

**Entwicklung und Analyse von Algorithmen für die  
automatische Indoor-Ortung basierend auf WiFi-Fingerprints**

Bachelorarbeit

Name des Studiengangs

Angewandte Informatik

**Fachbereich 4**

vorgelegt von

Friedrich Völkers

Datum:

Berlin, 22.08.2024

Erstgutachter: Prof. Dr. Alexander Huhn

Zweitgutachter: Prof. Dr.-Ing. Thomas Schwotzer

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Grundlagen</b>	<b>2</b>
2.1 Indoor-Ortung via WiFi Fingerprints . . . . .	2
2.2 Technische Grundlagen . . . . .	2
2.2.1 BSSID . . . . .	2
2.2.2 SSID . . . . .	3
2.2.3 RSSI . . . . .	3
2.3 Zusammenhang zwischen Signalstärke und Distanz . . . . .	3
<b>3 Algorithmen</b>	<b>5</b>
3.1 K-Nearest Neighbors (KNN) . . . . .	5
3.1.1 Distanzmetriken . . . . .	6
3.1.2 Anzahl der Nachbarn . . . . .	7
3.1.3 Gewichtung . . . . .	7
3.2 Support Vector Machines (SVM) . . . . .	8
3.2.1 Auswahl des Kernels . . . . .	9
3.2.2 Regularisierungsparameter C und Kernel-Parameter <code>gamma</code> . . . . .	10
3.3 Random Forest . . . . .	10
<b>4 Systemarchitektur und Implementierung</b>	<b>12</b>
4.1 Datenbankstruktur . . . . .	12
4.1.1 Tabelle <code>rooms</code> . . . . .	12
4.1.2 Tabelle <code>measurements</code> . . . . .	12
4.1.3 Tabelle <code>routers</code> . . . . .	13
4.1.4 Tabelle <code>measurement_router</code> . . . . .	13
4.2 Implementierung des Servers . . . . .	13

4.2.1	FastAPI . . . . .	14
4.2.2	Grafana und Prometheus . . . . .	16
4.3	Implementierung der <i>BVG Detection</i> App . . . . .	16
4.3.1	Wiederinbetriebnahme der App . . . . .	17
4.3.2	Datenspeicherung und Aufnahme der WiFi-Fingerprints . . . . .	18
4.3.3	Austausch der Daten . . . . .	19
4.3.4	Ortung von Räumen . . . . .	21
4.4	Quellcode der Microcontroller . . . . .	22
<b>5</b>	<b>Datenerhebung und Analyse der Algorithmen</b>	<b>24</b>
5.1	Aufnahme von WiFi-Fingerprints für die Analyse der Algorithmen . . . . .	24
5.2	Beschreibung der Testanwendung . . . . .	24
5.3	Analyse der Algorithmen und Parameter . . . . .	26
5.3.1	Voruntersuchung der nicht im Detail betrachteten Parameter . . . . .	26
5.3.2	Untersuchung der im Detail betrachteten Parameter . . . . .	29
5.4	Untersuchungen des Einflusses verschiedener Datenaufbereitungsmethoden . . . . .	32
5.5	Erweiterte Untersuchungen . . . . .	41
<b>6</b>	<b>Fazit</b>	<b>42</b>
6.1	Diskussion der Ergebnisse . . . . .	42
6.2	Ausblick auf zukünftige Arbeiten . . . . .	42

# Code Beispiele

4.1	Beispiel für eine Anfrage an /measurements/predict . . . . .	15
4.2	Beispiel für eine Anfrage an /measurements/predict . . . . .	16
4.3	Beispiel einer API-Antwort von /measurements/predict . . . . .	16
4.4	Netzwerkkonfiguration der App . . . . .	20
4.5	<i>MicroPython</i> -Quellcode für die Durchführung eines WiFi-Scans und die Raumbestimmung über die API . . . . .	22
5.1	<i>YAML</i> -Konfigurationsdatei der Testanwendung . . . . .	25

# Abbildungsverzeichnis

2.1	Exemplarischer Zusammenhang zwischen RSSI-Werten und der Entfernung nach dem Pfadverlustmodell . . . . .	4
3.1	Vergleich der Distanzmetriken Euklidisch und Sørensen . . . . .	7
3.2	Darstellung einer Hyperebene in einem Support Vector Machine (SVM) mit zwei Features (Bildquelle: Basecamp 2021) . . . . .	9
3.3	Darstellung von nicht linear trennbaren Daten in einem SVM (Bildquelle: Basecamp 2021) . . . . .	9
3.4	Darstellung eines Entscheidungsbaums anhand der Beispieldaten . . . . .	11
4.1	FastAPI Dokumentation . . . . .	14
4.2	Grafana Dashboard der <i>FastAPI</i> . . . . .	17
4.3	Grafana Dashboard der <i>MariaDB</i> . . . . .	17
4.4	Menüpunkt <i>Ort/Fingerprint aufnehmen</i> der <i>BVG Detection</i> -App . . . . .	18
4.5	Menüpunkt <i>Fingerprints verwalten</i> der <i>BVG Detection</i> -App . . . . .	19
4.6	Menüpunkt <i>Datenbank teilen</i> der <i>BVG Detection</i> -App . . . . .	20
4.7	Einstellungen der Datenaufbereitungsmethoden . . . . .	21
5.1	Anzahl der Messungen pro Raum . . . . .	24
5.2	Vergleich des gewichteten ( <code>weights = distance</code> ) und ungewichteten K-Nearest-Neighbor (KNN)-Algorithmus ( <code>weights = uniform</code> ) . . . . .	26
5.3	Vergleich der Parameter <code>auto</code> und <code>scale</code> des <code>gamma</code> -Parameters für den SVM-Algorithmus mit RBF-Kernel . . . . .	27
5.4	Vergleich der verschiedenen Werte des <code>max_depth</code> Parameters für den Random Forest Algorithmus . . . . .	27
5.5	Vergleich der verschiedenen Werte des <code>n_estimators</code> Parameters für den Random Forest Algorithmus . . . . .	28
5.6	Vergleich der Dauer des Random Forest Algorithmus in Abhängigkeit der Anzahl der Bäume . . . . .	28
5.7	Vergleich der Parameter in Abhängigkeit der Anzahl der Messungen . . . . .	30

5.8 Vergleich der durchschnittlichen Genauigkeit der Parameter in Abhangigkeit der Anzahl der Messungen . . . . .	30
5.9 Vergleich der Parameter des Random Forest Algorithmus in Abhangigkeit der Anzahl der Messungen . . . . .	31
5.10 Vergleich der Genauigkeit in Abhangigkeit der Strategie zum Umgang mit fehlenden Werten . . . . .	33
5.11 Vergleich der durchschnittlichen Genauigkeit in Abhangigkeit der Strategie zum Umgang mit fehlenden Werten . . . . .	33
5.12 Vergleich der Genauigkeit in Abhangigkeit der Strategie zur Auswahl der Router	34
5.13 Vergleich der durchschnittlichen Genauigkeit in Abhangigkeit des Schwellenwerts fur die Anzahl der Messungen eines Routers . . . . .	35
5.14 Vergleich der durchschnittlichen Genauigkeit in Abhangigkeit des Schwellenwerts fur die Mindestsignalstarke eines Routers . . . . .	36
5.15 Darstellung der verschiedenen Skalierungsmethoden . . . . .	38
5.16 Vergleich des KNN-Algorithmus mit den verschiedenen Skalierungsmethoden und beiden Gewichtungsfunktionen . . . . .	39
5.17 Vergleich der durchschnittlichen Genauigkeit des KNN-Algorithmus in Bezug auf die verschiedenen Skalierungsmethoden und beide Gewichtungsfunktionen . . . . .	39
5.18 Vergleich der Genauigkeit in Abhangigkeit der Skalierungsstrategien . . . . .	40
5.19 Vergleich der durchschnittlichen Genauigkeit in Abhangigkeit der Skalierungsstrategien . . . . .	40
5.20 Vergleich der Genauigkeit in Abhangigkeit der Anzahl an Messungen pro Raum und auf dem Flur . . . . .	41

# Tabellenverzeichnis

3.1	Beispielwerte von WiFi-Fingerprints aus der Offline-Phase . . . . .	5
3.2	Beispielwerte eines WiFi-Fingerprints aus der Online-Phase . . . . .	5
3.3	Berechnete Distanzen der Beispieldaten . . . . .	7
3.4	Gewichtete Distanzen der drei nächstgelegenen Trainingsdaten . . . . .	8
4.1	Struktur der Tabelle <code>rooms</code> . . . . .	12
4.2	Struktur der Tabelle <code>measurements</code> . . . . .	13
4.3	Struktur der Tabelle <code>routers</code> . . . . .	13
4.4	Struktur der Tabelle <code>measurement_router</code> . . . . .	13
4.5	Übersicht der Endpunkte der <i>FastAPI</i> -Anwendung. . . . .	14

# 1 Einleitung

In der vorliegenden Arbeit werden verschiedene Algorithmen zur Indoor-Ortung basierend auf WiFi-Fingerprints analysiert und implementiert. Dafür werden diese Algorithmen ausführlich in Kapitel 3 beschrieben, in Kapitel 4 implementiert und in den Kapitel 5.3, 5.4 und 5.5 mit der in Kapitel 5.2 beschriebenen Testanwendung analysiert und verglichen. Grundlage der Untersuchungen sind Datensätze, welche in Kapitel 5.1 erfasst wurden. Das Ziel dieser Arbeit ist es, die Genauigkeit der verschiedenen Algorithmen zu untersuchen und zu ermitteln unter welchen Bedingungen und mit welchem Algorithmus in den meisten Fällen eine korrekte Ortung möglich ist.

Die Ergebnisse dieser Arbeit sollen als Grundlage für die automatischen Indoor-Ortung in der HTW Berlin dienen, sodass zum Beispiel Microcontroller wie der *ESP32* autonom den Raum erkennen können, in dem sie sich befinden. Des Weiteren wird die App *BVG Detection* wieder in Betrieb genommen und dazu verwendet WiFi-Fingerprints zu erfassen und die Position des Nutzers zu bestimmen.

Der verwendete Quellcode ist auf der *GitLab*-Instanz der HTW Berlin unter <https://gitlab.rz.htw-berlin.de/s0585012/wifi-fingerprint-based-indoor-localization> gespeichert.

