zu sehen, werden der *Raspberry Pi* und der *BME680* Sensor neben der jeweiligen 3.3 Volt Stromversorgung (VIN, rot) und der Erdung (GND, schwarz) an SCL und SCK („Clock“, blau) sowie SDA und SDI („Data“, gelb) miteinander verkabelt (Fried, 2019). Für die Steuerung der Sensordaten wurde jeweils ein Python-Skript implementiert. Der *BME680* wird mithilfe des Pakets `bme680` angesprochen. Dieses nutzt die serielle I^2C Schnittstelle, um den Sensor auszulesen (Howard & Macdonald, 2018). Die einzelnen Werte von Temperatur (°C), relativer Luftfeuchtigkeit (%), Druck (hPa) und Gaswiderstand (Ohm) werden nacheinander und mit einem Abstand von jeweils einer Sekunde per MQTT veröffentlicht. Dazu wird außerdem die entsprechende Einheit und der aktuelle Zeitpunkt übermittelt.

Der *K30 FR* wird, wie Anhang B ebenfalls zeigt, neben der 5 Volt Stromversorgung (rot) und der Erdung (schwarz) über die Pins TXD („transmit“, grün) und RXD („receive“, gelb) an den *Raspberry Pi* angeschlossen (CO2Meter Inc., 2019). Die gemessenen CO_2-Werte (ppm) werden alle fünf Sekunden über UART (Universal Asynchronous Receiver Transmitter) ausgelesen, was mithilfe des Python-Moduls `serial` (Liechtl, 2017) realisiert ist.

Der *SR501* Bewegungssensor benötigt ebenso eine 5 Volt Stromversorgung (rot) und Erdung (schwarz) und wird über GPIO ausgelesen (gelb).

Die Skripte zur Steuerung der Sensoren werden intern über je einen *systemd* Service, ein Hintergrundüberwachungssystem für Linux-Systeme, gesteuert. Entsprechende *Cron*-Jobs starten die Services um 7 Uhr morgens und stoppen diese abends um 20 Uhr, um keine unnötigen Daten außerhalb der Kernarbeitszeit zu produzieren. Der sogenannte *Cron*-Daemon hilft dabei, wiederkehrende Aufgaben zu automatisieren und dient der zeitbasierten Ausführung von Prozessen in unixartigen Betriebssystemen (Keller, 1999).

Auf dem virtuellen Server läuft außerdem ein Service, der die gesendeten Nachrichten empfängt, auswertet und an die Datenbank weiterleitet. Die Zentrale Einheit bildet, wie Abbildung 3.1 zeigt, die Klasse `MqttServer`, welche die Subklasse eines `DBAgent` übergeben bekommt und dann instanziiert. Diese Designentscheidnug wurde getroffen, damit die Konfiguration, wie bspw. der Pfad der Datenbankdatei, nutzerfreundlich über eine externe Datei angepasst werden kann. Auch weitere Einstellungen wie Host, Port und entsprechende Topics können über die externe Konfigurationsdatei bereitgestellt werden und werden während der Initialisierung von `MqttServer` ausgelesen. Die Datei unterteilt sich in die beiden Sections `mqtt` und `topics`.

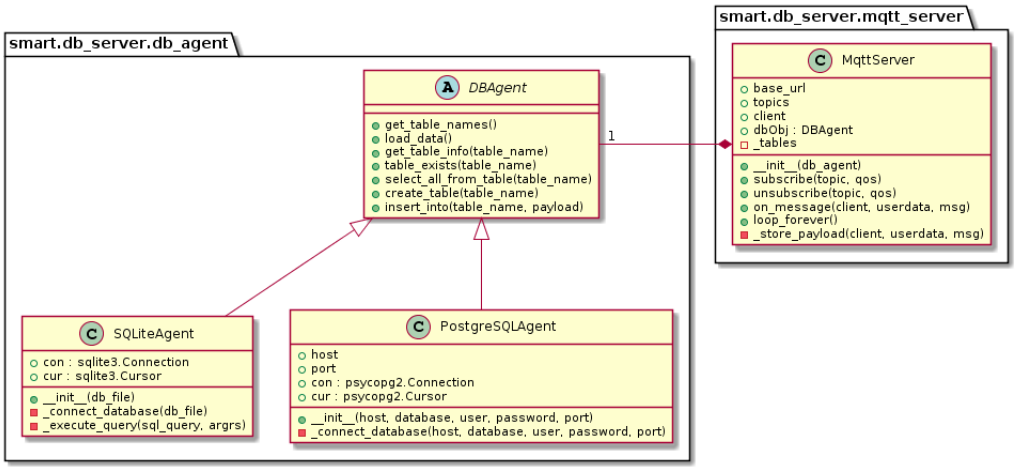


Abbildung 3.1: Klassendiagram des Datenbankservices

Mögliche Datenbanken, die aktuell genutzt werden können, sind SQLite sowie PostgreSQL. Der `SQLiteAgent` wurde so implementiert, dass eine Tabelle pro Topic bzw. Messwert angelegt und befüllt wird. Alle verfügbaren Tabellen können über die Methode `get_table_names()` abgefragt werden. Im `MqttServer` wird

zusätzlich ein Dictionary aus Topic und Tabellenname in der privaten Variable `_tables` gespeichert und bei jeder empfangenen Nachricht ggf. ergänzt. Weitere Datenbanktypen oder ein anderes Verhalten kann einfach ergänzt werden, indem die abstrakte Klasse `DBAgent` implementiert wird. Für das Skript wurde ein `systemd` Service eingerichtet, um sicherzustellen, dass der Server dauerhaft erreichbar ist.

3.2 Aufbereitung der gesammelten Daten

Für die Vorbereitung der gespeicherten Rohdaten wird eine Methode bereitgestellt, welche die Datenbank einliest und die einzelnen Tabellen miteinander vereint. Für das Einlesen der Datenbank kann eine `DBAgent`-Instanz mit der bereitgestellten Methode `select_all_from_table()` verwendet werden. Mithilfe der bereitgestellten Methode `load_data()` wird die gesamte Datenbank eingelesen. Die Tabellen werden dabei jeweils als `pandas.DataFrame` geladen und in einem Dictionary aus Tabellenname und Daten gespeichert.

Die geladenen Tabellen werden anschließend anhand der zugewiesenen Zeitstempel vereint. Da die Zeitstempel voneinander abweichen, kann kein exakter Merge vollzogen werden. Vielmehr muss, ausgehend von einer bestimmten Tabelle, der jeweils nächste Zeitstempel in einer definierten Umgebung gefunden werden. Praktisch ist dieser Left Outer Join in der Hilfsmethode `smart.util.data._merge_on_time()` implementiert. Dieses Vorgehen kann im Allgemeinen zwar dazu führen, dass Messwerte der weiteren Tabellen verloren gehen. Die kann allerdings vernachlässigt werden, wenn die Tabellen im Vorfeld der Größe (Anzahl der Datensätze) nach absteigend vereint werden. Ein Aufruf der Methode `smart.util.data.merge_columns()` stellt dies sicher. Um die einzelnen Spalten anschließend weiterhin unterscheiden zu können, wird der jeweilige Tabellenname dem Spaltennamen vorangestellt. Raum- sowie Sensor-ID und -name werden nicht mehrfach gespeichert.

Fehlende Belegungsdaten können im Nachhinein eingetragen werden. Die Methode `smart.util.data.fetch_missing_occupancy()` lädt dazu unter Verwendung der Klasse `o365_MqttClient` alle Veranstaltungen im gefragten Zeitraum vom Exchange Server und ergänzt für jeden unvollständigen Datensatz den jeweili-

gen Wert. Als weiteres Problem zeigt sich, dass in den gemessenen Daten etliche Stellen zu finden sind, an denen sich offensichtlich keine Personen im Raum befinden, der im Kalender gespeicherte Termin aber noch andauert. Dieser Zustand ist daran zu erkennen, dass die gemessenen Werte sich stabilisieren oder, wie beispielsweise bei CO_2, wieder deutlich sinken. Die Klassifizierung der einzelnen Zeitreihen wird anhand der gemessenen Bewegungsdaten entsprechend angepasst.

Wie den normalisierten Daten aus Abbildung 3.2 entnommen werden kann, war der Raum im Zeitraum von 14 bis 18 Uhr gebucht (rot). Im Bereich bis ca. 16:15 Uhr steigen CO_2 und relative Luftfeuchtigkeit, beide bedingt durch die Ausatemluft, sowie die Temperatur durch die natürliche Wärmestrahlung der anwesenden Personen nahezu linear an. Der Gaswiderstand im Vergleich dazu nimmt proportional ab, was ebenso auf eine sinkende Luftqualität zurückzuführen ist.

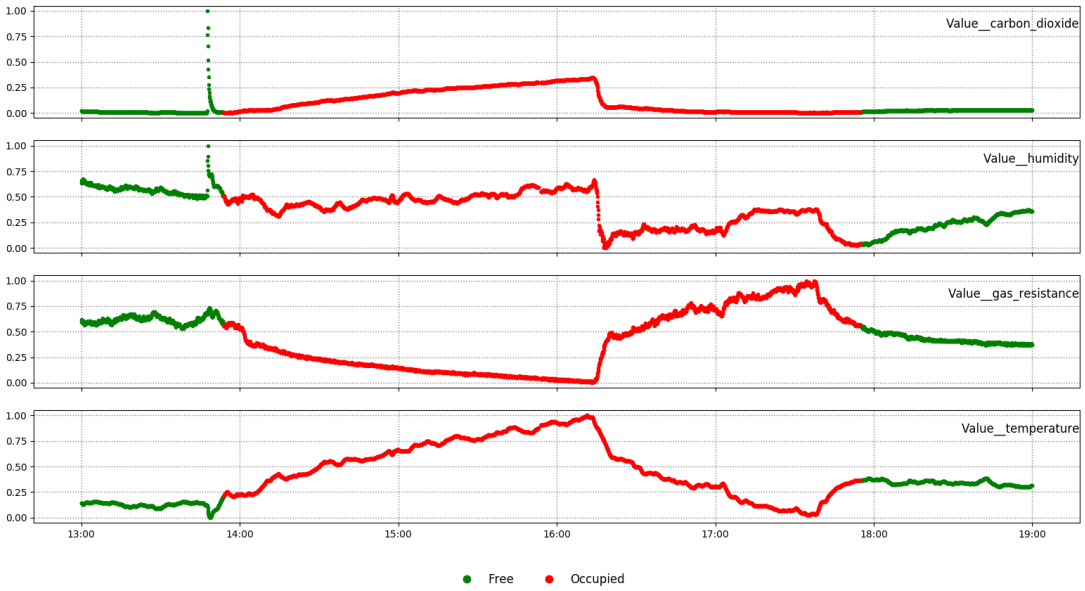


Abbildung 3.2: Ungenauigkeit der gemessenen Daten

Im zweiten Abschnitt nach 16:15 Uhr pendelt sich der CO_2-Gehalt auf einen konstanten Wert ein, Temperatur, Luftfeuchtigkeit und Gaswiderstand nehmen erneut den Ursprungszustand (grün) ein. Der abrupte Abfall der Kurven unmittelbar davor ist auf das Öffnen eines Fensters und die damit verbundene Vermischung mit der Frischluft zurückzuführen. Da die gemessenen Daten offensicht-

lich nicht den realen Belegungszustand widerspiegeln, müssen derartige Bereiche manuell identifiziert und bereinigt werden, um die Trainingsdaten so genau wie möglich zu halten. Zur Unterstützung können dabei die in Abschnitt 2.1 beschriebenen Daten der Bewegungssensoren herangezogen werden. Nach Abschluss der korrekten Aufbereitung werden die Daten als CSV-Datei gespeichert.

Die gesammelten Daten werden abschließend in zwei Fälle unterteilt. Etwa vier Wochen nach Beginn der Messungen wurde in beiden Testräumen eine Klimaanlage installiert, sodass ein direkter Vergleich der Werte nicht möglich ist. Die Daten werden getrennt voneinander standardisiert, um sie dennoch vergleichbar zu machen.

3.3 Klassische Machine Learning Verfahren

Während der letzten beiden Jahrzehnte erlangte vor allem der Umgang mit Zeitreihen den Ruf, mitunter das schwierigste Problem im Bereich des Data Mining zu sein. Eine univariate Zeitreihe $X = [x_1, x_2, \ldots, x_T]$ repräsentiert eine Sammlung T geordneter und über einen bestimmten Zeitraum gemessener Daten. Eine m-dimensionale mulivariate Zeitreihe $X = \left[X^1, X^2, \ldots, X^M\right]$ wiederum besteht aus M verschiedenen Zeitreihen $X^i \in \mathbb{R}^T$. Ein vollständiger Datensatz $D = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\}$ beinhaltet Paare (X_i, Y_i), wobei X_i eine uni- oder multivariate Zeitreihe ist, mit Y_i als korrespondierendem k-dimensionalem Zielvektor. Y beinhaltet dabei eine Wahrscheinlichkeitsverteilung der möglichen Klassen (Fawaz, Forestier, Weber, Idoumghar & Muller, 2019). Im vorliegenden Fall ist $M = 5$ die Anzahl der Sensorwerte.

Der Term Klassifizierung bezieht sich auf ein Teilgebiet des Supervised Learnings, wo eine Zuordnung zum Ergebnisvektor für eine Trainingsdatenmenge bereits bekannt ist. Algorithmen werden schließlich anhand der Trainingsmenge angepasst und häufig mittels eines unabhängigen Validierungssets bestätigt. Das Ziel von Supervised Learning ist daher, eine Abbildung der bekannten Eingaben x zur referenzierten Klasse y zu finden (Susto, Cenedese & Terzi, 2018).

Hintergrund und Definitionen

Die Zeitreihenklassifizierung kann grundsätzlich in zwei Bereiche unterteilt werden. Zum einen feature-basierte Klassifizierung; hier werden vor der Berechnung der Algorithmen einzelne zuvor definierte Features extrahiert und gespeichert. Dazu wird das ursprüngliche Signal $v(t) = (x_1, \ldots, x_n)$ in überlagerte Fenster definierter Größe k zerlegt, um auf diesem Teilbereich $w = (x_i, \ldots, x_{i+k})$ eine Menge an Features zu berechnen, die Form und Verlauf bestmöglich beschreiben. Weithin gängige Features umfassen beispielsweise Mittelwert, Standardabweichung, Extremwerte oder Entropie. Um relevante Informationen nicht zu verlieren, benötigt die richtige Auswahl geeigneter Features tiefgreifendes Domänenwissen und ist für gewöhnlich zeitintensiv. Auch die Wahl der richtigen Fenstergröße k ist alles andere als trivial (Hastie, Tibshirani & Friedman, 2009).

Der zweite Ansatz umfasst distanzbasierte Methoden. Hierbei müssen keine Features aufwendig berechnet werden, stattdessen wird die Ähnlichkeit von Signalfolgen anhand definierter Metriken gemessen. Das wohl populärste und eines der erfolgreichsten Beispiele für distanzbasierte Klassifizierung ist der *Nearest Neighbor Classifier* unter Berechnung der Euklidischen Distanz. Obwohl die zeitintensive Feature-Berechnung obsolet wird, ergibt sich nun allerdings ein Tradeoff zwischen der Komplexität bei der Wahl der richtigen Metrik und der Genauigkeit bei der Vorhersage von Testdaten, weswegen diese Art der Verfahren nachfolgend nicht weiter betrachtet werden (Susto et al., 2018).

Support Vektor Klassifizierung

Ein einfacher, sehr mächtiger Algorithmus aus dem Bereich der klassischen Mustererkennung sind sogenannte Support Vector Maschinen, die speziell für binäre Klassifizierungsprobleme konstruiert wurden. SVM eignen sich besonders für Supervised Learning und schaffen zumeist auch gute Ergebnisse ohne tiefgehendes A-priori-Wissen. In ihrem Vorgehen erzeugen SVM grundsätzlich eine die Daten linear trennende Gerade oder Hyperebene (engl. *hyper plane*) zwischen den beiden Klassen, sodass der Abstand (engl. *margin*) aller Punkte zur Ebene maximal wird. Daten, die im ursprünglichen Eingaberaum nicht linear separierbar sind, werden so lange in höhere Dimensionen transformiert, bis sie schließlich trennbar sind.