## 2 Grundlagen

### 2.1 Indoor-Ortung via WiFi Fingerprints

WiFi-Fingerprinting ist eine Technik die zur Lokalisierung in Innenräumen verwendet werden kann. Dabei werden an verschiedenen Positionen in einem Gebäude die Signalstärken der empfangenden WLAN-Router - die sogenannten Access Points - gemessen und in einer Datenbank gespeichert. Dadurch dass an verschiedenen Positionen verschiedene Access Points unterschiedlich stark empfangen werden, entsteht eine Art Fingerabdruck dieser Position, welcher verwendet werden kann um die Position eines Geräts anhand der empfangenen Access Points und deren Signalstärke zu bestimmen.

Das Prinzip der WiFi-Fingerprinting-basierten Ortung kann in eine Offline- und eine Online-Phase unterteilt werden. In der Offline-Phase werden die Fingerprints aufgezeichnet und mit den dazugehörigen Positionen gespeichert. In der Online-Phase wird die aktuelle Position eines Geräts bestimmt, indem dieses ebenfalls einen WiFi-Fingerprint aufzeichnet und diesen mit den bekannten Fingerprints in der Datenbank abgleicht. Die Position des Geräts wird anschließend anhand des am besten passenden Fingerprints bestimmt.

Neben der Indoor-Ortung mittels WiFi-Fingerprinting können auch andere Technologien wie Bluetooth, Ultra-wideband, RFID und Zigbee verwendet werden, wobei jede Technik seine Vor- und Nachteile mit sich bringt. Die Vorteile des WiFi-Fingerprinting sind, dass diese in vielen Fällen verwendet werden kann ohne zusätzliche Hardware zu installieren, da die meisten Gebäude bereits über eine WLAN-Infrastruktur verfügen. Zudem ist die Reichweite von WLAN-Signalen größer als die von Bluetooth oder RFID, wodurch insgesamt weniger Hardware benötigt wird. Der Nachteil dieser Technik ist jedoch der zeitliche Aufwand beim Erstellen der Fingerprints während der Offline-Phase<sup>1</sup>.

### 2.2 Technische Grundlagen

Ein WiFi-Fingerprint kann definiert werden als eine Liste der empfangenen Access Points, wobei in jedem Eintrag die MAC-Adresse und die Signalstärke steht.<sup>2</sup> In dieser Arbeit wurden diese Einträge um die Klarnamen der Netzwerke - die so genannte SSID - erweitert.

#### 2.2.1 BSSID

Der Basic Service Set Identifier (BSSID) ist die eindeutige Kennung eines Access Points und entspricht der MAC-Adresse des Access Points. Diese Adresse ist unveränderlich und kann dazu verwendet werden einen Access Point eindeutig zu identifizieren.<sup>3</sup>

---

<sup>1</sup> Shang und Wang (2022), S. 726 ff.

<sup>2</sup> Shang und Wang (2022), S. 729

<sup>3</sup> Amirisoori u. a. (2017), S. 27

## 2.2.2 SSID

Der Service Set Identifier (SSID) ist ein eindeutiger Bezeichner, der ein drahtloses Netzwerk kennzeichnet und aus einer alphanumerische Zeichenfolgen - wie zum Beispiel *eduroam* oder *FRITZ!Box 7590 XY* - besteht. Im Gegensatz zur BSSID kann die SSID frei gewählt werden, wodurch mehrere Access Points die selbe SSID haben können.<sup>4</sup>

## 2.2.3 RSSI

Der Received Signal Strength Indicator (RSSI) ist ein relativer Indikator für die Empfangsstärke eines kabellosen Kommunikationssystems nach IEEE 802.11 Standard und gibt die Qualität eines empfangen Signals auf einer Skala von -100 bis 0 in der Einheit dBm an. Je größer der Wert ist, also je näher dieser an 0 ist, desto besser ist das Signal.<sup>5</sup>

## 2.3 Zusammenhang zwischen Signalstärke und Distanz

Da für Indoor-Ortung mittels WiFi-Fingerprints die RSSI-Werte der Access Points verwendet werden, ist es hilfreich den Zusammenhang zwischen der Signalstärke un der Entfernung zu verstehen. Die RSSI-Werte sind abhängig von der Entfernung zwischen dem Sender und dem Empfänger und nehmen mit zunehmender Distanz ab. Der Zusammenhang zwischen dem RSSI-Wert und der Entfernung kann durch das Pfadverlustmodell (siehe Gleichung 2.1) beschrieben werden.

$$RSSI = -10 * n * \log_{10}(d) + C \quad (2.1)$$

Dabei ist  $n$  der Pfadverlustexponent, der je nach Umgebung variiert,  $d$  die Distanz zwischen Sender und Empfänger und  $C$  eine Konstante, die die Systemverluste berücksichtigt.<sup>6</sup>

In dem Paper *RSSI-Based Indoor Localization With the Internet of Things* wurden die Parameter des Pfadverlustmodells experimentell in einer Umgebung untersucht, welche mit der in dieser Arbeit verwendeten vergleichbar ist.<sup>7</sup> Die Experimente wurden in einem Forschungslabor (10,8 m x 7,3 m) - welches mit mehreren WiFi- und Bluetooth-Geräten und Computern ausgestattet ist - durchgeführt. In dieser Umgebung ergaben die Untersuchungen einen Pfadverlustexponenten von  $n = 2.013$  und eine Konstante von  $C = -49.99$  dBm.<sup>8</sup>

Das Pfadverlustmodell mit den Ergebnissen aus der erwähnten Studie ist in Abbildung 2.1 dargestellt und dient als Grundlage für die in Kapitel 5.4 untersuchten Datenaufbereitungsschritte.

---

<sup>4</sup> Masoud, Jaradat und Alia (2022), S. 5460

<sup>5</sup> IoT [mesh] (2024)

<sup>6</sup> Sadowski und Spachos (2018), S. 30153

<sup>7</sup> Sadowski und Spachos (2018)

<sup>8</sup> Sadowski und Spachos (2018), S. 30157

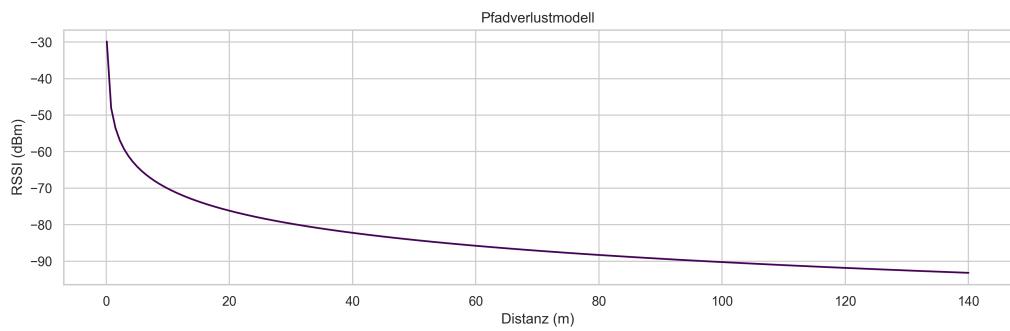


Abbildung 2.1: Exemplarischer Zusammenhang zwischen RSSI-Werten und der Entfernung nach dem Pfadverlustmodell

### 3 Algorithmen

Für die Vorhersage einer Position bei der Indoor-Ortung mittels WiFi-Fingerprints können verschiedene Algorithmen verwendet werden. Es wurde sich dafür entschieden in dieser Arbeit die Modelle K-Nearest Neighbors (KNN), Support Vector Machines (SVM) und Random Forest zu verwenden, da diese Modelle bereits in anderen Untersuchungen gute Ergebnisse erzielen konnten.<sup>1,2</sup>

Im Folgenden werden die drei verwendeten Algorithmen detailliert beschrieben und für eine bessere Verständlichkeit anhand eines Beispiels, welches in Tabelle 3.1 dargestellt ist, erläutert. In diesem Beispiel wurden die RSSI-Werte von sechs Messungen in drei verschiedenen Räumen sowie drei Access Points simuliert. In der Tabelle 3.2 sind die RSSI-Werte von einer Messung in einem unbekannten Raum, für die eine Vorhersage gemacht werden soll, aufgeführt. Die in den Tabellen dargestellten Werte basieren nicht auf realen Messungen und sind hypothetische Beispielwerte, welche gewählt wurden, um die Funktionsweise der Algorithmen zu veranschaulichen und die Nachvollziehbarkeit der Berechnungen zu erleichtern.

Messung	Raum	AP1	AP2	AP3
1	A	-45 dBm	-60 dBm	-70 dBm
2	A	-46 dBm	-61 dBm	-71 dBm
3	B	-55 dBm	-65 dBm	-75 dBm
4	B	-54 dBm	-66 dBm	-74 dBm
5	C	-50 dBm	-70 dBm	-80 dBm
6	C	-51 dBm	-69 dBm	-79 dBm

Tabelle 3.1: Beispielwerte von WiFi-Fingerprints aus der Offline-Phase

Messung	Raum	AP1	AP2	AP3
-	-	-52 dBm	-68 dBm	-78 dBm

Tabelle 3.2: Beispielwerte eines WiFi-Fingerprints aus der Online-Phase

#### 3.1 K-Nearest Neighbors (KNN)

Der KNN Algorithmus ist ein überwachter Lernalgorithmus, der auf dem Konzept der Nähe basiert und zur Lösung von Klassifikations- und Regressionsproblemen verwendet werden kann. Der Algorithmus funktioniert so, dass ein Datenpunkt mit den vorhandenen Datenpunkten in den Trainingsdaten verglichen wird und die Distanz zu jedem Datenpunkt berechnet wird. Dabei wird die Distanz der einzelnen Features, welche in dem Fall der Indoor-Ortung die RSSI-Werte der Access Points sind, berechnet und aufsummiert. Basierend auf diesen Distanzen werden die  $k$ -Datenpunkte ausgewählt, die den kleinsten Abstand haben. Aus diesen  $k$ -Datenpunkten wird dann die Klasse vorhergesagt, die am häufigsten vertreten ist. Bei der Indoor-Ortung entsprechen die Klassen den verschiedenen Räumen.<sup>3</sup>

<sup>1</sup> Han und He (2018), S. 3

<sup>2</sup> Rezgui u. a. (2017), S. 13

<sup>3</sup> IBM (2024a)

### 3.1.1 Distanzmetriken

Für die Berechnung der Distanzen können verschiedene Distanzmetriken verwendet werden. Die am häufigsten verwendete Distanzmetrik ist der euklidische Abstand. Bei der euklidischen Distanz wird das Quadrat der Abstände zwischen zwei Werten gebildet, über alle Wertepaare aufsummiert und abschließend die Quadratwurzel dieser Summe gezogen (siehe Gleichung 3.1).<sup>4</sup>

$$\text{distance}_{\text{euclidean}}(P, Q) = \sqrt{\sum_{i=1}^d (P_i - Q_i)^2} \quad (3.1)$$

Bei der Sørensen-Distanzfunktion werden die Beträge der Abstände der Datenpunkte aufsummiert und durch die Summe der addierten Wertepaare geteilt (siehe Gleichung 3.2). Grund für die Wahl dieser beiden Metriken sind die Ergebnisse der Arbeit *Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems*, welche mit der Sørensen Distanz gute Ergebnisse erzielen konnte und die Tatsache, dass die euklidische Distanz weit verbreitet ist und der bisherigen Implementierung in der *BVG Detection* App entspricht.<sup>5</sup>

$$\text{distance}_{\text{sorensen}}(P, Q) = \frac{\sum_{i=1}^d |P_i - Q_i|}{\sum_{i=1}^d (P_i + Q_i)} \quad (3.2)$$

Um die Unterschiede dieser beiden Metriken zu veranschaulichen, wurden die Distanzen für alle möglichen Wertepaare zwischen 0 dBm und -100 dBm berechnet und in Abbildung 3.1 dargestellt. Wie zu erkennen ist, ist bei der euklidischen Distanz ausschließlich die Differenz der beiden Werte ausschlaggebend, was sich anhand der symmetrischen Anordnung der Werte in der Heatmap entlang der Geraden, die durch die Punkte (0 dBm, -100 dBm) und (-100 dBm, 0 dBm) sowie (0 dBm, 0 dBm) und (-100 dBm, -100 dBm) verläuft, zeigt. Im Gegensatz dazu ist die Sørensen-Distanz sowohl von der Differenz als auch von der Summe der beiden Werte abhängig, was sich durch das Fehlen einer Symmetrie entlang der genannten Geraden zwischen den Punkten (0 dBm, -100 dBm) und (-100 dBm, 0 dBm) zeigt. Bei kleineren Werten ist ein größerer Abstand erforderlich, um die gleiche Distanz wie bei größeren Werten zu erreichen. Dies könnte sich vorteilhaft auf die Indoor-Ortung auswirken, da die RSSI-Werte nicht linear mit der Entfernung korrelieren, wie in Abbildung 2.1 dargestellt ist.

Für die Beispielwerte ergeben sich dadurch folgende Distanzen

Die sich für die Beispielwerte aus Tabelle 3.1 ergebenen Distanzen sind in Tabelle 3.3 dargestellt.

---

<sup>4</sup> IBM (2024a)

<sup>5</sup> Torres-Sospedra u. a. (2015)

Messung	Raum	Euklidische Distanz	Sørensen Distanz
1	A	11.0	0.084
2	A	9.8	0.077
3	B	5.2	0.034
4	B	4.5	0.030
5	C	3.5	0.020
6	C	2.6	0.015

Tabelle 3.3: Berechnete Distanzen der Beispieldaten

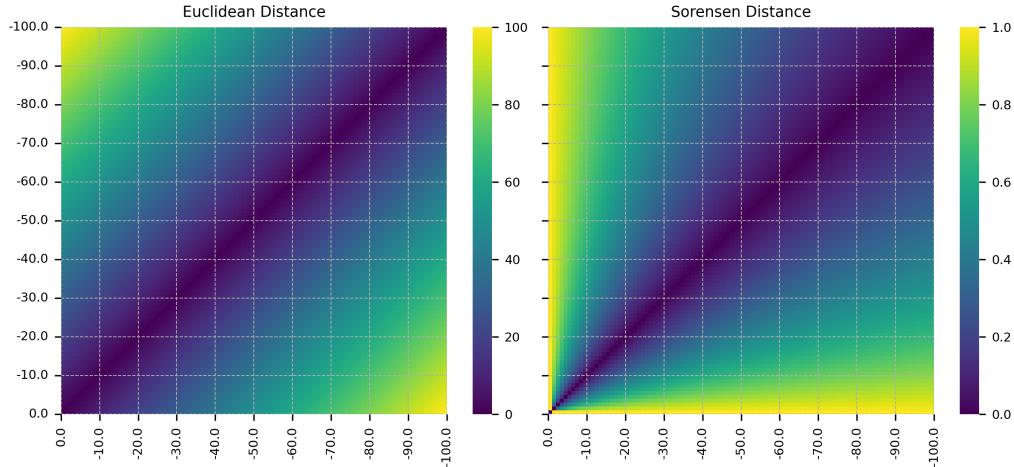


Abbildung 3.1: Vergleich der Distanzmetriken Euklidisch und Sørensen

### 3.1.2 Anzahl der Nachbarn

Der Parameter  $k$  legt fest, wie viele nächste Nachbarn für die Klassifizierung berücksichtigt werden. Kleine Werte für  $k$  können zu hoher Varianz und geringer Verzerrung führen, während größere Werte die Varianz verringern, jedoch die Verzerrung erhöhen. Die Wahl des optimalen  $k$ -Werts hängt stark von den vorhandenen Trainingsdaten ab. Bei Daten mit Ausreißern oder Rauschen empfiehlt es sich, höhere  $k$ -Werte zu wählen, da diese in solchen Fällen bessere Ergebnisse liefern. Um Gleichstände bei dem Mehrheitsvotum zu vermeiden, sollten ungerade Werte für  $k$  bevorzugt werden.<sup>6</sup>

Bei einem Wert von  $k = 3$  würden für beide Distanzen in dem Beispiel die Werte der vierten, fünften und sechsten Messung für die Vorhersage verwendet werden, da für diese drei Messungen die Distanzen am geringsten sind (siehe Tabelle 3.3) und das Modell würde Raum C vorhersagen, da dieser unter den drei Messungen am häufigsten vertreten ist.

### 3.1.3 Gewichtung

Der KNN-Algorithmus kann erweitert werden, indem die Distanzen der  $k$  nächsten Nachbarn nach ihrer Berechnung gewichtet werden. Hierfür stehen verschiedene Gewichtungsfunktionen,

<sup>6</sup> IBM (2024a)

wie zum Beispiel der Inversions-Kernel oder der Gauss-Kernel, zur Verfügung. Für Vorhersage unter Verwendung einer Gewichtungsfunktion, werden die Distanzen der  $k$  nächsten Nachbarn nach ihrer Berechnung in die Gewichtungsfunktion eingesetzt, für jede vertretene Klasse aufsummiert und es wird die Klasse mit dem größten Gewicht ausgewählt. Dadurch hat neben der relativen Häufigkeit der Klassen auch die Distanz der Nachbarn einen Einfluss auf die Klassifikation und es ist möglich eine nicht optimale Wahl des Parameters  $k$  auszugleichen.<sup>7</sup>

In dieser Arbeit wurde sich dafür entschieden den Inversions-Kernel zu verwenden, da für die Implementierung des KNN-Algorithmus die Bibliothek *scikit-learn* verwendet wird und der Inversions-Kernel der Standardgewichtungsfunktion dieser entspricht. Zudem hat die Wahl der Gewichtungsfunktion laut dem Paper *Weighted k-Nearest-Neighbor Techniques and Ordinal Classification* in der Regel keinen entscheidenden Einfluss auf die Genauigkeit der Klassifikation.<sup>8,9</sup>

Die Gewichtungsfunktion des Inversions-Kernels ist gegeben durch:

$$K(d) = \frac{1}{|d|} \quad (3.3)$$

wobei  $d$  die Distanz darstellt.

Raum	Gewicht (Euklidisch)	Gewicht (Sørensen)
C	0,671 (0,385 + 0,286)	116,667 (66,667 + 50)
B	0,222	33,333

Tabelle 3.4: Gewichtete Distanzen der drei nächstgelegenen Trainingsdaten

Unter Verwendung der Gewichtungsfunktion des Inversions-Kernels ergeben sich für das Beispiel die in Tabelle 3.4 dargestellten gewichteten Distanzen und das Modell würde Raum C vorhersagen.

Für die detaillierten Untersuchungen in Kapitel 5.3 wurde sich dafür entscheiden den KNN-Algorithmus ohne Gewichtungsfunktion (`weights = uniform`) und mit Gewichtungsfunktion (`weights = distance`) zu vergleichen, um zu untersuchen, ob die Gewichtungsfunktion die Genauigkeit der Klassifikation verbessern kann.

## 3.2 Support Vector Machines (SVM)

Der Support Vector Machine (SVM) Algorithmus ist ein maschinelles Lernalgorithmus, der für die Klassifizierung von Daten verwendet werden kann. Die Klassifizierung erfolgt, indem versucht wird die optimale Trennlinie – die sogenannte Hyperebene – zwischen den Datenpunkten zu finden. Die Hyperebene wird dabei so positioniert, dass der Abstand zwischen den Datenpunkten der beiden Klassen maximiert wird. In einem Beispiel mit zwei Merkmalen, wie in Abbildung 3.2, lassen sich die Datenpunkte beider Klassen in einem zweidimensionalen Koordinatensystem als Punkte darstellen und die Hyperebene als Linie.