SVM basieren auf der Annahme, dass unbekannte, neue Punkte aus der gleichen Verteilung wie die Trainingsdaten kommen, wodurch eine zuverlässige Trennebene auch mit wenigen Trainingsdaten gut berechnet werden kann. Auch darauf basierend, dass für die Trennebene hauptsächlich Punkte mit kleinem Abstand zur Margin entscheidend sind, können SVM sehr effizient und ressourcensparend berechnet und gespeichert werden (Russell & Norvig, 2009).

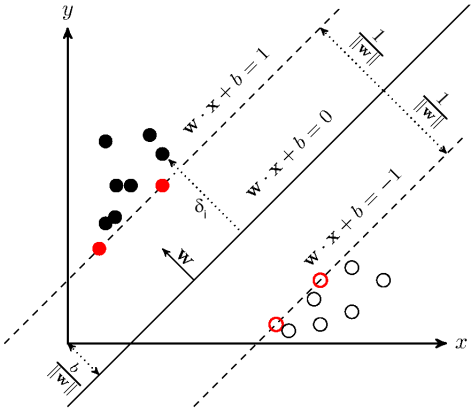


Abbildung 3.3: Lineare Trenngerade und Margin im 2D (Y. Peng, 2013)

Aus mathematischen Gründen gilt die Konvention, dass Klassen y_i mit ± 1 bezeichnet werden. Die Trennebene ergibt sich schließlich als Menge der Punkte $\{x_i \mid w \cdot x_i + b = 0\}$ mit Normalvektor w, orthogonal zur Trennebene, und der Entfernung b vom Ursprung, wodurch sie eindeutig bestimmt wird. Wie in Abbildung 3.3 zu sehen, gilt für alle Punkte x_1, x_2 auf dem Rand der Margin (rot), die sogenannten Stützvektoren, genau $w \cdot x_1 + b = +1$ bzw. $w \cdot x_2 + b = -1$ und für die Breite schließlich:

$$w(x_1 - x_2) = 2 \qquad \text{und daher} \qquad \tfrac{w}{\|w\|}(x_1 - x_2) = \tfrac{2}{\|w\|}$$

Die Trennung der beiden Klassen erfolgt dann anhand der Margin gemäß:

$$w \cdot x_i + b \leq -1 \Leftrightarrow y_i = -1$$
$$w \cdot x_i + b \geq +1 \Leftrightarrow y_i = +1$$

Beide Ungleichungen können zusammengefasst werden zu $y_i(w \cdot x + b) \geq 1$. Da die Breite der Margin insgesamt so groß wie möglich werden soll, kann dies als

das folgende Optimierungsproblem mit linearen Nebenbedingungen aufgestellt werden:

$$\begin{aligned} \max \quad & \frac{2}{\|w\|} \\ s.t. \quad & \forall i \ : \ y_i(w \cdot x + b) \geq 1 \end{aligned}$$

Durch die linearen Nebenbedingungen ist sichergestellt, dass alle gegebenen Feature-Vektoren tatsächlich maximalen Abstand zur Trennebene haben (He, Yao & Ye, 2017). Für den Fall, dass beide Klassen nicht linear trennbar sind oder die Berechnung zu zeitintensiv ist, gibt es eine Menge Strategien, um dennoch gute Parameter zu finden. Die wohl einfachste Methode ist, weitere *Slack Variablen* ξ_i einzuführen, um einen gewissen Fehler in den Trainingsdaten zuzulassen. Das Optimierungsproblem wird dann folgendermaßen erweitert:

$$\begin{aligned} \min \quad & \frac{1}{2}\|w\|^2 + \mu \sum_i \xi_i \\ s.t. \quad & \forall i \ : \ -(y_i(w \cdot x + b) - 1 + \xi_i \leq 1, \\ & \forall i \ : \ -\xi_i \leq 0 \end{aligned}$$

Da die meisten Optimierungsalgorithmen ein Minimierungsproblem erwarten, wurde zunächst das Vorzeichen umgedreht. Das Quadrat in der Zielfunktion verändert die Lösung des Problems nicht, aber vereinfacht die nötige Differentiation. Die Slack Variablen, die sicherstellen, dass Punkte auch auf der „falschen“ Seite der Margin liegen dürfen, werden schließlich als Fehlerterm mit Gewichtungsfaktor μ zur Breite der Margin addiert (Vapnik, 1995).

Für die Lösung des oben beschriebenen Problems gibt es viele Ansätze. Typischerweise wird eine Lösung mithilfe der Lagrange-Polynome und entsprechender Richtungsableitungen berechnet. Auch eine Transformierung in höhere Dimensionen oder eine Skalierung innerhalb des Raums sind gängige Verfahren, um die Daten trennbar zu gestalten und eine Support Vector Klassifizierung zu ermöglichen. Auf diese Verfahren soll an dieser Stelle allerdings nicht weiter eingegangen werden.

Entscheidungsbäume und Random Forest Classifier

Entscheidungsbäume (engl. *decision trees*) gehören ebenfalls zu den einfachsten Klassifizierungsverfahren und sind meist doch sehr zuverlässig. In vielen Bereichen in Industrie und Handel sind sie daher mitunter das erste Modell, das für Klassifizierungsprobleme herangezogen wird. Ein Entscheidungsbaum ist im weitesten Sinne eine Funktion, die einen Vektor aus diskreten Attributen als Eingabe erhält und einen einzigen Ausgabewert berechnet. Die Berechnung geschieht dabei in einer Abfolge verschiedener Tests. Jeder Knoten im Baum beinhaltet einen Test für genau ein Attribut des Eingabevektors. Die ausgehenden Kanten sind dann mit den möglichen Attributwerten beschriftet. Jeder Blattknoten spezifiziert einen Rückgabewert der Funktion und damit die Klasse des Datensatzes.

procedure DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*)

```
if examples is empty then
    return PLURALITY-VALUE(parent_examples)
else if all examples have the same classification then
    return the classification
else if attributes is empty then
    return PLURALITY-VALUE(examples)
else
    A ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test A
    for each value v_k of A do
        exs ← {elements of examples with A = v_k}
        subtree ← DECISION-TREE-LEARNING(exs, attributes \ A, examples)
        add a branch to tree with label v_k and subtree subtree
    end for
return tree
```

Abbildung 3.4: Funktionsweise des DECISION-TREE-LEARNING Algorithmus

Der Grundgedanke des DECISION-TREE-LEARNING Algorithmus in Abbildung 3.4 beruht auf dem „Teile–und-Beherrsche“-Prinzip (engl. *devide and conquer*). Die Reihenfolge der Attribute wird entsprechend ihrer Wichtigkeit festgelegt, so kann auch mit einfachen Heuristiken in annehmbarer Laufzeit ein geeigneter Baum gefunden werden. Beispielsweise mithilfe der Entropie wird in der Subroutine CHOOSE-ATTRIBUTE jeweils das Attribut ausgewählt, welches den größten Einfluss auf die Klassifizierung der Trainingsdaten hat. Die Entropie im Allgemeinen gibt ein Maß über die Unsicherheit einer stochastischen Zufallsvariable bzw. den mittleren Informationsgehalt eines Attributs. Ein Zugewinn an Information impliziert ein Sinken der Entropie. Für eine Zufallsvariable V mit Werten v_k und Wahrscheinlichkeiten $P(v_k)$ ist die Entropie H definiert als:

$$\mathrm{H}(V) = \sum_k P(v_k) \cdot \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \cdot \log_2 P(v_k)$$

Jeder Wert eines Attributs A teilt die Trainingsmenge in p_k positive und n_k negative Beispiele einer binären Zielvariable G mit p positiven und n negativen Datensätzen insgesamt. Der Informationsgewinn (engl. *information gain*) schließlich ist die erwartete Verringerung der Entropie bei der Wahl dieses Attributs:

$$Gain(A) = \mathrm{H}(G) - \sum_{v_k \in A} \frac{p_k + n_k}{p + n} \cdot \mathrm{H}(v_k)$$

Nach Berechnung aller Entropie-Werte wird schließlich das Attribut mit dem größten Informationsgewinn als Wurzel ausgewählt. Diese Entscheidungsregel wird getroffen, um die Tiefe des Baums so gering wie möglich zu halten. Die Subroutine PLURALITY-VALUE berechnet den häufigsten Ausgabewert einer Teilmenge (Russell & Norvig, 2009).

In manchen Fällen kann der Algorithmus komplexe Bäume erzeugen und Muster erkennen, auch wenn tatsächlich keine Regelmäßigkeiten in den Trainingsdaten enthalten sind, beispielsweise bei zufälligen Datensätzen. Dieses Problem der „Überanpassung“ wird auch *Overfitting* genannt, der Entscheidungsbaum passt sich der Struktur der gegebenen Daten also zu exakt an und kann dadurch unbekannte oder leicht veränderte Daten nur unzureichend genau klassifizieren. Auch unvollständige Datensätze oder Attribute mit kontinuierlichen Werten führen offensichtlich zu Problemen. Um dem oben genannten Problem des Overfittings entgegenzuwirken, wurden verschiedene Strategien entwickelt. Der aus der Botanik übernommene Begriff des Zurechtstutzens (engl. *pruning*) beispielsweise

umschreibt Verfahren, welche die Tiefe und Komplexität eines Entscheidungsbaums beherrschbar und verständlicher machen. Um schneller zur gewünschten Klassifizierung zu kommen, werden bereits während oder spätestens nach Erzeugung des Baums verschiedene statistische Regeln angewandt und ähnliche Zweige zusammengefasst (Mitchell, 1997).

Ein sehr erfolgreicher Einsatz von Entscheidungsbäumen findet sich auch im Konzept des *Random Forest Classifiers*. Es handelt sich dabei um ein Ensemble Learning Verfahren aus vielen einzelnen, voneinander unabhängigen Entscheidungsbäumen, deren Ergebnisse am Ende aggregiert werden. Dabei werden den einzelnen Bäumen sowohl die Trainingsdaten, als auch die Features, die zum Erzeugen des Baum verglichen werden sollen, zufällig zugeordnet (Yiu, 2012).

3.4 Deep Learning Verfahren

Neben der Verlaufsprognose unbekannter Daten ist auch die Klassifizierung zeitlich zusammenhängender Messungen ein Problem, das in vielen Bereichen der Wissenschaft Anwendung findet. Klassische Verfahren der Mustererkennung haben ihre Stärke in der Regel beim Erkennen von zeitlich unveränderlichen Strukturen. Neben der Vielzahl an Dimensionen führt jedoch vor allem der meist hohe Grad an Rauschen und die damit behaftete Unsicherheit in den vorliegenden Daten zu hoher Komplexität. Ein weiteres Problem, das mitunter eine wichtige Rolle spielt, ist die sinnvolle und effiziente Darstellung der Daten und die Auswahl der relevanten Features, um einerseits die hohe Komplexität so gering wie möglich zu halten, wichtige Charakteristiken in den Daten jedoch nicht zu verlieren. Auch im Hinblick auf den schnellen Zugriff bei großen Datenmengen muss diese Fragestellung problemabhängig und individuell diskutiert werden. Als dritte Aufgabe stellt sich die Frage, wie der Zusammenhang ähnlicher Zeitreihen formalisiert werden kann, um beispielsweise Objekte auch dann gemeinsam einordnen zu können, wenn diese mathematisch unterschiedlich sind (Esling & Agon, 2012).

Hintergrund und Definitionen

Neben vormals diskutierten klassischen Ansätzen des maschinellen Lernens haben vor allem sogenannte Deep Learning Verfahren das Feld der Zeitreihenverarbeitung revolutioniert und daher große Bekanntheit erlangt. Zunächst soll nun das Konzept der künstlichen Neuronalen Netzwerke (KNN) vorgestellt werden, bevor für die Zeitreihenklassifizierung geeignete Ansätze erläutert und diskutiert werden. Aufgrund ihrer komplexen, mehrstufigen Struktur erlauben diese Art mathematischer Modelle bereits die Verarbeitung von Rohdaten und berechnen eigenständig für Form und Struktur relevante Bestandteile, was zuvor einen hohen Grad an Domänenwissen und das Design von Features verlangte (LeCun, Bengio & Hinton, 2015).

Die ersten KNN gehen auf Rosenblatt (1957) zurück. Ein KNN ist die Zusammensetzung mehrerer hintereinander gereihter Funktionen oder Layer l_i, mit $i \in 1, \ldots, L$. Die Netzwerke sind konzipiert, um eine hierarchische Struktur der Daten zu erkennen und bestehen aus vielen kleinen Bestandteilen, den sogenannten *Perceptrons*. Die *Perceptrons* in Schicht l_i nehmen als Input das Ergebnis des vorgeschalteten Layers l_{i-1} und wenden schließlich eine nicht-lineare Aktivierungsfunktion a^i mit individuellen Parametern θ^i an, um den eigenen binären Output zu berechnen. Verbindungen bestehen immer nur zwischen *Perceptrons* benachbarter Schichten. Im Kontext von Deep Neural Networks (DNNs) werden die Parameter auch in Gewichte W und Bias b unterteilt und berechnen aus der Eingabe x aus dem vorherigen Layer die Ausgabe in der Form:

$$a^L(\theta^L, x) = a^{L-1}(\theta^{L-1}, a^{L-2}(\theta^{L-2}, \ldots, a^1(\theta^1, x)))$$

Dieser geschachtelte Prozess wird in der Literatur auch *feed-forward propagation* genannt. Während des Trainings werden die Gewichte iterativ angepasst, um die Zahl der fehlerhaften Vorhersagen in den Daten zu minimieren. Dazu wird für die Ausgabe a^L des Modells, also das Ergebnis des letzten Layers, ein Fehlerterm C, beispielsweise mithilfe des Maximum-Likelihood-Schätzers, berechnet.

$$C = \frac{1}{2n} \sum_x \left\| y(x) - a^L(\theta, x) \right\|^2$$

Dabei ist n die Zahl aller Trainingsbeispiele x. Die Parameter werden anschließend mithilfe des Gradientenabstiegsverfahren und der entsprechenden Richtungsablei-

tungen angepasst. Die Ableitung der Fehlerfunktion gibt Aufschluss darüber, wie sehr sich die Änderung der Parameter auf die Güte des Netzwerks auswirkt. Da der Fehler rückwärts durch das Netz gereicht wird, um die Parameter entsprechend ihres Einflusses auf den Fehler zu ändern, spricht man von *backpropagation* (Fawaz et al., 2019; Nielsen, 2015).

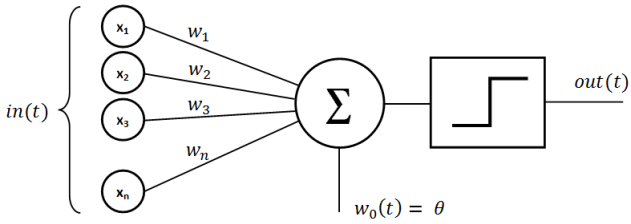


Abbildung 3.5: Aufbau eines künstlichen Neurons

Abbildung 3.5 zeigt, dass im Neuron neben der Gewichtung w_i der Eingabe $x = in(t) = (x_1, \ldots, x_n)$ noch ein zusätzlicher Bias $b = w_0$ verrechnet wird. Im Allgemeinen gilt für die Aktivierungsfunktion eines Neurons in Layer l daher $a^l = a^l(z^l) = f(W^l * x + b^l)$, wobei $*$ die komponentenweise Multiplikation definiert, auch bekannt als *Hadamard-Produkt*. Für die *backpropagation* werden dann die partiellen Ableitungen der Fehlerfunktion nach den Gewichten sowie dem Bias berechnet.

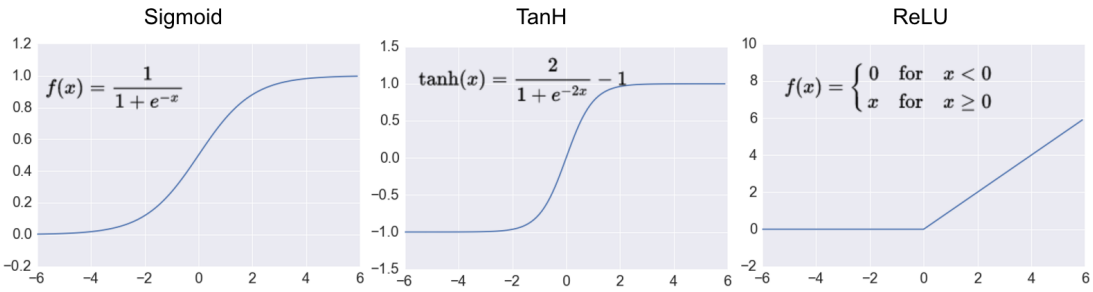


Abbildung 3.6: Verschiedene Aktivierungsfunktionen (Moujahid, 2016)

Zu den bekanntesten Aktivierungsfunktionen zählen die in Abbildung 3.6 dargestellte Logistic-Funktion σ (Sigmoid), sowie der Tangens Hyperbolicus tanh. Die nicht-linearen, monoton steigenden Funktionen führen dazu, die Ausgabe wieder in den Wertebereich zwischen 0 und 1, respektive ± 1, zu transferieren. Auf

der anderen Seite haben nahe des Ursprungs auch kleine Änderungen von θ eine große Auswirkung auf a^L, während dies an den Rändern des Definitionsbereichs kaum eine Veränderung bewirkt. Dies hilft bei der klaren Zuordnung zu einer bestimmten Klasse (Sharma, 2017).

Die wohl derzeit erfolgreichste Aktivierungsfunktion jedoch ist die Rectified Linear Unit (ReLU). Ein Problem, das beim Einsatz von Sigmoid-Funktion oder Tangens Hyperbolicus auftreten kann, ist, dass die Ableitung in höheren Schichten nahe der Eingabe und bei Werten für z^l an den Rändern des Definitionsbereichs verschwindend klein wird, was eine korrekte Anpassung der Parameter in die richtige Richtung schwer macht. Man spricht in diesem Fall vom *Vanishing Gradients Problem*. Beim Einsatz von ReLUs wird dieses Problem umgangen, wodurch sich insgesamt eine schnellere Konvergenz zeigt. Außerdem sind Kalkulation sowie Ableitung einfach und schnell zu berechnen (Ramachandran, Zoph & Le, 2017).

Recurrent Neural Networks

Einfache Künstliche Neuronale Netze (KNNs) wurden vor allem dafür entwickelt, statische Muster zu erkennen und gelangen beim Umgang mit Zeitreihen schnell an ihre Grenzen. So beziehen einzelne Einheiten kein Vorwissen über den Verlauf mit ein, sondern betrachten lediglich die Ausagbe ihrer jeweils vorgelagerten Schicht (Dorffner, 1996).

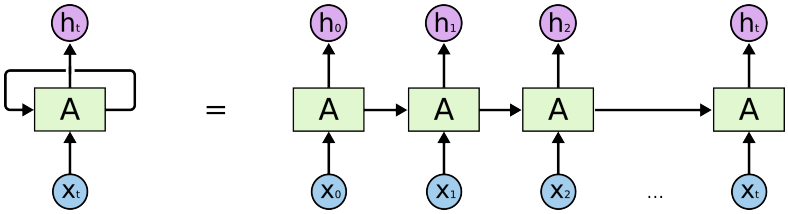


Abbildung 3.7: Ein Recurrent Neural Network (Olah, 2015)

RNN hingegen, schematisch dargestellt in Abbildung 3.7, sind dynamische Systeme, die auch den zeitlichen Verlauf der Eingabesequenz betrachten. Dies bedeutet, dass ein trainiertes Modell sowohl die aktuelle Eingabe, als auch einen internen Zustand einbezieht, um relevante Informationen zu speichern. RNN eignen sich

aufgrund der eingebauten Feedback-Loops daher sehr gut für die Klassifizierung oder Vorhersage von uni- oder multivariaten Zeitreihen (Hüsken & Stagge, 2003).

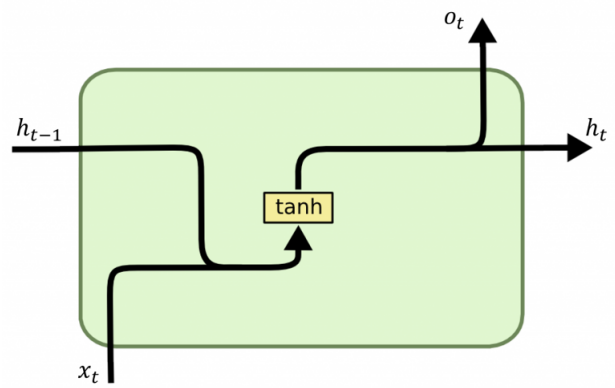


Abbildung 3.8: Ein RNN Perceptron (Lopez, 2019)

Wie Abbildung 3.8 zeigt, ändert sich an der Struktur der Neuronen im Vergleich zum KNN nur wenig. Neben Input x_t wird nun jedoch zusätzlich ein interner Zustand h_t (Hidden State) weitergereicht, der es dem Netzwerk möglich macht, auch frühere Informationen beizubehalten. Beide Vektoren werden anschließend mit einer Aktivierungsfunktion entsprechend an nachfolgende Layer weitergegeben (Olah, 2015).

Long Short-Term Memories

Bei der Anwendung von RNN zeigt sich, dass aufgrund der Struktur vor allem weit zurückliegende Ereignisse nur einen geringen Einfluss auf die Anpassung der Gewichte bei der Backpropagation haben. Da dennoch auch solche Zusammenhänge gespeichert werden sollen, führten Hochreiter und Schmidhuber (1997) das Konzept der Long Short-Term Memories (LSTMs) ein.

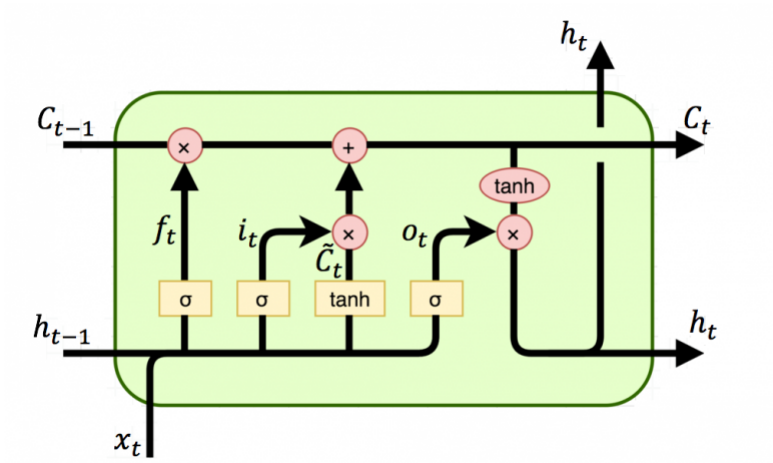


Abbildung 3.9: Ein LSTM Perceptron (Lopez, 2019)

In der komplexeren Struktur der einzelnen Zellen aus Abbildung 3.9 zeigt sich als wesentlicher Unterschied zu RNN-Perceptrons, dass neben dem internen Zustand h_t noch ein weiterer Zustand C_t eingeführt wird, der maßgeblich Einfluss auf den Output bzw. weiterzureichenden Hidden State hat. Mithilfe des Zellzustands C_t können einzelne Informationen auch gezielt lange im Netz behalten werden, um später darauf zuzugreifen. Über verschiedene Gates steuert das LSTM, welche Informationen hinzugefügt oder entfernt werden sollen. Der Aufbau der Perceptrons lässt sich in verschiedene Bestandteile aufteilen:

Über das Forget Gate f_t wird zunächst gesteuert, wie viel der früheren Informationen C_{t-1} für den neuen Zustand behalten werden sollen.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Die Sigmoid-Funktion berechnet Werte zwischen 0 und 1 und bewirkt gewissermaßen den Grad der Erhaltung der einzelnen Komponenten in C_{t-1}. Der Wert 0 bewirkt, dass der entsprechende Eintrag verworfen wird, bei 1 hingegen wird der Eintrag unverändert weitergereicht.

Im Input Gate i_t wird anschließend festgelegt, wie der Zellzustand auf Basis geeigneter Kandidaten $\tilde{C}_t$ aktualisiert wird.

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t
\end{aligned}
$$

Die Aktualisierung des Zellzustands verknüpft schließlich die einzelnen Datenflüsse unter Anwendung des *Hadamard-Produkts*, sodass sowohl relevante alte, als auch neue Informationen gesichert werden. Durch die Addition können relevante Informationen an nachfolgende Schichten auch dann weitergereicht werden, wenn sie zuvor bei der Berechnung von f_t bereits eliminiert wurden. Zuletzt wird, ähnlich wie bei RNN, der Output in Abhängigkeit von Zellzustand und Hidden State berechnet.

$$
\begin{aligned}
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Über das Output Gate o_t wird gesteuert, welche Informationen des aktuellen Zustands ausgegeben werden sollen. Dazu wird der eben kalkulierte Zellzustand zunächst mithilfe des Tangens Hyperbolicus in den Wertebereich ± 1 tranformiert und anschließend mit dem Ergebnis des Output Gates verrechnet, um zu entscheiden, welche der Informationen tatsächlich ausgegeben werden sollen. Die tatsächliche Ausgabe zum Zeitpunkt t ist äquivalent zum Hidden State h_t, welcher an die nachfolgende Einheit weitergegeben wird, während der Zellzustand C_t nach außen hin verborgen bleibt (Olah, 2015).

Gated Recurrent Units

Eine weitere Familie neuronaler Netze, welche in der Lage sind, auch länger zurückreichende Ereignisse weiter zu reichen, sind die von Cho, van Merrienboer, Bahdanau und Bengio (2014) entwickelten Gated Recurrent Units (GRU). Diese Netzwerke nutzen ähnliche Mechanismen wie LSTMs, um das *Vanishing Gradients Problem* zu lösen. Wie in Abbildung 3.10 zu sehen, werden dazu ebenfalls

verschiedene Gates verwendet, um einen internen Zustand zu kalkulieren. Insgesamt werden allerdings weniger Parameter benötigt, was sowohl Laufzeit als auch Komplexität verringert (Chung, Gülçehre, Cho & Bengio, 2014).

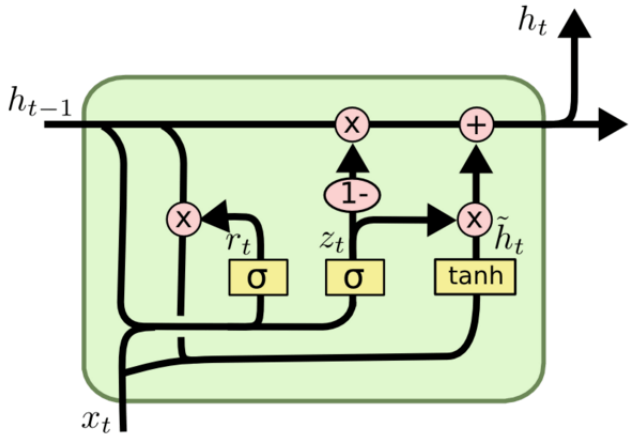


Abbildung 3.10: Ein GRU Perceptron (Lopez, 2019)

Im Update Gate z_t werden zunächst Hidden State h_{t-1} und Eingabe x_t miteinander verknüpft, um zu entscheiden, in welcher Form die vergangenen Informationen verändert werden sollen. Im Gegensatz zum LSTM hat ein Perceptron hier keinen Einfluss auf den Grad der Anpassung, da nur ein Zustand zum Speichern aller nötigen Informationen zur Verfügung steht. Informationen, die nicht weiter benötigt werden, können über das Reset Gate gelöscht werden. Die Formel ist äquivalent zum Update Gate, allerdings mit unterschiedlichen Gewichten.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Ein Unterschied ergibt sich vor allem in der Verwendung der berechneten Werte und damit dem Einfluss auf das Gesamtergebnis. Geeignete Kandidaten $\tilde{h}_t$ werden ähnlich zu RNN berechnet. Unter Verwendung des Resets Gates wird auf diesem Weg sichergestellt, dass relevante Informationen weitergereicht werden. Gehen an einer Stelle die Einträge von r_t gegen 0, so bewirkt das *Hadamard Produkt* ein Vergessen alter Informationen – der aktuelle Zustand x_t wird unverändert gespeichert.

Im letzten Schritt wird der neue Hidden State berechnet. Dazu wird mithilfe des Update Gates eine lineare Interpolation des Hidden States und der zuvor erzeugten Kandidation berechnet (Kostadinov, 2017).

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU und LSTMs teilen etliche Gemeinsamkeiten. Beide Units addieren den internen Zustand in jedem Zeitschritt anhand relevanter Informationen und neuem Input, was eine Reihe von Vorteilen bringt. Zum einen können auch weit zurückliegende Features gespeichert werden. Beide Gates können relevante Informationen verändern, ohne die Daten unwiderruflich zu überschreiben. Auf der anderen Seite wird bei der Backpropagation im Vergleich zur Multiplikation das *Vanishing Gradients Problem* umgangen.

Dennoch ergeben sich auch eine Reihe von Unterschieden zwischen beiden Units. Offensichtlich zu sehen ist, dass bei GRU eine Parameter Matrix und ein Bias Vektor weniger benötigt werden, da diese nur einen internen Zustand beinhalten. Bei LSTMs kann jedoch die Ausgabe mithilfe des Output Gates deutlich granularer gesteuert werden, während GRU ihren Zustand unverändert weiterreichen. Auch die Stelle des Input Gates bzw. des korrespondierenden Reset Gates bewirkt unterschiedliches Verhalten. Während bei LSTMs der Einfluss neuer Informationen unabhängig vom aktuellen Zellzustand agiert, sind die Berechnung geeigneter Kandidaten und die Aktualisierung des Hidden State bei GRU stark voneinander abhängig (Chung et al., 2014).

Insgesamt lässt sich im Allgemeinen nicht sagen, welcher Typ bessere Ergebnisse liefert. Auch wenn verschiedene empirische Experimente zeigten, dass beide Units in vielen Fällen ein ähnliches Verhalten zeigen, ist weiterhin unklar, ob dies auch auf beliebige von den Experimenten abweichende Probleme angewendet werden kann (Bahdanau, Cho & Bengio, 2014). Im Rahmen dieser Arbeit sollen aus diesem Grund ebenso beide Ansätze verfolgt werden.

3.5 Luftqualität in geschlossenen Räumen

Wie oben bereits beschrieben ist Luftverunreinigung ein globales Problem, das sich wesentlich auf die Gesundheit vieler Menschen, vor allem in urbanen Räumen, auswirkt. In vielen Städten weltweit werden Konzentrationen und Werte verschiedener Schadstoffe gemessen, um die aktuelle Belastung auszuwerten. Regierungen und Behörden weltweit stellen Echtzeitdaten zur Luftqualität öffentlich zugänglich bereit. Die aktuelle Forschung geht allerdings davon aus, dass die gemessenen absoluten Werte nicht grundsätzlich miteinander verglichen werden können und ggf. zu falschen Annahmen führen (Saad et al., 2017).

Um Aussagen über die Luftqualität besser vergleichbar zu machen, wurden daher verschiedene Indizes entwickelt, welche absolute Schadstoffwerte verrechnen und in eine einfache numerische oder farbliche Skala transformieren.