---

<sup>7</sup> Hechenbichler und Schliep (2004), S. 6

<sup>8</sup> Hechenbichler und Schliep (2004), S. 6

<sup>9</sup> scikit-learn developers (2024a)

Der SVM Algorithmus ist auch in der Lage mit mehr als zwei Merkmalen zu arbeiten, indem versucht wird eine Hyperebene in einem mehrdimensionalen Raum, welche die Datenpunkte der verschiedenen Klassen voneinander trennt, zu finden.<sup>10</sup>

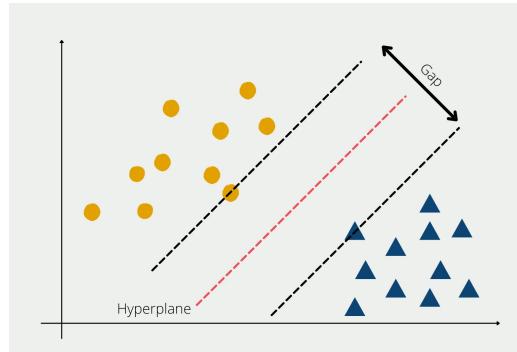


Abbildung 3.2: Darstellung einer Hyperebene in einem SVM mit zwei Features (Bildquelle: Basecamp 2021)

### 3.2.1 Auswahl des Kernels

In Abbildung 3.2 konnten die Daten durch eine Gerade getrennt werden, weswegen ein SVM mit einem linearen Kernel verwendet werden konnte. Sollten die Daten nicht linear trennbar sein - wie in Abbildung 3.3 - sollte ein anderer Kernel verwendet werden. In diesen Fällen wird zuerst die Kernel-Methode angewendet, welche die Daten in einen höherdimensionalen Raum transformiert, und anschließend wird versucht die Daten in diesem Raum linear zu trennen.<sup>11</sup>

In dieser Arbeit wurde sich dafür entschieden, den linearen Kernel mit einem nicht linearen Kernel zu vergleichen. Für das nicht lineare SVM wurde der RBF-Kernel gewählt, da in der Arbeit *Device-Free Presence Detection and Localization With SVM and CSI Fingerprinting* mit diesem Kernel die besten Ergebnisse unter den nicht linearen SVMs erzielt werden konnten.<sup>12</sup>

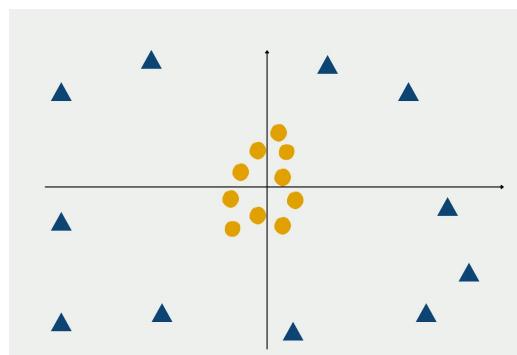


Abbildung 3.3: Darstellung von nicht linear trennbaren Daten in einem SVM (Bildquelle: Basecamp 2021)

---

<sup>10</sup> IBM (2024b)

<sup>11</sup> IBM (2024b)

<sup>12</sup> IBM (2024b)

### 3.2.2 Regularisierungsparameter C und Kernel-Parameter gamma

Der SVM Algorithmus kann über den Regularisierungsparameter C und den Kernel-Parameter `gamma` konfiguriert werden. Der Regularisierungsparameter C legt dabei fest, wie groß der Bereich ist, in dem die Hyperebene liegt. Also der Abstand zwischen den beiden schwarzen Linien in Abbildung 3.2. Je kleiner der Wert ist, desto größer ist der Abstand. Größere Werte für C können dazu führen, dass das Modell die Trainingsdaten gut klassifizieren kann, jedoch kann das auch dazu führen, dass das Modell fehleranfällig bei der Klassifikation neuer Daten ist.<sup>13</sup>

Der Kernel-Parameter `gamma` kann nur für nicht lineare SVM-Modelle verwendet werden und legt fest, wie groß der Einfluss eines einzelnen Trainingsbeispiels bei der Berechnung der Hyperebene ist. Bei einem großen Wert für `gamma` ist der Einfluss eines einzelnen Trainingsbeispiels gering und es kann zum Overfitting. Dies bedeutet das dass Modell die Trainingsdaten zu stark berücksichtigt und neue Daten nicht so gut klassifizieren kann. Ein kleiner Wert für `gamma` kann dazu führen, dass das Modell zu stark vereinfacht wird und es zum Underfitting kommt.<sup>14</sup>

## 3.3 Random Forest

Der Random Forest-Algorithmus ist Lernalgorithmus der zur Klassifizierung verwendet werden kann. Dabei basiert die Vorhersage auf einer Vielzahl von Entscheidungsbäumen, die jeweils eine eigene Vorhersage treffen. Ein Entscheidungsbau ist ein Modell, das Entscheidungen durch eine Abfolge von Wahl/Falsch-Abfragen trifft, die sich aus den Merkmalen der Daten ableiten. An jedem Knotenpunkt des Baumes wird eine Entscheidung basierend auf einem einzelnen Merkmal getroffen und am Ende eines jeden Entscheidungsbau erfolgt eine Vorhersage.

Für jeden Entscheidungsbau im Random Forest wird eine zufällige Teilmenge der Trainingsdaten verwendet und die endgültige Vorhersage des Random Forests Modells ergibt sich aus einem Mehrheitsvotum der individuellen Entscheidungsbäume.<sup>15</sup>

Laut der Studie *WiFi Indoor Localization with CSI Fingerprinting-Based Random Forest* sind die einflussreichsten Parameter des Random Forest Algorithmus die maximale Tiefe der Bäume (`max_depth`), die Anzahl der Entscheidungsbäume (`n_estimators`) sowie die maximale Anzahl der betrachteten Features (`max_features`). Aus diesem Grund werden in dieser Arbeit diese Parameter in Bezug auf die Genauigkeit bei der Vorhersage untersucht.<sup>16</sup>

In Abbildung 3.4 ist ein Entscheidungsbau dargestellt, welcher mit Hilfe der Beispieldaten aus Tabelle 3.1 erstellt wurde und für die Daten aus Tabelle 3.2 Raum C vorhersagen würde.

---

<sup>13</sup> Helmy (2024)

<sup>14</sup> GeeksforGeeks (2024)

<sup>15</sup> Donges (2024)

<sup>16</sup> Wang u. a. (2018), S. 18

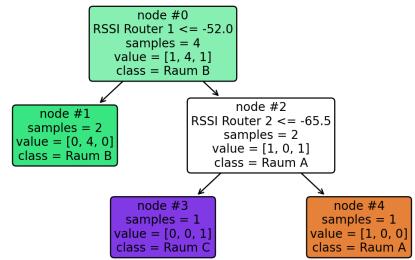


Abbildung 3.4: Darstellung eines Entscheidungsbaums anhand der Beispieldaten

# 4 Systemarchitektur und Implementierung

Die implementierte Software lässt sich in vier Komponenten unterteilen: Die Android App *BVG Detection*, welche bereits in einer vorherigen Arbeit entwickelt wurde<sup>1</sup>, einem Server der eine API bereitstellt, welche die in der Offline-Phase gesammelten Daten speichern kann und für WiFi-Fingerprints von unbekannten Räumen eine Vorhersage treffen kann, einer Datenbank zur Verwaltung der Fingerprints und dem Quellcode der Microcontroller.

## 4.1 Datenbankstruktur

Für die Speicherung der gesammelten WiFi-Fingerprints während der Offline-Phase wurde eine Datenbankstruktur entwickelt, welche auf einer *MariaDB*-Instanz auf dem Server und in einer *SQLite*-Datenbank innerhalb der App *BVG Detection* implementiert wurde. Die Datenbank besteht aus den vier Tabellen `rooms`, `measurements`, `routers` und `measurement_router`.

### 4.1.1 Tabelle rooms

In der Tabelle `rooms` werden die Informationen der Räume, in denen die Messungen durchgeführt wurden, gespeichert. Dazu gehören eine eindeutige Raum-ID, der Name des Raumes, eine Beschreibung, Koordinaten und der Pfad zu einer Bilddatei. Die Beschreibung, die Koordinaten und der Pfad der Bilddatei sind dabei nur für die *BVG Detection*-App relevant.

Spalte	Datentyp	Beschreibung
room_id	INT	Primärschlüssel der Tabelle
room_name	VARCHAR(255)	Eindeutiger Name des Raumes
description	VARCHAR(255)	Beschreibung des Raumes
coordinates	VARCHAR(255)	Geografische Koordinaten des Raumes
picture_path	VARCHAR(255)	Pfad zu einer Bilddatei des Raumes

Tabelle 4.1: Struktur der Tabelle `rooms`.

### 4.1.2 Tabelle measurements

Die Tabelle `measurements` dient der Erfassung aller Messungen. Jede Messung wird durch eine eindeutige Kombination aus Gerät-ID und Zeitstempel identifiziert, um doppelte Einträge zu vermeiden. Das ist wichtig, da in einer Datenbank auch die Messungen mehrerer Geräte vorhanden sein können und diese unterschieden werden müssen. Zudem wird die Raum-ID gespeichert, die auf die Tabelle `rooms` verweist, wodurch die Messung einem Raum zugeordnet werden kann.

---

<sup>1</sup> Winter (2017), S. 11 ff.

Spalte	Datentyp	Beschreibung
measurement_id	INT	Primärschlüssel der Tabelle
timestamp	TIMESTAMP	Zeitpunkt der Messung
device_id	VARCHAR(255)	ID des Gerätes, das die Messung durchgeführt hat
room_id	INT	Verweis auf den Raum, in dem die Messung stattfand

Tabelle 4.2: Struktur der Tabelle `measurements`.

#### 4.1.3 Tabelle routers

In der Tabelle `routers` werden die Informationen der erfassten Access Points gespeichert. Die Einträge der Spalte *BSSID* werden dazu verwendet, um zu überprüfen, ob ein Access Point bereits in der Tabelle gespeichert ist, da die *BSSID* jeden Access Point eindeutig kennzeichnet.

Spalte	Datentyp	Beschreibung
router_id	INT	Primärschlüssel der Tabelle
ssid	VARCHAR(255)	Name des WiFi-Netzwerks (SSID)
bssid	VARCHAR(255)	MAC-Adresse des Routers (BSSID)

Tabelle 4.3: Struktur der Tabelle `routers`.

#### 4.1.4 Tabelle measurement\_router

Die Tabelle `measurement_router` speichert die empfangenen Signalstärken der Access Points aller Messungen und verknüpft diese über den Eintrag `measurement_id` mit den Einträgen aus der Tabelle `measurements`.

Spalte	Datentyp	Beschreibung
measurement_id	INT	Verweis auf eine Messung
router_id	INT	Verweis auf einen Router
signal_strength	INT	Empfangene Signalstärke bei der Messung

Tabelle 4.4: Struktur der Tabelle `measurement_router`.

## 4.2 Implementierung des Servers

Der Server läuft auf einer virtuellen Maschine der HTW Berlin unter der Linux Distribution *Debian*, ist mit 2 CPU-Kernen, 3 GB RAM und einem Speicherplatz von 16 GB konfiguriert und ist über die IP-Adresse 141.45.212.246 aus dem Netzwerk der HTW erreichbar. Auf diesem Server sind vier zentrale Anwendungen in separaten Docker-Containern implementiert: eine *FastAPI*-Anwendung, *Prometheus*, *Grafana* und *MariaDB*. Diese Anwendungen können mithilfe von *Docker Compose* über den Befehl `sudo docker-compose build` installiert werden und über `sudo docker-compose up -d` und `sudo docker-compose down` gestartet bzw. beendet werden. Das bietet den Vorteil, dass alle Anwendungen ohne großen Aufwand installiert werden können und die Abhängigkeiten der Anwendungen korrekt konfiguriert sind. Die Implementierung des Servers basiert auf der Anleitung von *Zoo Codes*.<sup>2</sup>

<sup>2</sup> Codes (2023)

#### 4.2.1 FastAPI

Die API zur Verwaltung von WiFi-Fingerprints und zur Vorhersage von Räumen basierend auf WiFi-Fingerprints von nicht bekannten Räumen wurde mithilfe des Python-Frameworks *FastAPI* implementiert. *FastAPI* bietet den Vorteil, dass automatisch eine interaktive API-Dokumentation (aufrufbar unter <http://141.45.212.246:8000/docs>) bereitgestellt wird (siehe Abbildung 4.1), die es ermöglicht, die Endpunkte der API zu testen.