1999 veröffentlichte die U.S. Enironmental Protection Agency (EPA) einen Luftqualitätsindex (engl. *Air Quality Index (AQI)*), um eine Aussage über die tägliche Schadstoffbelastung in Großstädten treffen zu können.

Breakpoints						**AQI**	**Kategorie**
O_3 (ppm)	PM_{10} (μgm^3)	$PM_{2.5}$ (μgm^3)	CO (ppm)	SO_2 (ppm)	NO_2 (ppm)		
0-0.064	0-54	0-15.4	0-4.4	0-0.034	–	0-50	Gut
0.065-0.084	55-154	15.5-40.4	4.5-9.4	0.035-0.144	–	51-100	Moderat
0.085-0.104	155-254	40.5-65.4	9.5-12.4	0.145-0.224	–	101-150	Teilweise ungesund
0.105-0.124	255-354	65.5-150.4	12.5-15.4	0.225-0.304	–	151-200	Ungesund
0.125-0.374	355-424	150.5-250.4	15.5-30.4	0.305-0.604	0.65-1.24	201-300	Sehr ungesund
0.375-0.504	425-504	250.5-350.4	30.5-40.4	0.605-0.804	1.25-1.64	301-400	gefährlich
0.505-0.604	505-604	350.5-500.4	40.5-50.4	0.805-1.004	1.65-2.04	401-500	gefährlich

Abbildung 3.11: Breakpoint Tabelle der EPA zur Berechnung des AQI

Die Berechnung des AQI basiert auf Messungen der fünf relevanten Schadstoffe Ozon (O_3), Feinstaub (PM), Kohlenstoffmonoxid (CO), Schwefeldioxid (SO_2)

und Stickstoffdioxid (NO_2). Die gemessenen Werte werden über einen definierten Zeitraum bis zu acht Stunden gemittelt. Jeder Mittelwert wird anschließend in einen individuellen Index I_p für Schadstoff p auf Basis einer Breakpoint Tabelle (siehe dazu Tabelle 3.11) umgerechnet. Die Werte wurden von der EPA auf Basis einer epidemiologischen Studie festgelegt. Für NO_2 wurden keine Standards für kleine Belastungen definiert, es kann daher nur Werte über 200 annehmen. Die einzelnen Indizes werden mithilfe folgender linearen Interpolation berechnet:

$$I_p = \frac{I_{HI} - I_{LO}}{BP_{HI} - BP_{LO}} \times (C_p - BP_{LO}) + I_{LO}$$

Dabei ist C_p die gemessene Konzentration des Schadstoffs p, BP_{HI} die obere Grenze bzw. BP_{LO} die untere Grenze des Bereichs der Breakpoint Tabelle in den die gemessene Konzentration fällt. I_{HI} und I_{LO} sind die zu BP_{HI} bzw. BP_{LO} korrespondierenden AQI Werte der Tabelle. Der größte individuelle Index I_p schließlich definiert den aktuellen AQI des Orts. Der Index kann Werte zwischen 0 und 500 annehmen, wobei höhere Werte auch eine höhere Belastung anzeigen (Kanchan, Gorai & Goyal, 2015).

Die beschriebene Methode ist recht starr und ignoriert mögliche Aggregationen verschiedener oder neuer Schadstoffe. 2007 wurde daher ein erweiterter AQI vorgestellt, der ebenso die fünf genannten Werte verrechnet:

$$I = \left[\sum_{i=1}^{n} (\mathrm{AQI}_i)^p \right]^{\frac{1}{p}}$$

Die Formel beinhaltet dabei den Index AQI_i für Schadstoff i gemäß oben stehender Formel sowie eine Konstante $p \in [1, \infty)$. Für $p = 1$ stimmt der Wert mit der Summe der einzelnen Indizes überein. Für $p \to \infty$, deckt sich der Index mit dem Maximum der einzelnen Werte, äquivalent zur ursprünglichen Methode der EPA. Die richtige Wahl von p wird in der Literatur weiterhin diskutiert, Werte größer als 2 werden allerdings zumeist nicht angenommen (Kyrkilis, Chaloulakou & Kassomenos, 2007; Ruggieri & Plaia, 2012).

Auch wenn der EPA AQI weitgehend anerkannt ist, müssen jedoch andere Parameter verrechnet werden, um Aussagen über die Luftqualität in geschlossenen Räumen treffen zu können. Dazu zählen neben bekannten Schadstoffen wie Koh-

lenstoffdioxid (CO_2) oder flüchtigen organischen Verbindungen (VOC) vor allem auch die vorherrschende Temperatur und Luftfeuchtigkeit. Für die Belastung in geschlossenen Räumen muss daher auch die Breakpoint Tabelle entsprechend erweitert werden. Wie in Tabelle 3.12 dargestellt, werden zur Quantifizierung der Luftqualität schließlich vier Bereiche definiert. Während beim AQI große Werte für eine hohe Schadstoffbelastung stehen, soll der Indoor Air Quality Index (IAQI) die Güte der Luft widerspiegeln, weswegen niedrige Werte mit schlechter Luft einhergehen. Während beim EPA Index vor allem lange Zeiträume gemessen werden, werden beim IAQI lediglich kurze Zeiträume bis hin zu einer Minute gemessen, da sich die Luftqualität in Räumen deutlich schneller ändern kann (Saad et al., 2017).

Breakpoints				**IAQI/TCI**	**Kategorie**
CO_2 (ppm)	VOC (ppm)	$Temp$ (℃)	$Humi$ (%)		
340-600	0.000-0.087	20.0-26.0	40.0-70.0	100-76	Gut
601-1000	0.088-0.261	26.1-29.0	70.1-80.0	75-51	Moderat
1001-1500	0.262-0.43	29.1-39.0	80.1-90.0	50-26	Ungesund
1501-5000	0.44-3.00	39.1-45.0	90.1-100.0	25-0	Gefährlich

Abbildung 3.12: Breakpoint Tabelle zur Berechnung des IAQI

Neben einzelnen Bestandteilen der Luft werden beim IAQI auch Temperatur und Luftfeuchtigkeit verrechnet, da diese einen wesentlichen Einfluss auf das komfortable Empfinden (engl. Thermal Comfort Index (TCI)) der Luftqualität haben. Die definierten Bereiche können individuell, aber auch von Region zu Region, stark schwanken. Oben stehende Tabelle stammt beispielsweise aus Malaysia, wo deutlich höhere Temperaturen als noch angenehm empfunden werden, als in Mitteleuropa. Ziel der Messungen ist es daher, die Bereiche bestmöglich anzupassen, um in Zukunft eine verlässliche Aussage über die Luftqualität treffen zu können.

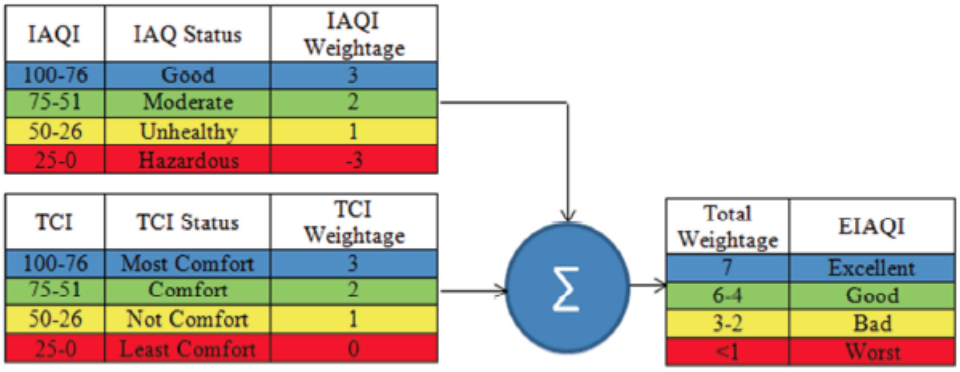


Abbildung 3.13: Berechnung des Indoor Air Quality Index (Saad et al., 2017)

Da sich Schadstoffe und thermische Einflüsse unterschiedlich auf das Empfinden der Luftqualität auswirken, werden zunächst jeweils eigene Indizes berechnet. Für die Klassifizierung der Gesamtqualität werden die einzelnen Werte entsprechend gewichtet und schließlich summert, wie Abbildung 3.13 zeigt (Saad et al., 2017).

4 Auswertung und Ergebnisse

Im folgenden Kapitel werden die trainierten Modelle beschrieben und verglichen. Der Fokus soll zum einen auf der Struktur der Modelle liegen, zum anderen werden verschiedene Ansätze miteinander verglichen und auf ihre Einsetzbarkeit untersucht. Die Skripte zur Berechnung der Modelle laufen auf einem ThinkPad T450s mit 2,6 GHz Intel Core™ i7 Prozessor (4 Kerne) und 8 GB RAM. Auf der Maschine ist Ubuntu 18.04 installiert. Als Programmiersprache wird Python in der Version 3.6.8 verwendet.

Die gesammelten Trainingsdaten werden in verschiedene Datensätze separiert. Vor dem Einbau der Klimaanlage wurden an 16 Werktagen Daten von zwei Räumen gemessen. In diesem Zeitraum fanden 36 Veranstaltungen statt, was einem Anteil von etwa 21 Prozent aller Daten entspricht. Im Zeitraum nach dem Einbau stehen für das Training Daten von 20 Werktagen mit insgesamt 126 Veranstaltungen zur Verfügung. Die Datensätze werden zufällig in 60% Trainingsmenge und jeweils 20% Validierungs- sowie Testmenge aufgeteilt. Die Daten werden sensorweise standardisiert, mit einem Mittelwert von 0 und einer Standardabweichung von 1, an die Algorithmen übergeben.

4.1 Klassische Modelle

Wie in Abschnitt 3.3 bereits beschrieben, müssen bei klassischen Modellen im Vergleich zu neuronalen Netzen zunächst definierte Features berechnet werden. Zur automatisierten Berechnung der Features wird das Python Paket *tsfresh* verwendet (Christ, Braun, Neuffer & Kempa-Liehr, 2018).

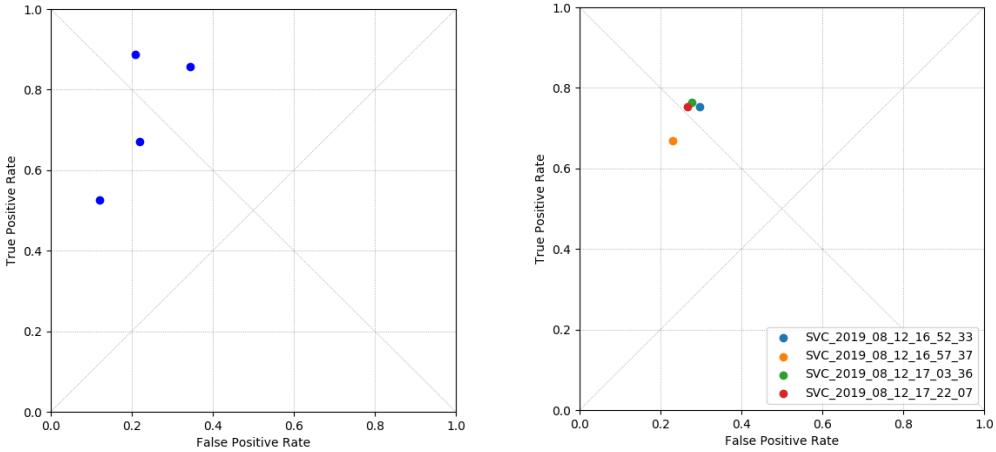
Dazu wird die Klasse `EfficientFCParameters`[1] ohne die Keys `change_quantiles` und `fft_coefficient` genutzt, da diese die Laufzeit wesentlich erhöhen. Bei Ersterem werden Veränderungen auf durch Quantile definierten Abschnitten berechnet, beim Zweiten die Koeffizienten der eindimensionalen Fourier-Transformation. Die einzelnen Features werden für Fenster der Größe $w_s \in \{12, 20, 40\}$ mit einer Überlappung von 20% bzw. $w_s \in \{60, 120\}$ mit einer Überlappung von 80% berechnet. Dies entspricht Intervallen von 60, 100, 200, 300 und 600 Sekunden. Für den betrachteten Zeitraum werden zunächst 288 verschiedene Features pro Sensor berechnet, um im Nachhinein die Anzahl $k_n \in [10, 80]$ der Werte, welche die Zeitreihe am besten beschreiben, einzuschränken. Die Auswahl der entsprechenden Features basiert auf den jeweiligen Verteilungen und wird mithilfe einer Varianzanalyse und unter Zuhilfenahme der Klasse `SelectKBest` aus dem *Scikit-Learn* Paket getroffen. Auch für Random Forests und Support Vektor Maschinen werden die Implementierungen des bekannten Python-Moduls *Scikit-Learn*, Version 0.21.2, eingebunden (Grisel et al., 2019). Die Auswahl der genannten Hyperparameter wird über die Bibliothek *Scikit-Optimize* gesteuert. Es handelt sich dabei um eine einfache und effiziente Bibliothek, um komplexe Funktionen anhand der gegebenen Parameter zu optimieren. Dazu wird das Ergebnis der Funktion mithilfe eines Gauss-Prozesses geschätzt, iterativ verbessert und anschließend validiert. Auf diese Weise können die Hyperparameter effizienter und schneller optimiert werden (Head et al., 2018).

Support Vektor Maschinen

Bei der Anwendung klassischer Machine Learning Verfahren gibt es eine Menge an Hyperparametern, die maßgeblich beeinflussen, wie schnell und gut sich das Modell den Trainingsdaten anpassen kann. Für das Training der Support Vektor Maschinen werden daher neben Fenstergröße und Featureanzahl zusätzlich das Gewicht des Fehlerterms $\mu \in [0.7, 1.0]$, die Toleranz hinsichtlich des Abbruchkriterium $t_s \in [10^{-6}, 10^{-2}]$ sowie die maximale Anzahl an Iterationen $i_{max} \in [500, 20000]$ in verschiedenen Konstellationen getestet. Zusätzlich werden die verschieden Kerne $kernel \in \{poly, rbf, sigmoid\}$ angewandt. Über den Kern wird die Kovarianz

[1] Eine vollständige Auflistung der einzelnen Features kann der Online-Dokumentation entnommen werden (Christ, Braun, Neuffer & Kempa-Liehr, 2019).

bzw. der Zusammenhang der einzelnen Features untereinander beeinflusst. Er gibt daher ein zusätzliches Maß für die Gewichtung der Features.



(a) Support Vector Maschinen trainiert mit *Round Table*

(b) Ergebnisse für Testset

Abbildung 4.1: Receiver Operating Characteristic für trainierte SVM

Es zeigt sich, dass mit einfachen SVM bereits sehr gute Ergebnisse erzielt werden können. Wie in Abbildung 4.1 (a) zu sehen, erzielen die einzelnen Modelle jedoch sehr unterschiedliche Werte. Allgemein lässt sich sagen, dass die trainierten Modelle zumeist eine der beiden Klassen zuverlässiger prognostizieren. Es konnte eine Specificity (TNR) von 80% erzielt werden, bei einer Sensitivity (TPR) von 89% mit einer Accuracy von 75%. Die Übertragung auf unbekannte Testdaten zeigte mit einer Specificity von 75% bei einer Accuracy von 74% (`SVC_2019_08_12_17_22_07`, rot) ähnliche und ebenfalls zufriedenstellende Ergebnisse. Die erreichte Sensitivity liegt hier bei 73%. Generell kann das Modell den Raum also zuverlässiger „frei“ als „belegt“ klassifizieren. Die Ergebnisse der anderen Modelle können Abbildung 4.1 (b) entnommen werden.

Name	w_s	k_n	μ	t_s	i_{max}	*kernel*
SVC_2019_08_12_16_52_33	12	25	0.9	0.001	3000	*rbf*
SVC_2019_08_12_16_57_37	40	63	0.75	0.004	2040	*rbf*
SVC_2019_08_12_17_03_36	40	61	0.72	0.004	2757	*rbf*
SVC_2019_08_12_17_22_07	40	72	0.77	0.005	2209	*rbf*

Abbildung 4.2: Hyperparameter der trainierten SVM

Tabelle 4.2 zeigt die jeweiligen Hyperparameter der ausgewählten, trainierten SVM. Auffällig ist, dass die Fenstergröße $w_s = 40$ (200 Sekunden) ausreichend scheint, um zuverlässige Aussagen treffen zu können. Auch Penalty-Term μ und Toleranz t_s zeigen jeweils kaum Schwankungen. Außerdem nutzt jedes der Modelle den Radial basis function (RBF) Kernel, der für zwei Vektoren x und x' definiert ist als:

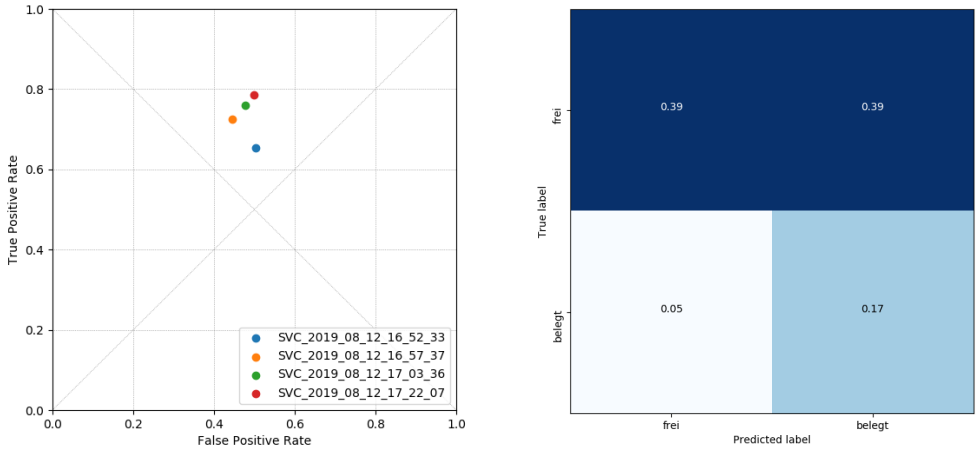
$$K(x, x') = exp\left(\frac{\|x - x'\|^2}{2\sigma^2}\right),$$

wobei durch $\|x - x'\|^2$ die Euklidische Distanz der Vektoren definiert ist und σ ein frei wählbarer Parameter (Vert, Tsuda & Schölkopf, 2004).

Es zeigt sich, dass die Anzahl der Feature einen wesentlichen Einfluss auf die Genauigkeit der SVM hat. Je mehr Feature betrachtet werden, desto besser passt sich das Modell den Trainingsdaten an. Andererseits werden gleichbedeutend Vorhersagen für unbekannte Daten schlechter, was ein deutliches Anzeichen für *Overfitting* ist. So ergibt sich ein Trade-Off zwischen der Anzahl der Feature und damit der Zuverlässigkeit des Modells und der Übertragbarkeit auf neue Daten. Mehr als 70 Features scheinen im Allgemeinen nicht sinnvoll, um den Grad an *Overfitting* so gering wie möglich zu halten.

Unter den gewählten Features befinden sich fast ausschließlich Kennzahlen über den CO_2-Gehalt. Zu den wichtigsten Features zählen unter anderem Median, Mittelwert und die Quantile der Verteilung der Daten. Außerdem diverse Verhältnisse im betrachteten Fenster wiederkehrender Punkte, die Koeffizienten verschiedener Wavelet-Transformationen sowie lineare Trends. Daneben befinden sich lediglich einige wenige Berechnungen für die Luftfeuchtigkeit. Temperatur, Druck und Gas-

widerstand werden beim Training außer Acht gelassen. Da die Daten aufgrund des RBF Kerns in eine andere Dimension transformiert werden, lässt sich über die Gewichtung der SVM der einzelnen Features an dieser Stelle keine Auskunft treffen.



(a) ROC für Support Vector Maschinen getestet mit *Europa*

(b) Confusion Matrix für Testset

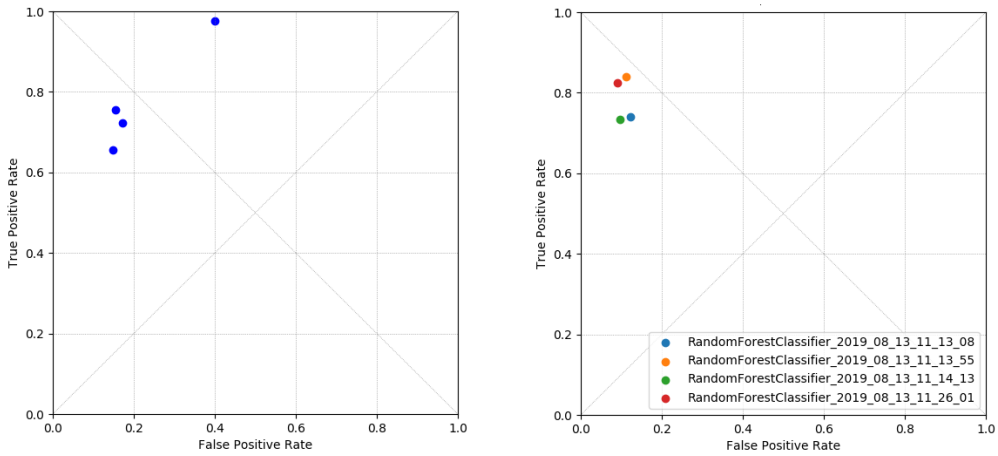
Abbildung 4.3: SVM Vorhersage für separaten (unbekannten) Raum

Die oben erläuterten Modelle können nicht auf einen anderen Raum übertragen werden. Wie in Abbildung 4.3 (a) zu sehen, erzielen die Modelle für den Belegungsstatus von *Europa* zwar eine gute Specificity bis knapp unter 80%, können aber nicht erkennen, wenn der Raum „frei“ ist. Dieses Ergebnis verdeutlicht auch die Confusion Matrix in Abbildung 4.3 (b). In der Realität ist dieses Ergebnis allerdings mit Vorsicht zu bewerten, da die vorliegenden Daten nur unzureichend genau sind. Im genannten Raum finden häufig kurze Besprechungen statt, die keinen Eintrag im Kalender haben. Das Modell klassifiziert das Fenster dann korrekt als „belegt“, obwohl dieses in der Statistik als False negative gezählt wird.

Zusammenfassend lässt sich sagen, dass mit SVM bereits auf einfachem Weg gute Ergebnisse erzielt werden können. Die trainierten Modelle scheinen gegenüber kleinen Veränderungen in den Daten robust zu sein und können neue Daten daher gut klassifizieren. Grundsätzlich neigen sie zu *Overfitting*, weswegen vor allem durch Auswahl und Berechnung geeigneter Feature noch mehr optimiert werden kann.

Random Forest Klassifizierung

Für das Training der Random Forests werden eine Reihe zusätzlicher Hyperparameter festgelegt. In den Suchraum werden neben Fenstergröße und betrachteten Features auch die Anzahl der Bäume $n_e \in [10, 80]$, die maximale Tiefe $d_{max} \in [10, 100]$, die untere Grenze der Anzahl korrekter Datensätze pro Blatt $s_{min} \in [3, 20]$ sowie der maximale Anteil an zu betrachtenden Features für einen Split $f_{max} \in [0.1, 0.5]$ aufgenommen.



(a) Random Forests trainiert mit *Round Table*

(b) Ergebnisse für Testset

Abbildung 4.4: Receiver Operating Characteristic für Random Forests

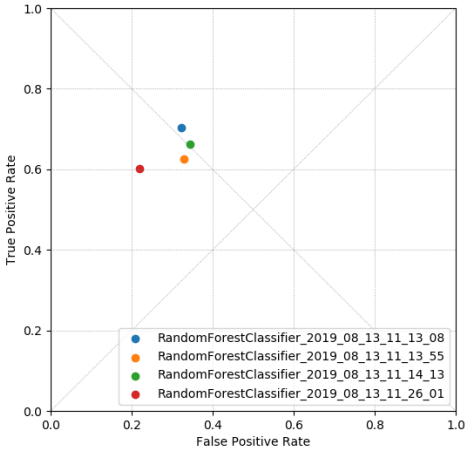
Es zeigt sich, dass die einzelnen Classifier sehr ähnliche Ergebnisse erzielen. Wie in Abbildung 4.4 (a) zu sehen, erreichen auf den Trainingsdaten alle Modelle etwa 85% Specificity und wenigstens 75% Sensitivity. Dies ist einhergehend mit einer Accuracy von 82%. Die Klassifizierung von „frei“ ist hier also etwas genauer als die von „belegt“. Die Testdaten können in diesem Fall sogar noch besser vorhergesagt werden, wie Abbildung 4.4 (b) zeigt.

So erzielt das Modell `RandomForestClassifier_2019_08_13_11_26_01` eine Sensitivity von 83% und eine Specificity von 91% auf den Testdaten. Dies entspricht einer Accuracy von 88%.

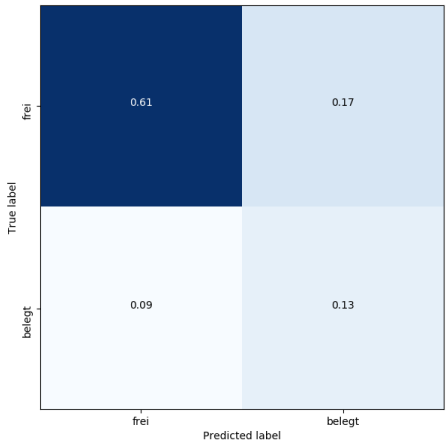
Name	w_s	k_n	f_{max}	s_{min}	d_{max}	n_e
RandomForestClassifier_2019_08_13_11_13_08	40	50	0.4	20	60	60
RandomForestClassifier_2019_08_13_11_13_55	120	51	0.2	18	10	27
RandomForestClassifier_2019_08_13_11_14_13	40	78	0.4	15	39	51
RandomForestClassifier_2019_08_13_11_26_01	120	61	0.5	20	33	50

Abbildung 4.5: Hypterparameter der trainierten Random Forests

Wie oben bereits erwähnt, erzielen alle Modelle ähnliche Ergebnisse. Dies bekräftigt auch die Auswahl der Hyperparameter in Tabelle 4.5. In den gewählten Parametern lassen sich keine Muster erkennen. Pro Split scheinen etwas weniger als die Hälfte der Feature benötigt zu werden, ein Ensemble aus maximal 50 einzelnen Bäumen mit einer Tiefe von höchstens 40 ist ausreichend. Wie bereits bei Support Vector Maschinen reicht eine Teilmenge von etwa 60 Features für eine korrekte Klassifizierung. Auch bei Random Forests werden nahezu ausschließlich Kennzahlen des CO_2-Gehalts betrachtet. Die drei herausragenden Features sind die Häufigkeit der einzelnen CO_2-Werte im betrachteten Fenster, der Anteil verschiedener CO_2-Werte und das Temperatur-Minimum. Ebenfalls wichtig ist die Position des CO_2-Maximums sowie die letzte Position des Minimums. Da der CO_2-Gehalt stetig steigt, wenn sich Personen im Raum befinden und konstant bleibt bzw. fällt, wenn nicht, erscheint diese Gewichtung sinnvoll.



(a) ROC für Random Forests getestet mit *Europa*



(b) Confusion Matrix für Testset

Abbildung 4.6: Random Forest Vorhersage für separaten (unbekannten) Raum

Die Vorhersage unbekannter Daten liefert bei Random Forests bessere Ergebnisse als die oben beschriebenen Support Vector Maschinen. Das Modell `RandomForestClassifier_2019_08_13_11_26_01` (rot) erzielt neben einer Specificity von 78% immerhin eine Sensitivity von 60%, wie Abbildung 4.6 zu entnehmen ist. Auch hier gilt, dass die Testdaten nur unzureichend genau sind. Es finden sich sowohl Fenster, in denen der Raum trotz Kalendereintrag offensichtlich „frei“ zu sein scheint und andersherum. Außerdem sind die Ergebnisse in Anbetracht dessen, dass das Modell die Daten noch nie gesehen hat, zufriedenstellend, da deutlich besser als zufällig.

Zusammenfassung und Ausblick

Abschließend lässt sich sagen, dass mit einfachen Machine Learning Verfahren bereits gute Ergebnisse erzielt werden. Die Random Forest Modelle erreichen insgesamt zuverlässigere Vorhersagen als Support Vector Maschinen. Aufgrund der Unsicherheit der Trainingsdaten liegt die tatsächliche Trefferrate vermutlich sogar etwas höher. Um die Ergebnisse weiter zu verbessern müssen, die Daten weiter aufbereitet und die vorliegende Klassifizierung verifiziert werden, beispielsweise durch den Einbau weiterer Bewegungssensoren. Insgesamt zeigt sich, dass vor

allem der CO_2-Gehalt die entscheidende Komponente ist. Über geeignetes Engineering können weitere, relevante Features herausgearbeitet werden, um einerseits die Anzahl der zu berechnenden Feature zu verringern und andererseits die Genauigkeit der Algorithmen trotzdem zu erhöhen. Zusätzlich können Rauschen in den Daten und Messungenauigkeiten der Sensoren über entsprechende Transformationen bereinigt werden. Das Preprocessing der Daten wird in dieser Arbeit bewusst knapp gehalten, um zu Beginn einem möglichen Informationsverlust vorzubeugen.

Als wesentlicher Nachteil zeigt sich, dass klassische Verfahren nur diskrete Klassifizierungen treffen können und die Berechnung von geeigneten Features aufwändig ist. Da die Fensterbreite bewusst klein gehalten wurde, schwankt die Vorhersage oftmals von Fenster zu Fenster. Andererseits zeigen die trainierten Modelle insgesamt recht robuste Ergebnisse, das heißt, auch externe Einflüsse auf das Raumklima, wie eine Klimaanlage, können erfolgreich verarbeitet werden.

4.2 Neuronale Netze

Um die im vorherigen Abschnitt erläuterten Probleme und Nachteile zu umgehen und genauere Ergebnisse zu erzielen, werden zusätzlich neuronale Netze trainiert. Für die Anwendung von neuronalen Netzen wird die bekannte Python Open Source Deep Learning Bibliothek *Keras* (Chollet et al., 2015), Version 2.2.4, mit dem *TensorFlow* Backend (Abadi et al., 2015), Version 1.13.1, verwendet. Zunächst werden die verschiedenen Netzwerkstrukturen miteinander verglichen. Anschließend wird erläutert, wie sich die Netze auf unbekannte Daten übertragen lassen und wie wichtig die einzelnen Sensoren dabei sind. Für den Vergleich der einzelnen Netze werden die in Kapitel 1.1 beschriebenen Anforderungen sowie die definierten Zielfunktionen berechnet. Neben der Receiver Operating Characteristic werden vor allem Sensitivity und Accuracy zum Vergleich der einzelnen Modelle herangezogen.

Wahl der Hyperparameter und Netzwerkstruktur

Wie bereits bei klassischen Machine Learning Verfahren ist auch die Performance von Neuronalen Netzen maßgeblich von der Wahl der Hyperparameter abhängig. Neuronale Netze optimal zu trainieren ist von einer Vielzahl an Umständen abhängig. Die Fehlerfunktion ist offensichtlich nicht konvex und enthält daher eine Vielzahl lokaler Minima. Zunächst stellt sich daher die Frage, welche Startwerte für die Gewichte gewählt werden sollten, um mögliche lokale Optima zu vermeiden. Für Gewichte in der Nähe von Null ist die Sigmoid-Aktivierungsfunktion nahezu linear und die Ableitung daher konstant. Erst im späteren Verlauf des Trainingsprozesses vergrößern sich die Gewichte, um relevante Muster in den Daten hervorzuheben. Werden die Gewichte initial zu groß gewählt, führt dies häufig zu schlechten Ergebnissen (Hastie et al., 2009, S. 397-401). Durch sogenannte Grid Search werden unterschiedliche Konstellationen von definierten Parametern angewandt und anschließend anhand ihrer Ergebnisse miteinander verglichen.