Abbildung 4.1: FastAPI Dokumentation

Die API stellt verschiedene Routen zur Verfügung, welche in Tabelle 4.5 aufgelistet sind.

Route	Methode	Beschreibung
/health	GET	Prüft den Status der API.
/measurements/add	POST	Fügt eine neue Messung hinzu.
/measurements/predict	POST	Sagt den Raum basierend auf Wi-Fi-Daten voraus.
/measurements/all	GET	Ruft alle gespeicherten Messungen ab.
/measurements/{measurement_id}	GET	Ruft eine spezifische Messung nach ID ab.
/rooms/all	GET	Ruft alle gespeicherten Räume ab.
/rooms/{room_id}	GET	Ruft einen spezifischen Raum nach ID ab.
/routers/all	GET	Ruft alle gespeicherten Router ab.
/routers/{router_id}	GET	Ruft einen spezifischen Router nach ID ab.
/reset-database	POST	Setzt die Datenbank zurück.

Tabelle 4.5: Übersicht der Endpunkte der *FastAPI*-Anwendung.

## Route: /measurements/add

Zum Hinzufügen eines neuen WiFi-Fingerprints, der während der Offline-Phase gesammelt wurde, kann die Route `/measurements/add` verwendet werden. Dabei wird im ersten Schritt überprüft, ob es bereits einen Eintrag in der Tabelle `measurements` gibt, bei dem die Einträge `device_id` und `timestamp` mit den übergebenen Werten übereinstimmen. Falls ein solcher Eintrag bereits existiert, wird die Messung nicht hinzugefügt. Dadurch kann sichergestellt werden, dass keine doppelten Einträge in der Datenbank vorhanden sind. Sollte die Messung noch nicht in der Datenbank vorhanden sein, wird für diese Messung ein neuer Eintrag in der Tabelle `measurements` hinzugefügt und für jeden empfangenen Router ein Eintrag in der Tabelle `measurement_router` erstellt.

In dem Code Beispiel 4.1 ist eine Beispielanfrage an die Route `/measurements/add` dargestellt, bei der ein neuer WiFi-Fingerprint für den Raum `WH_C_625` hinzugefügt werden soll.

```
1 {
2     "room_name": "WH_C_625",
3     "device_id": "google_pixel_8",
4     "timestamp": 1721479799,
5     "routers": [
6         {
7             "ssid": "eduroam",
8             "bssid": "dc:b8:08:c9:73:02",
9             "signal_strength": -73
10        }
11    ]
12 }
```

Listing 4.1: Beispiel für eine Anfrage an `/measurements/predict`

## Route: /measurements/predict

Die POST Route `/measurements/predict` wird verwendet um für einen WiFi-Fingerprint eine Raumvorhersage zu treffen. Dafür wurden die in Kapitel 3 beschriebenen Algorithmen KNN, SVM und Random Forest mit den dazugehörigen Parametern implementiert. Für die Implementierung der Algorithmen wurde die Python Bibliothek `scikit-learn` verwendet.

Der Endpunkt erwartet unter dem Parameter `routers` eine Liste von Router Objekten, welche jeweils die Einträge `ssid`, `bssid` und `signal_strength` beinhalten. Zusätzlich kann über den optionalen Parameter `algorithm` der Algorithmus zur Vorhersage festgelegt werden (die möglichen Werte hierbei sind `knn_euclidean`, `knn_sorensen`, `random_forest`, `svm_rbf` und `svm_linear`). Die Algorithmus spezifischen Parameter können über die Parameter `k_value`, `weights`, `n_estimators`, `c_value`, `gamma_value`, `max_depth` und `max_features` gesetzt werden und entsprechen den Parametern der `scikit-learn` Bibliothek. Über den Parameter `ignore_measurements` können *IDs* von Messungen angegeben werden, die bei der Vorhersage ignoriert werden sollen.

Für die Verwendung der in Kapitel 5.4 untersuchten Datenaufbereitungsmethoden können die Parameter `use_remove_unreceived_bssids`, `handle_missing_values_strategy`, `router_selection`, `router_presence_threshold`, `value_scaling_strategy` und `router_rssi_threshold` gesetzt werden.

In dem Code Beispiel 4.2 ist eine Beispielanfrage an die Route `/measurements/predict` dargestellt, bei der für einen WiFi-Fingerprint bestehend aus zwei Access Points eine Raumvorhersage getroffen werden soll unter Verwendung des KNN-Algorithmus mit dem Wert 5 für k und ohne Gewichtungsfunktion (`weights = uniform`).

```

1 {
2     "routers": [
3         {
4             "ssid": "eduroam",
5             "bssid": "dc:b8:08:c9:54:a1",
6             "signal_strength": -88
7         },
8         {
9             "ssid": "Gast@HTW",
10            "bssid": "dc:b8:08:c9:54:a2",
11            "signal_strength": -85
12        }
13    ],
14    "algorithm": "knn_euclidean",
15    "k_value": 5,
16    "weights": "uniform"
17 }
```

Listing 4.2: Beispiel für eine Anfrage an `/measurements/predict`

Als Antwort liefert die API ein *JSON*-Objekt, das den vorhergesagten Raum zurückgibt.

```

1 {
2     "room_name": "WH_C_625",
3 }
```

Listing 4.3: Beispiel einer API-Antwort von `/measurements/predict`

#### 4.2.2 Grafana und Prometheus

Neben der Datenbank und der *FastAPI*-Anwendung wurden auch die Monitoring-Tools **Prometheus** und **Grafana** auf dem Server installiert, um die Leistung der Anwendungen zu überwachen und die Daten in Echtzeit zu visualisieren. Dabei dient Prometheus zur Speicherung der Metriken von der *FastAPI* und *Grafana* zur Visualisierung. Für die Überwachung der *FastAPI* wurde ein Dashboard implementiert, welches die CPU- und RAM-Auslastung, die Anfragen pro Minute nach *HTTP*-Statuscode und die durchschnittliche Antwortzeit für jeden Endpunkt anzeigt (siehe Abbildung 4.2).

Für die Visualisierung der *MariaDB* wurde ebenfalls ein *Grafana*-Dashboard erstellt, welches eine Übersicht der Datenbank, die Inhalte der Tabellen und die Zeitpunkte darstellt, an denen die Messungen durchgeführt wurden (siehe Abbildung 4.3). Die Dashboards können aus dem Netz der HTW über Adresse <http://141.45.212.246:3000> aufgerufen werden.

### 4.3 Implementierung der *BVG Detection* App

Die Android App *BVG Detection* (<https://github.com/OpenHistoricalDataMap/BVGDetection>) kann unter anderem dazu verwendet werden WiFi-Fingerprints von Orten zu speichern und die eigene Position anhand der bekannten WiFi-Fingerprints zu bestimmen.

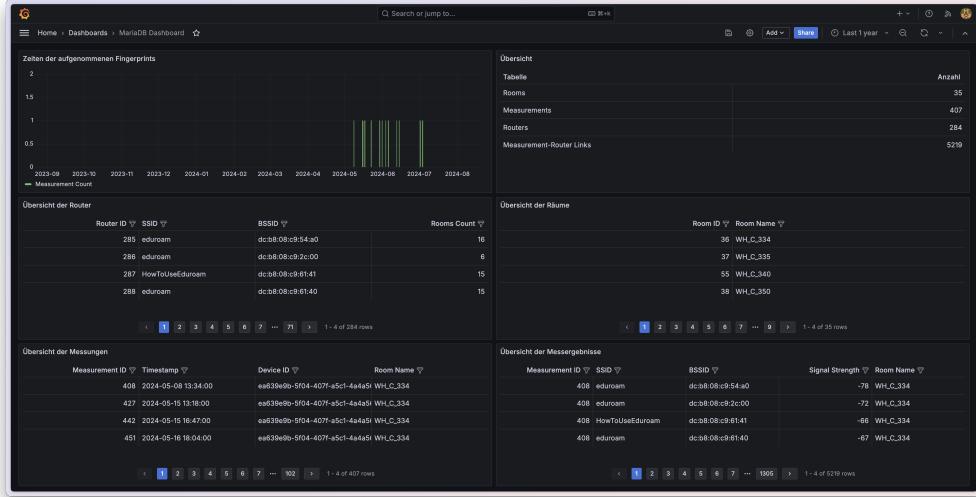


Abbildung 4.2: Grafana Dashboard der *FastAPI*



Abbildung 4.3: Grafana Dashboard der *MariaDB*

Die bisherige Implementierung wurde um die Funktionalität erweitert mehr als einen Fingerprint pro Raum aufzunehmen und die in der Offline Phase gemessenen Fingerprints mit anderen Geräten und der API auszutauschen. Zudem wurden die untersuchten Algorithmen aus dieser Arbeit für die Raumvorhersage implementiert.

### 4.3.1 Wiederinbetriebnahme der App

In der bisherigen Implementierung konnte die App zum Zeitpunkt dieser Arbeit nicht mehr kompiliert werden. Dies lag daran, dass das verwendete Build-Tool *Gradle* veraltet war und in der aktuellen Version von *Android Studio (Koala / 2024.1.1)* nicht mehr unterstützt wurde. Damit die App wieder kompiliert werden konnte, wurde *Gradle* von der Version 2.3.3 auf die Version 8.3.2 aktualisiert.<sup>3</sup>

Zudem wurden die verwendeten *Android Support Libraries*, welche veraltet sind und für die UI-Komponenten und die Abwärtskompatibilität zuständig sind, durch *AndroidX* ersetzt.<sup>4</sup>

<sup>3</sup> Google (2024a)

<sup>4</sup> Google (2024b)

Durch diese Änderungen kann die *BVG Detection*-App wieder kompiliert werden und wird auf allen Android Geräten mit Android 9.0 (Pie) oder höher unterstützt.