Die neuronalen Netze nutzen jeweils die binäre Kreuzentropie (engl. *binary cross-entropy*) zwischen den beiden Klassen „frei“ und „belegt“, um ihre Parameter zu optimieren. Um den Lernprozess der einzelnen Modelle zu stabilisieren und Overfitting zu vermeiden, werden Regularisierung und Dropout verwendet, zwei der derzeit wirksamsten Methoden (Hinton, 2012). Bei Regularisierung wird der Loss-Funktion ein zusätzlicher Fehlerterm hinzugefügt, um den Einfluss der temporär berechneten Ergebnisse zu steuern und mögliche lokale Optima zu übergehen. Für die Modelle wird die sogenannte L2-Regularisierung ergänzt, bei der das Skalarprodukt der Parameter in der Form $\lambda \|\theta\|_2^2$ als zusätzlicher Summand der Kostenfunktion verrechnet wird (Azizi et al., 2018). Die Gewichtung λ des Fehlerterms wird als Hyperparameter in die Grid Search mit einbezogen. Dropout sorgt vor allem für die Vermeidung von starken Abhängigkeiten aufeinanderfolgender Units. Durch die zufällige Abschaltung einzelner Perceptrons mit Wahrscheinlichkeit p in jedem Schritt werden die Parameter dahingehend angepasst, markante Strukturen in den Daten zu beachten und sich nicht von den vorgeschalteten Berechnungen beeinflussen zu lassen (M. L. Peng, 2018).

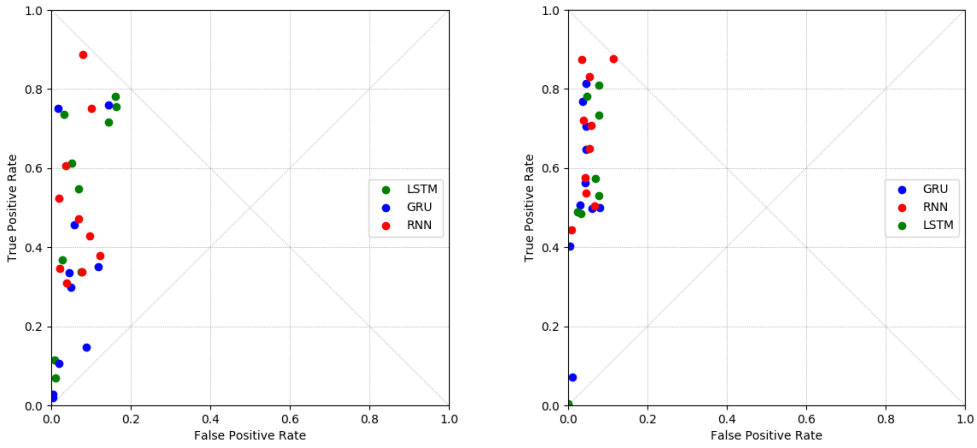
Für Grid Search werden verschiedene Konstellationen der Hyperparameter getestet. So werden die Zahl der Hidden Layer $n_h \in \{1, 2\}$, die Größe der Batches $b_s \in \{32, 64, 128\}$, sowie die Fenstergröße $w_s \in \{4, 6, 12\}$ verändert. Dies ent-

spricht einem Zeitraum von 20, 30 und 60 Sekunden. Pro Layer werden $2w_s$ Units u erzeugt. Zur Minimierung der Loss-Funktion werden die bekannten Algorithmen *Stochastic Gradient Descent* (SGD), *Root Mean Square Propagation* (RMSprop) und *Adaptive Moment Estimation* (Adam) verwendet. Zusätzlich werden verschiedene Dropout-Wahrscheinlichkeiten $d_r \in \{0, 0.1, 0.2\}$ und L2-Regularisierungsgewichte $l_{reg} \in \{0.1 \times 10^{-3}, 0.2 \times 10^{-3}\}$ evaluiert. Dies führt zu insgesamt 324 verschiedenen Kombinationen aus Hyper-Parametern pro Netzwerk $n \in \{\text{RNN}, \text{LSTM}, \text{GRU}\}$.

Alle Modelle werden mit der gleichen Anzahl an Iterationen trainiert und nach 100 Epochen gestoppt. Sobald nach zehn Schritten keine Verbesserung der Validation Accuracy mehr festzustellen ist, wird die Learning Rate außerdem mit dem Faktor 0.9 reduziert. Für die Feinkalibrierung und die Suche nach weiteren Hyperparametern wird außerdem das Python-Paket *Scikit-Optimize* genutzt (Azizi et al., 2018).

Vergleich der einzelnen Architekturen

Zunächst werden die verschiedenen Netzwerke RNN, LSTM und GRU anhand ihrer Accuracy miteinander verglichen. Der Graph in Abbildung 4.7 (a) zeigt die Performance der jeweils zehn besten Netze, trainiert anhand der Daten von *Round Table* und mithilfe von Grid Search, und bezieht sich auf eine unabhängige Testmenge. Im Mittel konnten dabei 86% Accuracy erreicht werden (mit einem Median von 86%) und sogar 89% Accuracy (Median 88%) auf den Trainingsdaten. Abbildung 4.7 (a) zeigt deutlich, dass die einzelnen Architekturen sich nicht voneinander unterscheiden lassen, alle drei Implementierungen erreichen ähnliche Ergebnisse. Zu erkennen ist außerdem die geringe False Positive Rate bzw. dazu äquivalent eine hohe True Negative Rate. Alle Netze können eine Raumbelegung also sehr zuverlässig vorhersagen.



(a) Deep Neural Networks trainiert mit *Round Table*

(b) Deep Neural Networks trainiert mit *Speaker's Corner*

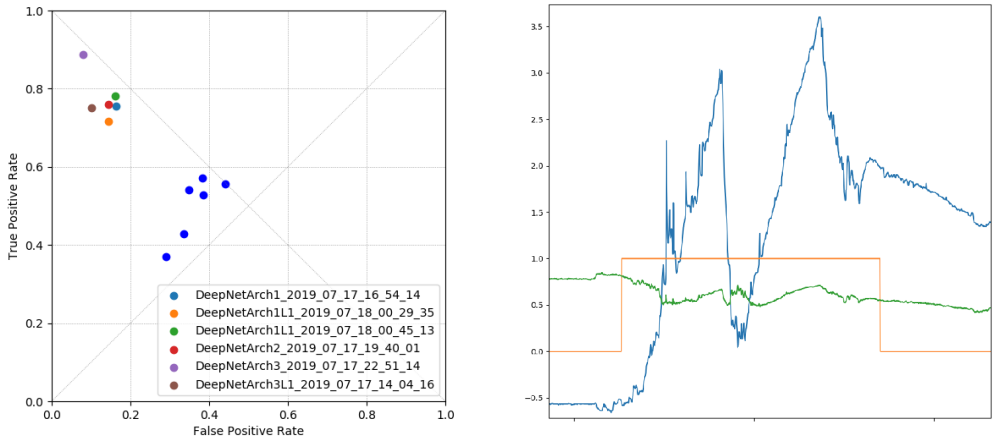
Abbildung 4.7: Receiver Operating Characteristic für verschiedene Deep Neural Networks und Räume

Fehler bei der Vorhersage ergeben sich fast ausschließlich, wenn sich keine Personen im Raum befinden. Die True Positive Rate ist beim Großteil der Netze von *Round Table* sogar unter 50%, also geringer als zufälliges Raten. Für Grid Search ist dies allerdings nicht verwunderlich, da unvermeidbar auch schlechte Konstellationen aus Hyperparametern getestet werden. Dennoch zeigt sich, dass viele Netze dazu tendieren, bereits bei geringen Schwankungen falsche Vorhersagen zu treffen. Daher ergibt sich zunächst die Annahme, dass die Trainingsdaten unzureichend sind. Da bereits Daten ab 6 Uhr frühs und bis 22 Uhr abends vorliegen, existieren viele freie Abschnitte mit großteils konstanten Werten. Im Tagesverlauf und vor allem zwischen zwei Belegungen schwanken die Werte oftmals jedoch deutlich, auch wenn sich keine Personen im Raum befinden. Im Netzwerk können diese Schwankungen schließlich zunächst nicht richtig klassifiziert werden.

Für die Daten von *Speaker's Corner* zeigen sich ähnliche Ergebnisse. Wie Abbildung 4.7 (b) entnommen werden kann, ergeben sich auch hier keine Unterschiede zwischen den einzelnen Architekturen. Insgesamt zeigen die Netze im Mittel etwas bessere Ergebnisse. Dies ist darauf zurückzuführen, dass der Seminarraum meist von großen Gruppen und nicht von Einzelpersonen genutzt wird und somit deutlichere Schwankungen in den einzelnen Kurven der Sensoren vorliegen.

Übertragung der trainierten Netze auf unbekannte Daten

Abbildung 4.8 (a) zeigt unter anderem die Werte für die sechs besten Netze aus dem vorhergehenden Abschnitt. Das beste Ergebnis erzielt das Recurrent Neural Network `DeepNetArch3_2019_07_17_22_51_14` (violett) mit zwei Layern und einer Fenstergröße von 12. Es erreicht insgesamt eine Accuracy von 97%. Auf den Testdaten konnte eine Accuracy von 78% erreicht werden mit einer Specificity von 75%.



(a) Recurrent Neural Networks getestet mit *Round Table* und *Speaker's Corner*

(b) CO_2-Verlauf und vorhergesagter Belegungsstatus für *Speaker's Corner*

Abbildung 4.8: Mit RNN vorhergesagter Belegungsstatus für verschiedene Datensätze

Aus Abbildung 4.8 (a) wird außerdem ersichtlich, dass es nicht möglich ist, Trainingsdaten bzw. erfolgreich trainierte Netze auf beliebige Räume zu übertragen. Auch wenn die blauen Punkte oberhalb der Winkelhalbierenden liegen, schaffen es die Netze kaum, eine korrekte Vorhersage zu treffen. Für *Speaker's Corner* sinkt die Accuracy auf 60% bei einer Sensitivity von nur knapp über 50%. Dabei zeigen sich sowohl etliche False Positives als auch False Negatives. Auch andersherum können die für *Speaker's Corner* trainierten Netze keine verlässlichen Vorhersagen über den Belegungsstatus von *Round Table* treffen. Dieses Verhalten ist vor allem auf die deutlich unterschiedlichen Gegebenheiten der Räume zurückzuführen.

Um dem oben beschriebenen Problem entgegenzuwirken, werden die Netze anschließend mit einem gemischten Datensatz von beiden Räumen trainiert. Die Daten werden standardisiert übergeben und in Trainings-, Validierungs- und Testmenge unterteilt. Insgesamt ist keine Verbesserung ersichtlich. Wie Abbildung 4.8 (b) zeigt, schwanken hier die vorhergesagten Werte (grün) im Bereich um 0.5, das Netz tendiert also zu keiner der beiden Klassen. Die blaue Kurve zeigt exemplarisch den Verlauf des CO_2-Gehalts, während der orangefarbene Plot den tatsächlichen Belegungsstatus anhand der Kalenderdaten anzeigt. Es lässt sich daher vermuten, dass die gemessenen Werte der Räume zu unterschiedlich sind und die Trainingsmenge zu klein, um für beide Räume relevante Muster zu bilden und die Gewichte entsprechend für beide passend zu berechnen. Allgemein zeigt sich, dass die Netze bei einem gemischten Datensatz auch häufig dazu tendieren, immer frei oder immer belegt vorherzusagen.

In Tabelle 4.9 sind die jeweiligen relevanten Hyperparameter der sechs besten Netze gelistet. Insgesamt zeigt sich, dass *Stochastic Gradient Descent* der erfolgreichste Optimierungsalgorithmus ist. Die Anzahl der Layer n_h scheint mit maximal 2 Schichten und $2w_s$ Perceptrons pro Schicht absolut ausreichend. Auch geringer Dropout d_r und kleine Regulierungsgewichte l_{reg} scheinen für die erfolgreiche Anpassung der Netze passend zu sein. Auffällig ist, dass die gewählte Größe w_s der Fenster und damit das Wissen über zurückliegende Punkte nicht pauschalisiert werden kann.

Name	**Typ**	n_h	w_s	d_r	l_{reg}	*opt*
DeepNetArch3L1_2019_07_17_14_04_16	RNN	1	4	0.2	0.001	SGD
DeepNetArch1_2019_07_17_16_54_14	LSTM	2	4	0.1	0.001	SGD
DeepNetArch2_2019_07_17_19_40_01	GRU	2	12	0.0	0.001	SGD
DeepNetArch3_2019_07_17_22_51_14	RNN	2	12	0.0	0.002	SGD
DeepNetArch1L1_2019_07_18_00_29_35	LSTM	1	12	0.1	0.001	Adam
DeepNetArch1L1_2019_07_18_00_45_13	LSTM	1	6	0.0	0.001	Adam

Abbildung 4.9: Hypterparameter der trainierten Deep Neural Networks

Die aus Grid Search ermittelten Hyperparameter werden als Startwerte für die Suche mit *Scikit-Optimize* herangezogen.

Einfluss der einzelnen Sensoren

Grundsätzlich liegt die Vermutung nahe, dass die einzelnen Sensoren auch bei neuronalen Netzen einen unterschiedlichen Einfluss auf eine erfolgreiche Vorhersage des Belegungsstatus haben, wie dies bereits bei Support Vector Maschine und Random Forests der Fall war. Bei Durchsicht der Daten zeigt sich, dass vor allem CO_2-Gehalt und Gaswiderstand als relevante Parameter in Frage kommen. Wie Abbildung 4.10 zeigt, nähert sich die Vorhersage des ausgewählten Netzes (grün) dem Anstieg des CO_2-Verlaufs (blau) an. Bei steigenden Werten von CO_2-Gehalt und sinkendem Gaswiderstand (braun) wird der Raum als belegt klassifiziert. Bleiben beide Werte konstant oder ändern sie sich nur geringfügig, so wird der Raum als frei klassifiziert. Das Ende einer Besprechung wird scheinbar durch den abrupten Abfall der CO_2-Kurve erkannt.

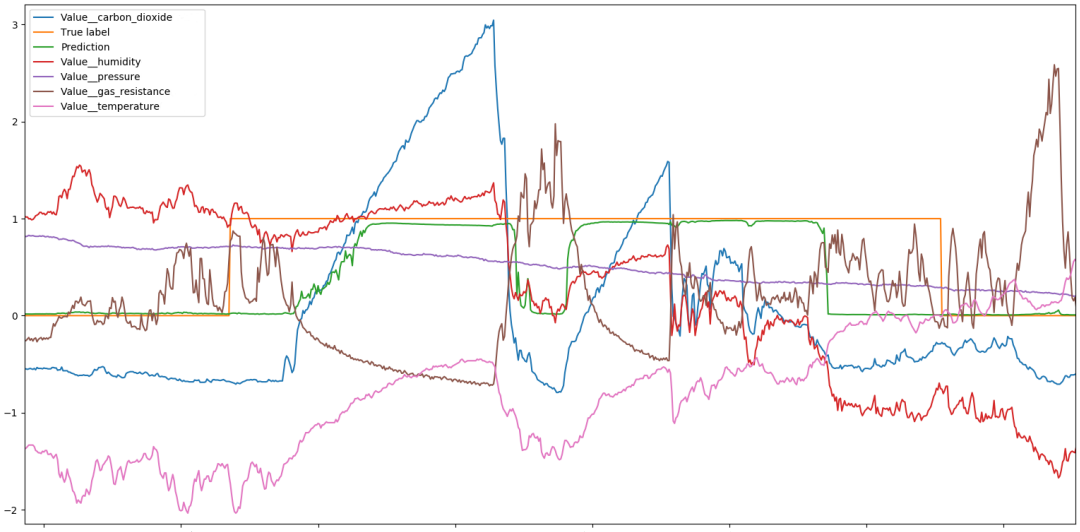


Abbildung 4.10: Einfluss der einzelnen Sensoren auf den Belegungsstatus

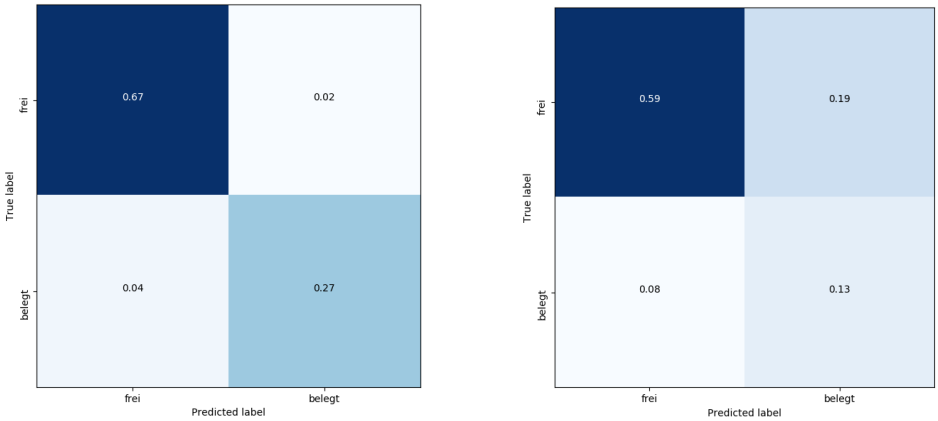
Aufgrund der erläuterten Vermutung wird schließlich das *Scikit-Optimize* Modell um einen zusätzlichen Parameter erweitert, um die Wahl der relevanten Features zu optimieren. Die einzelnen Sensoren werden variabel ausgewählt, um das Netz mit dem kleineren Set zu trainieren. Anders als vermutet, zeigen verschiedene Testläufe, dass die Netze die besten Ergebnisse liefern, die alle Sensoren betrachten. Lediglich Druck (lila) macht in den Ergebnissen keinen Unterschied. Die Netze mit dem Teilset an Sensordaten erkennen auch in den Temperatur-

und Luftfeuchtigkeitskurven relevante Informationen. Wie schon bei klassischen Verfahren geeignetes Feature-Engineering, hat bei Recurrent Neural Networks die Ergänzung von zusätzlichen Messwerten eine ähnliche Wirkung auf die Zuverlässigkeit der Modelle.

Verbesserung der Neuronalen Netze

Nachdem weder die Übertragung der Trainingsdaten auf beliebige Räume, noch die Verwendung gemischter Datensätze brauchbare Ergebnisse erzielten, werden die Netze fortan weiterhin mithilfe der Daten eines Raumes trainiert, allerdings mit einer größeren Datenmenge. Für das erneute Training stehen nun Daten von 20 Wochentagen zu Verfügung. Zum Vergleich werden Testdaten von *Round Table* und *Europa* herangezogen, da diese Räume ähnliche Abmessungen und Gegebenheiten besitzen. Nach Durchsicht der Daten und Abgleich mit den gemessenen Werten des Bewegungssensors zeigt sich, dass die Kalendertermine *Round Table* zuverlässiger den tatsächlichen Belegungsstatus widerspiegeln, als die von *Europa*.

Durch die größere Trainingsmenge von *Round Table* können die Ergebnisse deutlich verbessert werden, wie in Abbildung 4.11 (a) zu sehen. Nach Anpassung der Hyperparameter erreicht das beste RNN eine Accuracy von 94% mit einer Sensitivity von 97% und einer Specificity von 87%. Das Netz liegt also zu 97% richtig, wenn der Raum frei ist. Auch die Übertragung der Netze auf Räume ähnlicher Abmessungen liefert deutlich bessere Ergebnisse. Aufgrund der Unsicherheit der Testdaten können diese allerdings nicht zuverlässig quantifiziert werden.



(a) RNN trainiert mit *Round Table*

(b) RNN getestet mit *Europa*

Abbildung 4.11: Confusion Matrix für RNN für verschiedene Datensätze

Die Werte aus der Confusion Matrix in Abbildung 4.11 (b) können nicht grundsätzlich als Referenz genommen werden. Gemäß der Label aus den Kalenderdaten ergibt sich eine True positive rate (TPR) von 76% und eine True negative rate (TNR) von etwa 62%. Tatsächlich lassen sich ein Großteil der fehlerhaften Vorhersagen für *Europa* anhand der Daten erklären. In Abbildung 4.12 sieht man im ersten Bereich, dass die Vorhersage (grün) die Belegung gemäß Kalender (orange) erst später erkennt. Aufgrund des ebenso verzögerten Anstiegs des CO_2-Gehalts (blau) lässt sich sagen, dass sich erst später Personen im Raum aufgehalten haben und die Vorhersage somit korrekt ist. In der Confusion Matrix wird dies dennoch als False negative gewertet. Im hinteren Bereich zeigen sich ebenfalls deutliche Spitzen im CO_2-Gehalt, obwohl der Raum nicht gebucht war. Es scheint plausibel, dass der Raum zu diesem Zeitpunkt auch ohne Buchung belegt war, obwohl dies laut Confusion Matrix als False positive zählt.

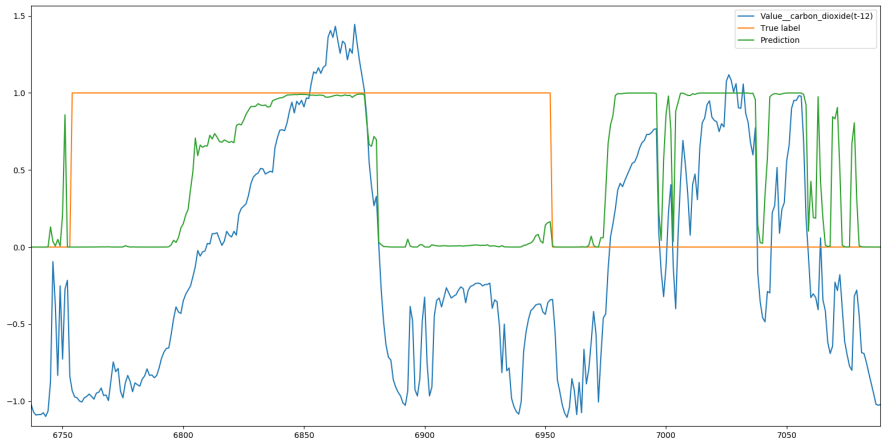


Abbildung 4.12: Fehlerhafte Testdaten von *Round Table*

Tabelle 4.13 zeigt die relevanten Hyperparameter der verbesserten Netze. Es zeigt sich, dass ein Layer ausreicht, wenn die Anzahl der Perceptrons bei 350 bis 400 liegt. Als sinnvolle Fenstergröße ergibt sich bei zwölf Zeitstempeln etwa eine Minute. Ein Dropout von höchstens 10% und die Anwendung der Sigmoid-Aktivierungsfunktion brachten die besten Ergebnisse.

Name	n_h	u	w_s	d_r	a_f
2019_07_17_09_11_27_SimpleRNN	1	398	8	0.05	*sigmoid*
2019_07_17_09_16_55_SimpleRNN	1	389	11	0.1	*sigmoid*
2019_07_17_09_19_39_SimpleRNN	1	398	12	0.01	*sigmoid*
2019_07_17_09_43_48_SimpleRNN	1	337	12	0.1	*sigmoid*
2019_07_17_09_53_03_SimpleRNN	1	347	12	0.1	*relu*

Abbildung 4.13: Hypterparameter der trainierten Recurrent Neural Networks

Zusammenfassung und Ausblick

Abschließend lässt sich sagen, dass Deep Neural Networks zwar ein mächtiges Tool für die Zeitreihenklassifizierung sind, allerdings viele Unsicherheiten auftreten. Es hat sich gezeigt, dass vor allem die Genauigkeit der Trainingsdaten einen

großen Einfluss auf die Güte der Vorhersagen hat. So konnten zum einen trainierte Netze nur teilweise erfolgreich auf andere Räume übertragen werden. Auch die Veränderung der Eingabeparameter durch den Einbau einer Klimaanlage verursachte Probleme. Auf den individuellen Datensätzen konnten sehr gute Ergebnisse erzielt werden, eine Validierung und Nachbereitung der Trainingsdaten anhand von installierten Bewegungssensoren wirkte sich positiv auf die Genauigkeit aus. Ausschlaggebend für die korrekte Klassifizierung waren CO_2-Gehalt, Gaswiderstand, Temperatur und Luftfeuchtigkeit. Insgesamt zeigten Netze mit maximal einem Layer, geringem Dropout und kleiner Fenstergröße die besten Ergebnisse.

Die getesteten Neuronalen Netze tendieren ebenfalls zu Overfitting. Aufgrund der geringen Menge an Trainingsdaten passt sich das Netz auch kleinen Schwankungen der Sensoren exakt an. Während die Trainingsmenge daher fast komplett richtig vorhergesagt wird, können andere Datensätze nicht richtig prognostiziert werden. Um robustere Ergebnisse zu erhalten, werden mehr Daten benötigt. Zusätzlich sollten die gemessenen Daten besser verteilt sein und verschiedene Bedingungen umfassen. Da die Messwerte aus dem Sommer stammen, ist nicht zu erwarten, dass diese auch im Winter die richtigen Ergebnisse liefern. Räume mit ähnlichen Gegebenheiten können grundsätzlich gemeinsam trainiert werden. Um bessere Ergebnisse zu erhalten, ist eine Trennung trotzdem sinnvoll.

Während sich bei klassischen Verfahren der Einfluss der berechneten Features exakt nachvollziehen lässt, ist es bei Neuronalen Netzen oftmals schwierig, den Grund falscher Klassifizierungen zu finden. Ein großer Vorteil, der sich vor allem im Vergleich zu klassischen Verfahren allerdings gezeigt hat, ist die Vorhersage kontinuierlicher Werte. Um dem Problem einer zu geringen True positive rate entgegenzuwirken, kann der Schwellwert der Umwandlung in diskrete Werte etwas erhöht werden. Auch starke Schwankungen oder Sprünge in den Vorhersagen sind nicht zu erwarten.

Mithilfe Neuronaler Netze können leicht eine Vielzahl weiterer wichtiger Anhaltspunkte prognostiziert werden. Aus ökologischer Sicht wäre es beispielsweise interessant, die Kombination aus laufender Heizung und offenem Fenster zu messen, um entsprechend handeln zu können. Die Schwierigkeit liegt allerdings darin, geeignete Trainingsdaten zu sammeln, weswegen derartige Konstellationen im Rahmen dieser Arbeit nicht weiter verfolgt werden. Allgemein können hier in Zukunft

auch gut Ansätze des Unsupervised Learnings sinnvoll eingesetzt werden.

4.3 Luftqualität und Indoor Air Quality Index

Zum Abgleich der oben beschriebenen Verfahren liegen 63 Abstimmungen vor. 28 davon grün, 27 gelb und acht rot. Die durchschnittliche Dauer der mit grün und gelb bewerteten Besprechungen liegt bei drei Stunden, mit einem Median von zwei Stunden. Die mit rot bewerteten Besprechungen dauerten im Mittel nur 1.5 Stunden und haben einen Median von 45 Minuten. Die Bewertungen wurden anhand der Dauer der Besprechung normiert, anschließend werden für die relevanten Parameter Temperatur in °C, Luftfeuchtigkeit in % und Kohlenstoffdioxid in ppm jeweils Minimum, Maximum und Durchschnitt gebildet. Die Werte können Tabelle 4.14 entnommen werden.

Wert	**Temp**$_{min}$	**Temp**$_{max}$	**Temp**$_{avg}$	**Hum**$_{min}$	**Hum**$_{max}$	**Hum**$_{avg}$	**CO2**$_{min}$	**CO2**$_{max}$	**CO2**$_{avg}$
grün	27.1	30.4	28.9	34.5	40.2	37.2	442.2	1139.1	729.2
gelb	27.4	30.6	29.1	34.1	40.2	37.1	436.7	1121.0	726.1
rot	28.6	31.4	29.9	32.2	39.5	36.0	435.4	1234.7	830.1

Abbildung 4.14: Mittelwerte der relevanten Parameter während Besprechungen in *Speaker's Corner* anhand der qualitativen Erhebung

Es zeigt sich, dass sich die Werte der grünen und gelben Bewertungen nur unwesentlich voneinander unterscheiden. Lediglich die Temperaturwerte bei den gelb bewerteten Besprechungen sind leicht höher als bei den als grün eingestuften. Bei den als rot bewerteten Besprechungen liegen die Temperaturwerte etwa 1°C über den anderen Werten. Auch der durchschnittliche sowie der maximale CO_2-Gehalt liegen mit jeweils etwa 100 ppm deutlich über den anderen Messwerten.

Aus den evaluierten Messungen lässt sich folgern, dass die Luftqualität bei gleichen Parametern, aber kürzerer Dauer als schlechter bewertet wurde. Außerdem wurden vor allem Temperaturen über 30°C als negativ bewertet. Über den Einfluss der Luftfeuchtigkeit lassen sich im Allgemeinen keine Aussagen tätigen.

Ein Zusammenhang mit den in Abschnitt 3.5 vorgestellten Formeln konnte aufgrund der geringen Datenbasis nicht hergestellt werden. Es zeigte sich, dass ne-

ben durchschnittlichen Temperaturen über 30°C vor allem hohe CO_2-Werte über 1000 ppm negativ bewertet wurden. In diesen Fällen liegen auch die berechneten Werte für IAQI und TCI im ungesunden Bereich. Für keine der evaluierten Besprechungen wurde ein IAQI von über 75 (gut) erreicht. Ein Großteil der grün und gelb bewerteten Termine befindet sich im moderaten Bereich mit Ausschlägen nach unten. Aufgrund der Wahl der Bereiche treten diese vor allem in Zusammenhang mit der gemessenen Temperatur über 29°C auf.

Zusammenfassend lässt sich sagen, dass der IAQI einen guten Anhaltspunkt für die Luftqualität gibt. Auch wenn ein exakter Zusammenhang zwischen den gesammelten Bewertungen und den Formeln nicht gezeigt werden konnte, ist eine Tendenz ersichtlich. Vor allem bei quantitativ schlechter Luft waren auch die Einschätzungen der Teilnehmer entsprechend negativ. Ausschläge im mittleren Bereich sowohl nach oben, als auch nach unten, lassen sich einfach durch das subjektive Empfinden einzelner Personen erklären. Diese Beobachtungen korrelieren mit der oben aufgeführten Tabelle.