### 4.3.2 Datenspeicherung und Aufnahme der WiFi-Fingerprints

In der bisherigen Implementierung war bereits eine *SQLite*-Datenbank vorhanden. Diese wurde durch die in Kapitel 4.1 erläuterte Datenbank ersetzt. Über den Menüpunkt *Ort/Fingerprint aufnehmen* (siehe Abbildung 4.4) kann ein neuer Raum inklusive Fingerprint hinzugefügt werden bzw. für einen bereits vorhandenen Raum ein neuer Fingerprint aufgenommen werden. Bei der Aufnahme eines neuen Fingerprints muss der Raumname angegeben werden, damit der Fingerprint diesem Raum zugeordnet werden kann. Sollte es bereits eine Messung von diesem Raum geben, kann dieser aus einem Dropdown Menü ausgewählt werden. Optional kann auch eine Beschreibung des Raums sowie ein Foto des Raums hinzugefügt werden. Für die Aufnahme der Fingerprints nutzt die App den Android *WiFiManager*. Seit der Android Version 9 ist die Anzahl der WiFi-Scans auf vier Scans innerhalb von 2 Minuten limitiert. Aus diesem Grund wird bei der Aufnahme eines neuen Fingerprints alle 30 Sekunden, sofern mehrere Fingerprints aufgenommen werden sollen, ein Scan durchgeführt.<sup>5</sup>

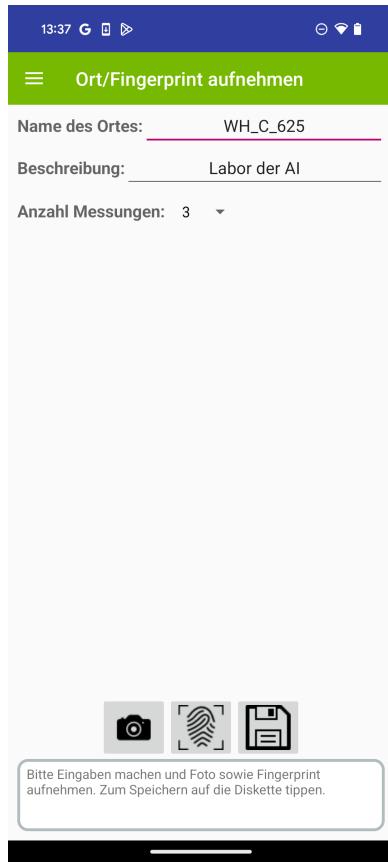


Abbildung 4.4: Menüpunkt *Ort/Fingerprint aufnehmen* der *BVG Detection*-App

Unter dem Menüpunkt *Fingerprints verwalten* (siehe Abbildung 4.5a) können alle Räume inklusive Messungen der Messungen (siehe Abbildung 4.5b) und den darin enthaltenen Access Points (siehe Abbildung 4.5) angezeigt werden.

Die kompilierte Version der App ist in dem *GitLab*-Repository dieser Arbeit in dem Unterordner

<sup>5</sup> Google (2024c)

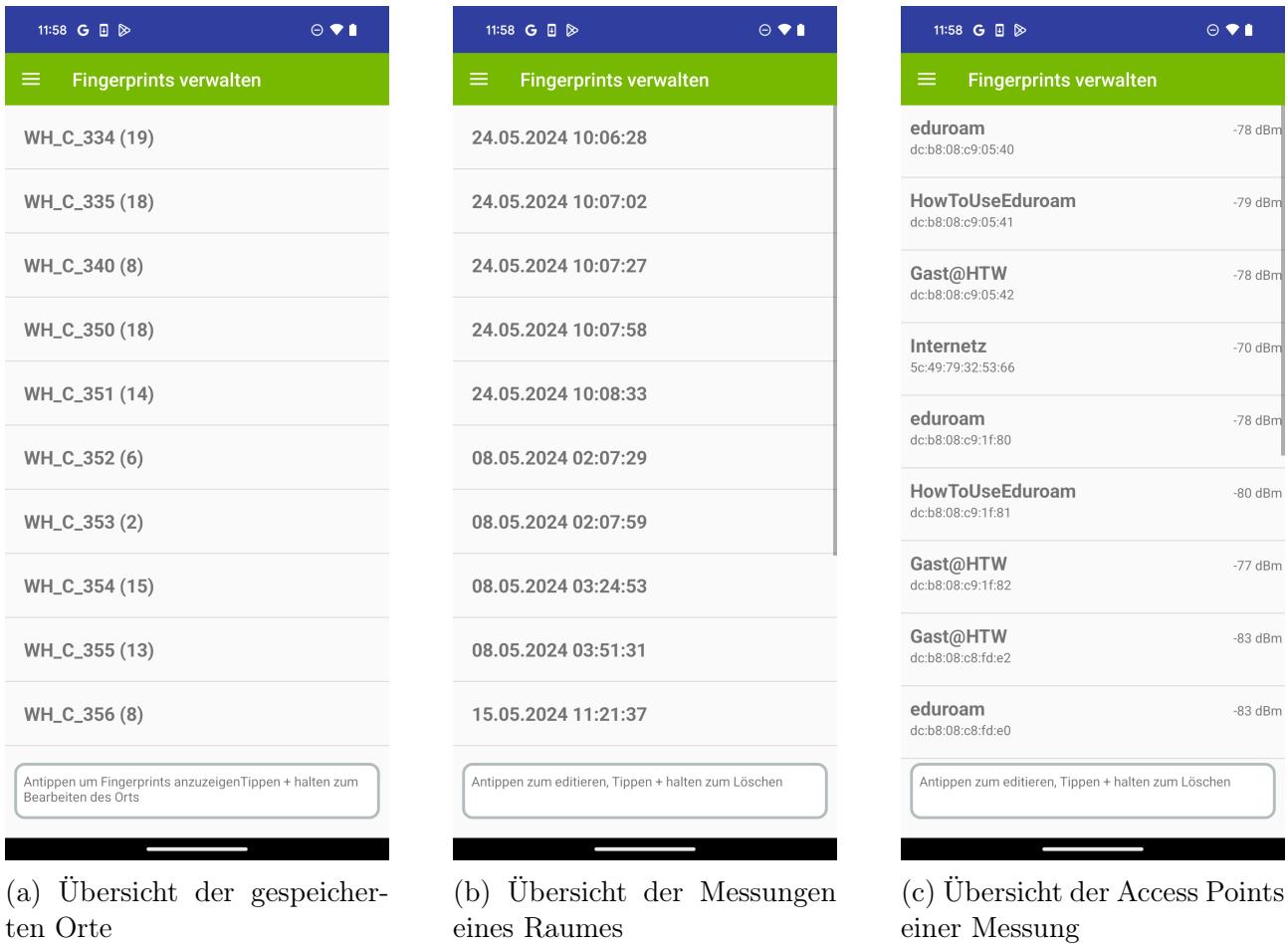


Abbildung 4.5: Menüpunkt *Fingerprints verwalten* der *BVG Detection*-App

`bvg-detection-app` zu finden und in dem *GitHub*-Repository der *BVG Detection App* (<https://github.com/OpenHistoricalDataMap/BVGDetection>) auf dem Branch v2.

### 4.3.3 Austausch der Daten

Für den Austausch der Daten wurde der Menüpunkt *Datenbank teilen* (siehe Abbildung 4.6) hinzugefügt. Unter diesem Menüpunkt können die gespeicherter Fingerprints per Bluetooth an andere Geräte gesendet werden bzw. von anderen Geräten empfangen werden und an die API aus Kapitel 4.2 gesendet bzw. von dieser empfangen werden.

Damit die Daten über Bluetooth ausgetauscht werden können, müssen beide Geräte eine Bluetooth-Verbindung eingehen. Dazu muss eins der beiden Geräte einen `BluetoothSocket` und das andere Gerät einen `BluetoothServerSocket` öffnen. Dies geschieht über die beiden Buttons *Listen* und *List Devices*. Im Anschluss werden alle sichtabren Geräte in einer Liste angezeigt und das entsprechende Gerät kann ausgewählt werden. Nachdem beide Geräte eine Bluetooth Verbindung aufgebaut haben, wird das in der App in dem Statustextfeld angezeigt und über den Button *Send* kann der Transfer der Daten gestartet werden.

Die zu übertragenden Messdaten werden aus der *SQLite*-Datenbank des sendenden Geräts in einem *JSON*-Array gesammelt. Anschließend wird dieses Array durchlaufen und jeder Eintrag in ein Byte-Array umgewandelt, welches dann über die Bluetooth Verbindung an den Empfänger

gesendet werden kann.

Um die Struktur der gesendeten Daten zu kennzeichnen und die Nachrichten korrekt zu trennen, wird zwischen den einzelnen Messungen das Trennzeichen (`MESSAGE_SPLIT = “_NEW_MESSAGE_”`) eingefügt und am Ende der gesamten Übertragung wird das Endzeichen (`MESSAGE_END = “_MESSAGE_END_”`) gesendet. Dadurch wird der Empfänger darüber informiert, dass die Übertragung abgeschlossen ist. Der Empfänger sammelt die eingehenden Daten, bis das Endzeichen erkannt wird. Anschließend werden die gesammelten Daten wieder in ein *JSON*-Array konvertiert und in der *SQLite*-Datenbank des Empfängers gespeichert.

Für den Austausch der Daten mit der API aus Kapitel 4.2 muss sich das Smartphone in dem Netzwerk der HTW Berlin befinden. Wenn das der Fall ist kann über den Button *Send to API* die Übertragung der Messungen an die API gestartet werden. Über den Button *Receive from API* werden alle Fingerprints die in der *MariaDB* auf dem Server gespeichert sind empfangen und lokal gespeichert werden