Anhang A Dashboard der Sensorik

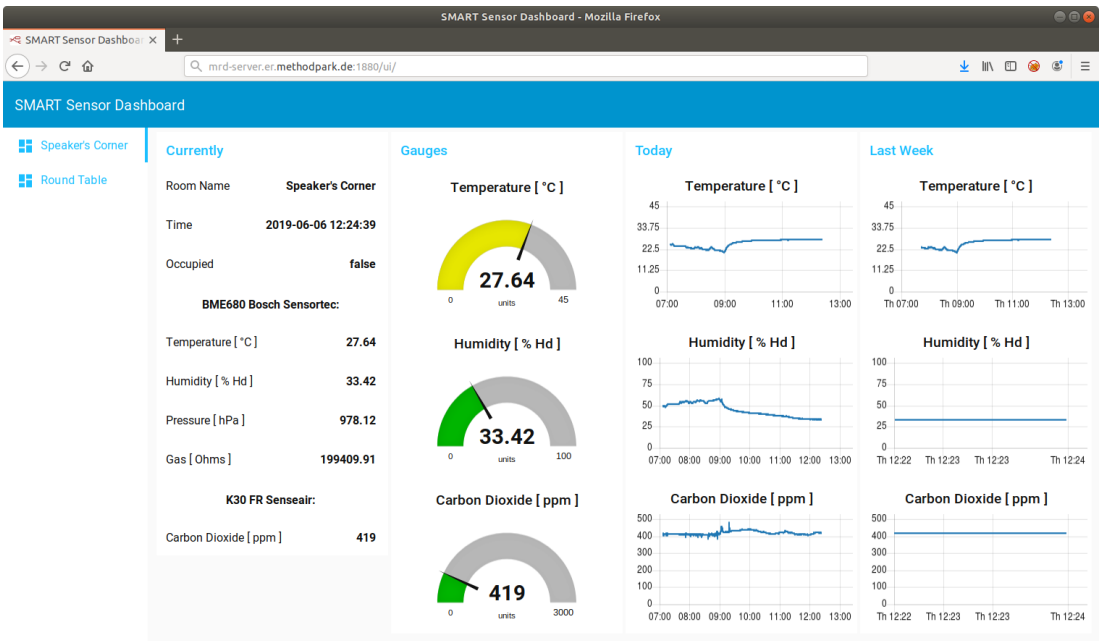


Abbildung 4.15: *Node-RED* Dashboard zur Überwachung der Luftqualität

Anhang B Verkabelung der Sensoren

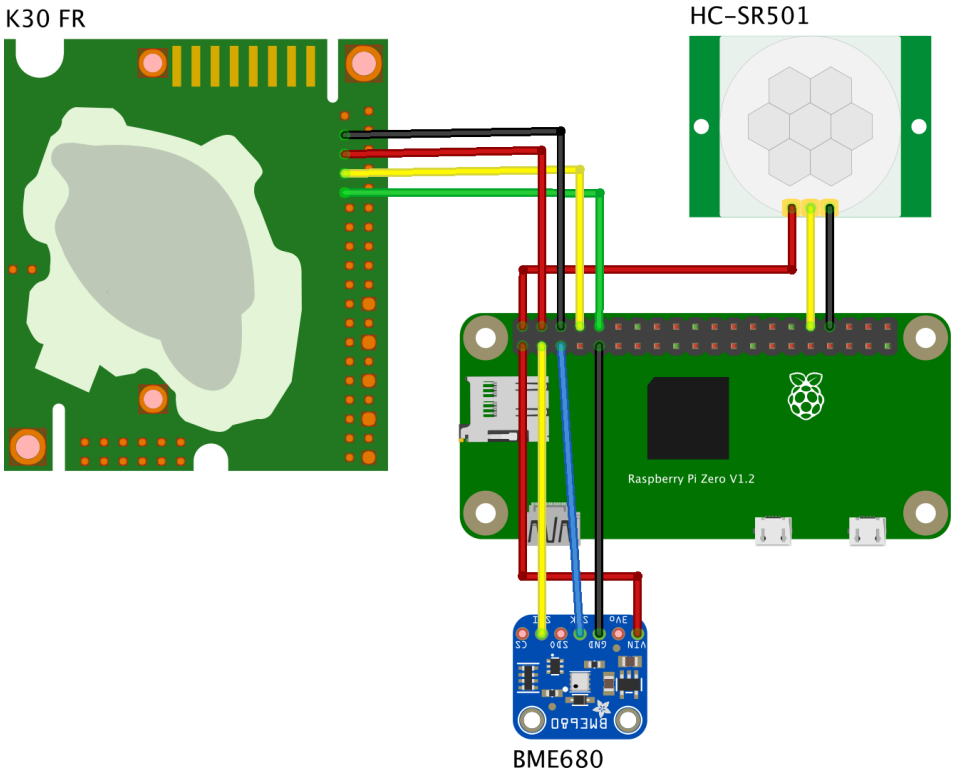


Abbildung 4.16: *Raspberry Pi Zero* mit *BME680*, *K30 FR* und *SR501* Sensoren

Literaturverzeichnis

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Zugriff unter https://www.tensorflow.org/

Allen, J., Macnaughton, P., Satish, U., Santanam, S., Vallarino, J. & Spengler, J. D. (2015). Associations of Cognitive Function Scores with Carbon Dioxide, Ventilation, and Volatile Organic Compound Exposures in Office Workers: A Controlled Exposure Study of Green and Conventional Office Environments. *Environmental Health Perspectives*, *124*. doi:10.1289/ehp.1510037

Archer, T., Melborn, R. & Casanovas, A. (2018). O365: Microsoft Graph and Office 365 API made easy. https://github.com/O365/python-o365. Version 1.0.5. GitHub.

Azizi, S., Bayat, S., Yan, P., Tahmasebi, A., Kwak, J. T., Xu, S., ... Abolmaesumi, P. (2018). Deep Recurrent Neural Networks for Prostate Cancer Detection: Analysis of Temporal Enhanced Ultrasound. *IEEE Transactions on Medical Imaging*, *37*(12), 2695–2703. doi:10.1109/TMI.2018.2849959

Bahdanau, D., Cho, K. & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. Zugriff unter http://arxiv.org/abs/1409.0473

Banks, A., Briggs, E., Borgendale, K. & Gupta, R. (Hrsg.). (2019). *MQTT OASIS Standard*. Version 5.0. Organization for the Advancement of Structured Information Standards. Zugriff unter https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

Bosch Sensortec GmbH. (2019). *BME680 Datasheet: Low power gas, pressure, temperature & humidity sensor*. Version 1.1. BST-BME680-DS001-01. Zu-

griff unter https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME680-DS001.pdf

Cho, K., van Merrienboer, B., Bahdanau, D. & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR*, *abs/1409.1259*. arXiv: 1409.1259. Zugriff unter http://arxiv.org/abs/1409.1259

Chollet, F. et al. (2015). Keras. https://keras.io.

Christ, M., Braun, N., Neuffer, J. & Kempa-Liehr, A. W. (2018). Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing*, *307*, 72–77. doi:10.1016/j.neucom.2018.03.067

Christ, M., Braun, N., Neuffer, J. & Kempa-Liehr, A. W. (2019). Overview on extracted features. Zugriff 19. August 2019 unter https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

Chung, J., Gülçehre, Ç., Cho, K. & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, *abs/1412.3555*. arXiv: 1412.3555. Zugriff unter http://arxiv.org/abs/1412.3555

CO2Meter Inc. (2019). *Application Note AN137: Raspberry Pi UART Interface to K-30 CO2 Sensor*. Zugriff 13. Juni 2019 unter http://www.co2meters.com/Documentation/AppNotes/AN137-K30-sensor-raspberry-pi-uart.pdf

Cockburn, A. (2005). Ports and adapters architecture. Zugriff 1. August 2019 unter http://alistair.cockburn.us/Hexagonal+architecture

Dorffner, G. (1996). Neural Networks for Time Series Processing. *Neural Network World*, *6*, 447–468.

Esling, P. & Agon, C. (2012). Time-series Data Mining. *ACM Comput. Surv.* *45*(1), 12:1–12:34. doi:10.1145/2379776.2379788

Espressif Systems. (2015). *ESP8266EX Datasheet*. Version 4.3. Zugriff unter https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

Fan, J., Upadhye, S. & Worster, A. (2006). Understanding receiver operating characteristic (ROC) curves. *Canadian Journal of Emergency Medicine*, *8*(1), 19–20. doi:10.1017/S1481803500013336

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L. & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963. doi:10.1007/s10618-019-00619-1

Flach, P. (2003). The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC Isometrics. (Bd. 1, S. 194–201).

Fried, L. (2018). *PIR Motion Sensor*. Adafruit Industries. Zugriff 13. Juni 2019 unter https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf

Fried, L. (2019). BME680 Python Computer Wiring. Zugriff 13. Juni 2019 unter https://learn.adafruit.com/adafruit-bme680-humidity-temperature-barometic-pressure-voc-gas/python-circuitpython

Garff, J. (2018). rpi_ws281x: Userspace Raspberry Pi PWM library for WS281X LEDs. https://github.com/jgarff/rpi_ws281x. Version 1.1.0. GitHub.

Gerstl, S. (2018). Was ist MQTT? Zugriff 17. Mai 2019 unter https://www.embedded-software-engineering.de/was-ist-mqtt-a-725485/

Gonçalves, L., Subtil, A., Rosário, O. M. & De Zea Bermudez, P. (2014). ROC curve estimation: An overview. *Revstat – Statistical Journal*, *12*, 1–20.

Grisel, O., Mueller, A., Lars, Gramfort, A., Louppe, G., Prettenhofer, P., ... Alsalhi, H. (2019). scikit-learn/scikit-learn 0.21.2. doi:10.5281/zenodo.3208700

Häring, G. (2015). sqlite3: DB-API 2.0 interface for SQLite databases. https://github.com/ghaering/pysqlite. Version 2.6.0. GitHub.

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction* (2. Aufl.). Springer. doi:10.1007/978-0-387-84858-7

He, C., Yao, Y.-j. & Ye, X.-s. (2017). An Emotion Recognition System Based on Physiological Signals Obtained by Wearable Sensors. In C. Yang, G. S. Virk & H. Yang (Hrsg.), *Wearable Sensors and Robots* (S. 15–25). Singapore: Springer Singapore.

Head, T., MechCoder, Louppe, G., Shcherbatyi, I., fcharras, Vinícius, Z., ... Fabisch, A. (2018). scikit-optimize/scikit-optimize: v0.5.2. doi:10.5281/zenodo.1207017

Hinton, G. E. (2012). A Practical Guide to Training Restricted Boltzmann Machines. In G. Montavon, G. B. Orr & K.-R. Müller (Hrsg.), *Neural Networks:*

Tricks of the Trade: Second Edition (S. 599–619). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-35289-8_32

Hipp, D. R., Kennedy, D. & Mistachkin, J. (2019). About SQLite. The SQLite Development Team. Zugriff 17. Mai 2019 unter https://www.sqlite.org/about.html

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Comput. 9*(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735

Howard, P. & Macdonald, S. (2018). BME680. https://github.com/pimoroni/bme680-python. Version 1.0.5. GitHub.

Hüsken, M. & Stagge, P. (2003). Recurrent neural networks for time series classification. *Neurocomputing*, *50*, 223–235. doi:https://doi.org/10.1016/S0925-2312(01)00706-8

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 1–84. doi:10.1109/IEEESTD.1990.101064

ISO/IEC 25010. (2011). Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. *ISO/IEC 25010:2011*, 1–34. Zugriff unter https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Kanchan, K., Gorai, A. & Goyal, P. (2015). A Review on Air Quality Indexing System. *Asian Journal of Atmospheric Environment*, *9*, 101–113. doi:10.5572/ajae.2015.9.2.101

Keller, M. S. (1999). Take Command: Cron: Job Scheduler. *Linux J. 1999*(65es). Zugriff unter http://dl.acm.org/citation.cfm?id=327966.327981

Kern, S. (2018). DIY – Belegungsanzeige für Besprechungsräume. Zugriff 13. Mai 2019 unter https://www.methodpark.de/blog/diy-belegungsanzeige-fuer-besprechungsraeume/

Kostadinov, S. (2017). Understanding GRU Networks. Zugriff 18. Juli 2019 unter https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Kyrkilis, G., Chaloulakou, A. & Kassomenos, P. (2007). Development of an aggregate Air Quality Index for an urban Mediterranean agglomeration: Relation to potential health effects. *Environment international*, *33*, 670–6. doi:10.1016/j.envint.2007.01.010

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep Learning. *Nature*, *521*, 436–44. doi:10.1038/nature14539

Lewis, K. (2016). Node-RED visual programming for the Internet of Things (IoT) is now a JS Foundation Project. Zugriff 6. Juni 2019 unter https://www.ibm.com/blogs/internet-of-things/open-source-iot/

Liechtl, C. (2017). pySerial. https://github.com/pyserial/pyserial. Version 3.4. GitHub.

Lopez, D. (2019). RNN, LSTM & GRU. Zugriff 26. Juni 2019 unter http://dprogrammer.org/rnn-lstm-gru

Martin, R. C. (2002). *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Mitchell, T. M. (1997). *Machine Learning* (1. Aufl.). New York, NY, USA: McGraw-Hill, Inc.

Moujahid, A. (2016). A Practical Introduction to Deep Learning with Caffe and Python. Zugriff 21. Juni 2019 unter http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Nielsen, M. (2015). *Neural Networks and Deep Learning*. Determination Press. Zugriff unter https://books.google.de/books?id=STDBswEACAAJ

O'Mahony, M. & Hautus, M. (2008). The signal detection theory ROC curve: some applications in food sensory science. *Journal of Sensory Studies*, *23*(2), 186–204. doi:10.1111/j.1745-459X.2007.00149.x

Olah, C. (2015). Understanding LSTM Networks. Zugriff 25. Juni 2019 unter https://colah.github.io/posts/2015-08-Understanding-LSTMs/

OpenJS Foundation. (2019). About Node.js. Zugriff 17. Mai 2019 unter https://nodejs.org/en/about/

OSHA. (1990). Sampling and Analytical Methods: Carbon Dioxide In Workplace Atmospheres. Zugriff 21. Mai 2019 unter https://www.osha.gov/dts/sltc/methods/inorganic/id172/id172.html

Peng, M. L. (2018). Improving neural networks by preventing co-adaptation of feature detectors. Zugriff 17. Juli 2019 unter https://medium.com/@lipeng2/dropout-is-so-important-e517bbe3ffcc

Peng, Y. (2013). Tikz example – SVM trained with samples from two classes. Zugriff 19. Juli 2019 unter http://blog.pengyifan.com/tikz-example-svm-trained-with-samples-from-two-classes/

Powers, D. (2008). Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Mach. Learn. Technol. 2*.

Ramachandran, P., Zoph, B. & Le, Q. V. (2017). Searching for Activation Functions. Zugriff unter http://arxiv.org/abs/1710.05941

Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory Report. Cornell Aeronautical Laboratory.

Ruggieri, M. & Plaia, A. (2012). An aggregate AQI: Comparing different standardizations and introducing a variability index. *Science of The Total Environment*, *420*, 263–272. doi:https://doi.org/10.1016/j.scitotenv.2011.09.019

Russell, S. & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd). Upper Saddle River, NJ, USA: Prentice Hall Press.

Saad, S. M., Shakaff, A. Y. M., Saad, A. R. M., Yusof, A. M., Andrew, A. M., Zakaria, A. & Adom, A. H. (2017). Development of indoor environmental index: Air quality index and thermal comfort index. (Bd. 1808, S. 020043). doi:10.1063/1.4975276

Satish, U., Mendell, M. J., Shekhar, K., Hotchi, T., Sullivan, D., Streufert, S. & Fisk, W. J. (2012). Is CO_2 an Indoor Pollutant? Direct Effects of Low-to-Moderate CO_2 Concentrations on Human Decision-Making Performance. *Environmental health perspectives*, *120*. doi:10.1289/ehp.1104789

SenseAir AB. (2015). *Product Specification Senseair CO_2 Engine^{TM} K30 FR: Fast Response CO_2 Sensor*. Version 7. PSP109. Zugriff unter https://rmtplusstoragesenseair.blob.core.windows.net/docs/publicerat/PSP109.pdf

Sharma, A. (2017). Understanding Activation Functions in Neural Networks. Zugriff 24. Juni 2019 unter https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Susto, G. A., Cenedese, A. & Terzi, M. (2018). Time-Series Classification Methods: Review and Applications to Power Systems Data. (S. 179–220). doi:10.1016/B978-0-12-811968-6.00009-7

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag.

Vernon, V. (2013). *Implementing Domain-Driven Design* (1. Aufl.). Addison-Wesley Professional.

Vert, J., Tsuda, K. & Schölkopf, B. (2004). A Primer on Kernel Methods. In *Kernel Methods in Computational Biology* (S. 35–70). Cambridge, MA, USA: MIT Press.

Yiu, T. (2012). Understanding Random Forest: How the Algorithm Works and Why it Is So Effective. Zugriff 10. Juli 2019 unter https://towardsdatascience.com/understanding-random-forest-58381e0602d2