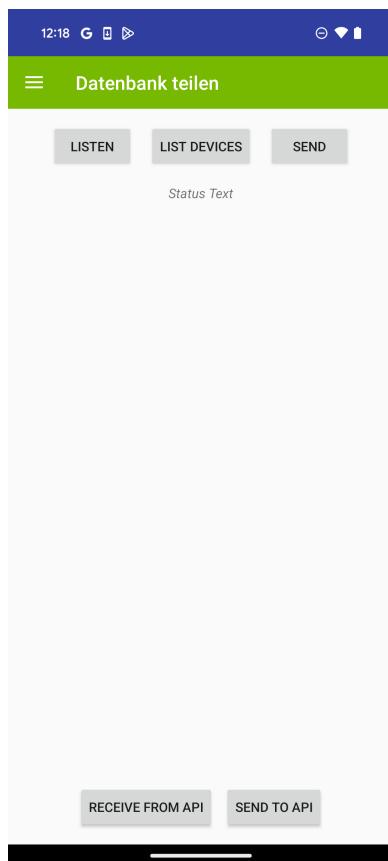


Abbildung 4.6: Menüpunkt *Datenbank teilen* der *BVG Detection*-App

Da die API auf einem HTTP-Server läuft und kein SSL Zertifikat hat, muss der Zugriff explizit erlaubt werden, indem die IP-Adresse des Servers in der Netzwerkkonfiguration der App eingetragen wird. Sollte sich die IP-Adresse des Servers ändern, muss diese manuell im Quellcode der App geändert werden.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <network-security-config>
3     <domain-config cleartextTrafficPermitted="true">
4         <domain includeSubdomains="true">141.45.212.246</domain>
5     </domain-config>
6 </network-security-config>
```

Listing 4.4: Netzwerkkonfiguration der App

#### 4.3.4 Ortung von Räumen

Die Ortung kann über den Menüpunkt *Standort ermitteln* (siehe Abbildung 4.7a) gestartet werden und wird lokal auf dem Gerät ausgeführt. Dazu wurden die in dieser Arbeit untersuchten Algorithmen (KNN, SVM und Random Forest), die dazugehörigen Parameter und die Datenaufbereitungsschritte (siehe Kapitel 5.4) implementiert. Aus dem Grund, dass für die Algorithmen in der API die Python Bibliothek *scikit-learn* verwendet wurde und diese nicht unter Android verwendet werden kann, wurden die Algorithmen in Java nachimplementiert.

Der KNN und Random Forest Algorithmus wurden nach den Vorlagen von *Kenzo Takahashi* und *Florian Zyprian*, in welchen die Algorithmen von *scikit-learn* nachimplementiert wurden, implementiert<sup>6,7</sup>. Für die Implementierung des SVM Algorithmus wurde die Bibliothek *AndroidLibSvm* (<https://github.com/yctung/AndroidLibSVM>), welche eine Open-Source Implementierung der *LIBSVM* Bibliothek (<https://www.csie.ntu.edu.tw/~cjlin/libsvm/>) ist, verwendet.

Die Algorithmen inklusive Parameter und die verschiedenen Methoden zur Aufbereitung der Daten können in den Einstellungen (siehe Abbildungen 4.7b und 4.7c) ausgewählt werden und werden bei der Ortung (siehe Abbildung 4.7a) angezeigt.

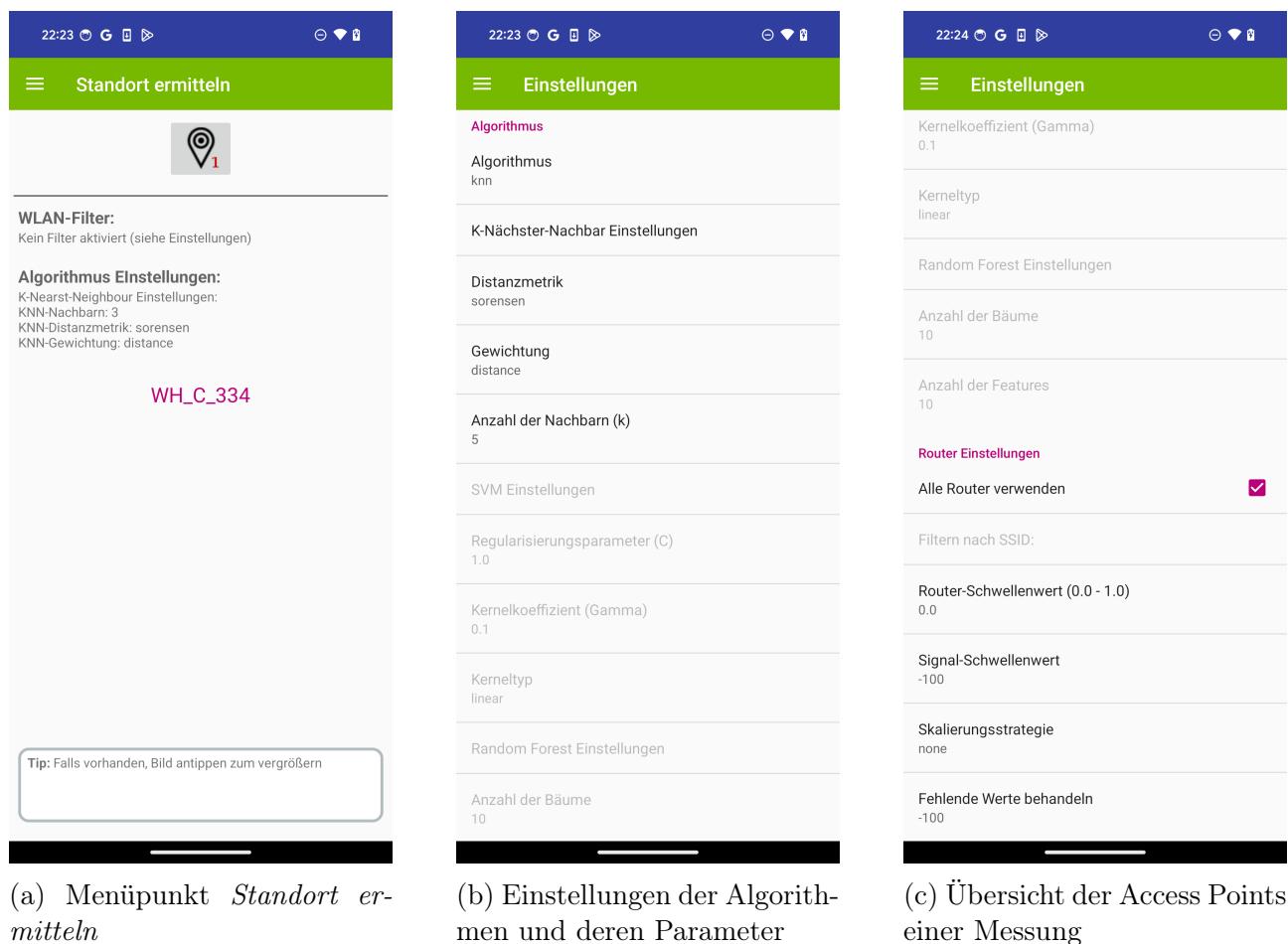


Abbildung 4.7: Einstellungen der Datenaufbereitungsmethoden

<sup>6</sup> Takahashi (2016)

<sup>7</sup> Zyprian (2023)

## 4.4 Quellcode der Microcontroller

Damit Microcontroller wie zum Beispiel der *ESP32* seine Position bestimmen kann, muss dieser einen WiFi-Scan durchführen und die Daten an die API aus Kapitel 4.2 senden. Dafür wurde ein *MicroPython*-Script geschrieben, dass in dem Code Beispiel 4.5 dargestellt ist.

```
1 import network
2 import urequests
3 import utime
4
5 # Initialize the Wi-Fi interface globally
6 wlan = network.WLAN(network.STA_IF)
7
8 # Function to connect to a Wi-Fi network
9 def connect_to_wifi(ssid, password):
10     global wlan
11     wlan.active(True)
12
13     if not wlan.isconnected():
14         print(f"Connecting to {ssid}...")
15         wlan.connect(ssid, password)
16
17         while not wlan.isconnected():
18             print("Attempting to establish connection...")
19             utime.sleep(1)
20
21     print("Connected to Wi-Fi!")
22
23 # Function to scan available Wi-Fi networks and send the data to an API
24 def scan_and_send(api_route):
25     global wlan
26
27     # Scan for available networks
28     networks = wlan.scan()
29
30     # Prepare the data structure for the API
31     routers = []
32     for network in networks:
33         bssid = ":".join("{:02x}".format(byte) for byte in network[1])
34         signal_strength = network[3]
35         ssid = network[0].decode('utf-8')
36
37         if len(ssid) > 0:
38             routers.append({
39                 "ssid": ssid,
40                 "bssid": bssid,
41                 "signal_strength": signal_strength
42             })
43
44     data = {
45         "routers": routers
46     }
47
48     # Send the data via a POST request to the API
49     try:
50         response = urequests.post(api_route, json=data)
51         if response.status_code == 200:
52             response_json = response.json()
53             print(f"Room Prediction: {response_json['room_name']}")
```

```

54     else:
55         print(f"Error in room prediction: {response.status_code} - {
56             response.text}")
56     except Exception as e:
57         print(f"Error sending data: {e}")
58
59 # Main function
60 def main(ssid, password, api_route):
61     connect_to_wifi(ssid, password)
62
63     # Perform the Wi-Fi scan and send the data to the API
64     scan_and_send(api_route)
65
66 if __name__ == '__main__':
67     # Define SSID, password, and API route here
68     ssid = 'Rechnernetze'
69     password = ''
70     api_route = 'http://141.45.212.246:8000/measurements/predict',
71
72     main(ssid, password, api_route)

```

Listing 4.5: *MicroPython*-Quellcode für die Durchführung eines WiFi-Scans und die Raumbestimmung über die API

# 5 Datenerhebung und Analyse der Algorithmen

## 5.1 Aufnahme von WiFi-Fingerprints für die Analyse der Algorithmen

Für die Auswertung der Algorithmen und deren Parameter wurden 407 Messungen in 35 Räumen auf dem Campus Wilhelminenhof der Hochschule für Technik und Wirtschaft Berlin (HTW Berlin) in Gebäude C durchgeführt. Die Messungen fanden in dem Zeitraum vom 08. Mai 2024 bis zum 03. Juli 2024 statt und es wurden insgesamt 263 Router erfasst. Für die Aufnahme der WiFi-Fingerprints wurden ein Doogee Y8 und ein Google Pixel 8 Smartphone verwendet.

Die Messungen erfolgten dabei randomisiert, abhängig davon, welche Räume zu dem Zeitpunkt frei waren, und wurden hauptsächlich während der Pausenzeiten und am Abend durchgeführt, da zu diesen Zeiten die meisten Räume zur Verfügung standen. Dabei wurde bewusst darauf verzichtet, den genauen Standort innerhalb der Räume zu planen, um realistische Einsatzbedingungen widerzuspiegeln.

Eine Übersicht der Messungen pro Raum ist in Abbildung 5.1 dargestellt.

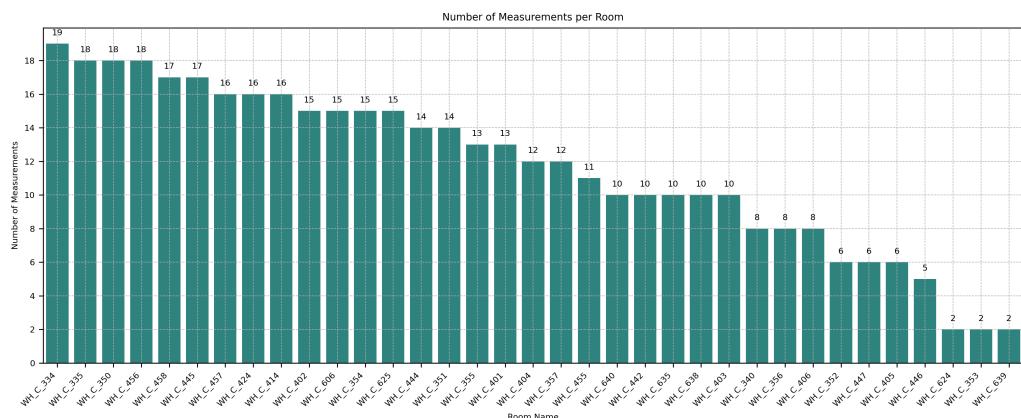


Abbildung 5.1: Anzahl der Messungen pro Raum

## 5.2 Beschreibung der Testanwendung

Für die Untersuchung der Modelle und deren Parameter wurde ein Anwendung in Python entwickelt, welche die verschiedenen Parameter der Algorithmen testet, indem alle möglichen Kombinationen der Parameter an die API gesendet werden und für jede Messung die Vorhersage gespeichert wird. Die Anwendung ist über eine *YAML*-Datei (siehe Code-Beispiel 5.1), in der die grundlegenden Einstellungen, wie die Endpunkte der API, die Anzahl der zu verarbeitenden Messungen und die zu betrachtenden Räume und Flure, definiert werden, konfigurierbar.

Zudem können unter dem Parameter *parameter\_sets* die verschiedenen Untersuchungsreihen definiert werden. Jeder Eintrag in *parameter\_sets* entspricht einem Testlauf, dessen Ergebnisse unter dem angegebenen Namen in einer *CSV*-Datei gespeichert werden. Innerhalb jedes

Eintrags werden die zu testenden Parameter und deren Werte in einem Array angegeben.

In dem Code-Beispiel 5.1 sind zwei Testreihen definiert: *01\_knn\_weights* und *08\_router\_presence\_threshold*. In der ersten Testreihe werden für den KNN-Algorithmus mit der euklidischen und Sørensen Distanzmetrik und den Werten 5, 7 und 9 für den Parameter *k\_value* die Gewichtungsfunktionen *distance* und *uniform* getestet. In der zweiten Testreihe wurden für den Parameter *router\_presence\_threshold* die Werte 0, 0.25, 0.5 und 0.75 getestet und für die Algorithmen KNN mit der euklidischen Distanz und den Werten 5, 7 und 9 für den Parameter *k\_value* und SVM mit dem RBF-Kernel und den Werten 5.0, 1.0 und 0.5 für den Parameter *c\_value* untersucht.

Der Ablauf des Programms ist, dass zunächst die *YAML*-Datei eingelesen, und über alle in den *parameter\_sets* definierten Einträge iteriert wird. Für jeden Eintrag werden alle möglichen Kombinationen der angegebenen Parameter gebildet. Im Fall des zweiten Beispiels ergibt das 24 mögliche Kombinationen, da zwei Algorithmen mit jeweils drei unterschiedlichen Parametern und vier verschiedenen Werten für *router\_presence\_threshold* getestet werden.

Anschließend durchläuft das Programm alle definierten Messungen (in diesem Beispiel 10) und sendet für jede dieser Messungen jede mögliche Parameterkombination über eine *HTTP POST*-Anfrage an die API. Dabei wird auch die ID der jeweiligen Messung über den Parameter *ignore\_measurements* mitgesendet, wodurch sichergestellt wird, dass diese Messung bei der Vorhersage nicht beachtet wird. Dies verhindert, dass die API eine Messung in der Datenbank findet, die exakt dem übermittelten Werten entspricht. Die Resultate der API-Anfragen werden nach dem Durchlauf aller Messungen in einer CSV-Datei, die nach dem Namen des jeweiligen *parameter\_set*-Eintrags benannt ist, gespeichert.

```
1 url_fetch: "http://141.45.212.246:8000/measurements/all"
2 url_predict: "http://141.45.212.246:8000/measurements/predict"
3
4 num_measurements: 10
5 rooms: ["WH_C_351", "WH_C_352", "WH_C_353", "WH_C_335"]
6 corridors: ["WH_C_35_corridor"]
7
8 parameter_sets:
9   - name: "01_knn_weights"
10     parameters:
11       weights: ["distance", "uniform"]
12       algorithm:
13         knn_euclidean:
14           k_value: [5, 7, 9]
15         knn_sorenson:
16           k_value: [5, 7, 9]
17   - name: "08_router_presence_threshold"
18     parameters:
19       router_presence_threshold: [0, 0.25, 0.5, 0.75]
20       algorithm:
21         knn_euclidean:
22           k_value: [5, 7, 9]
23         svm_rbf:
24           c_value: [5.0, 1.0, 0.5]
```

Listing 5.1: *YAML*-Konfigurationsdatei der Testanwendung

## 5.3 Analyse der Algorithmen und Parameter

In dem folgenden Kapitel werden die verschiedenen Algorithmen und deren Parameter miteinander verglichen. Dafür wurde mithilfe der in Kapitel 5.2 vorgestellten Anwendung für jede Untersuchung ein Eintrag in der `YAML`-Konfigurationsdatei erstellt und es wurde für alle untersuchten Parameter und jede der 407 Messungen eine Vorhersage getroffen.

### 5.3.1 Voruntersuchung der nicht im Detail betrachteten Parameter

Da in dieser Arbeit für jeden Algorithmus ein Parameter im Detail untersucht wird und für jeden Algorithmus mehrere Parameter existieren, werden im ersten Schritt die Parameter untersucht, die nicht im Detail betrachtet werden. Das sind bei dem KNN-Modell der `weights` Parameter, der die Gewichtungsfunktion angibt, bei dem SVM-Modell mit dem RBF-Kernel der `gamma` Parameter und bei dem Random Forest Modell sind es die Parameter `max_depth` und `n_estimators`.

In Abbildung 5.2 sind die Ergebnisse des gewichteten und des ungewichteten KNN-Algorithmus in Abhängigkeit der Anzahl an Messungen pro Raum dargestellt. Wie zu erkennen ist, steigt die Genauigkeit mit der Anzahl an Messungen pro Raum, wobei der Unterschied zwischen `weights = distance` und `weights = uniform` mit zunehmender Anzahl an Messungen abnimmt und mit `weights = distance` bessere Ergebnisse erzielt werden konnten, wenn nur wenige Messungen vorhanden sind. Aus diesem Grund wurde sich dafür entschieden für die weiteren Untersuchungen den `weights` Parameter auf `distance` zu setzen.

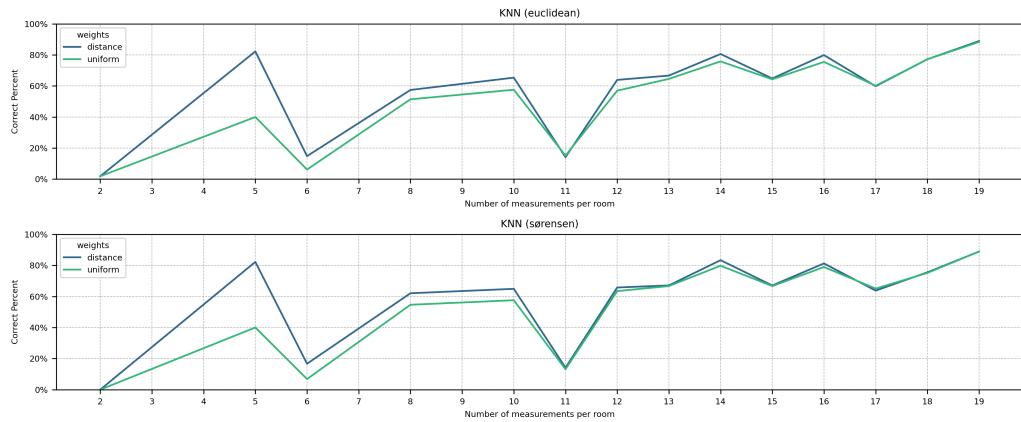


Abbildung 5.2: Vergleich des gewichteten (`weights = distance`) und ungewichteten KNN-Algorithmus (`weights = uniform`)

Bei dem SVM Algorithmus wurde der Parameter `gamma` untersucht, welcher nur für den RBF-Kernel gesetzt werden kann. Dabei wurde die Standardeinstellung von `scikit-learn`, welche `scale` ist, mit der Einstellung `auto` verglichen. Bei `scale` wird `gamma` auf

$$\gamma = \frac{1}{n_{\text{features}} \cdot \text{Var}(X)} \quad (5.1)$$

gesetzt und bei `auto` auf

$$\gamma = \frac{1}{n_{\text{features}}}. \quad (5.2)$$

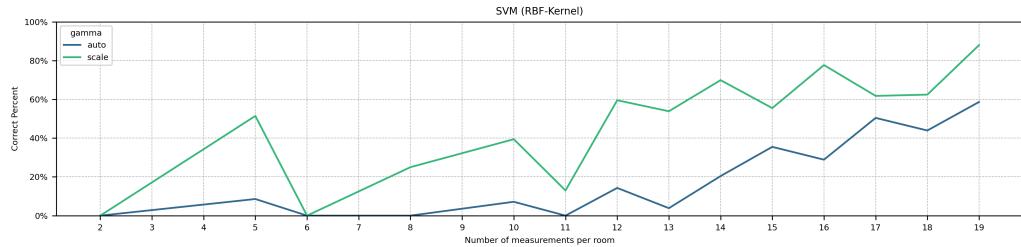


Abbildung 5.3: Vergleich der Parameter `auto` und `scale` des `gamma`-Parameters für den SVM-Algorithmus mit RBF-Kernel

In Abbildung 5.3 sind die Ergebnisse der beiden `gamma` Parameter in Abhängigkeit der Anzahl der Messungen pro Raum dargestellt. Wie zu erkennen ist, ist die Genauigkeit bei `scale` besser als bei `auto`. Aus diesem Grund wurde sich dafür entschieden für die weiteren Untersuchungen den `gamma` Parameter auf `scale` zu setzen.

Bei Random Forest werden die beiden Parameter `max_depth` (maximale Tiefe der Entscheidungsbäume) und `n_estimators` (Anzahl der Entscheidungsbäume) untersucht, da diese laut dem Paper *WiFi Indoor Localization with CSI Fingerprinting-Based Random Forest* neben dem Parameter `max_features` (maximale Anzahl der verwendeten Features) den zweit- und drittgrößten Einfluss auf die Genauigkeit haben.<sup>1</sup>

Bei der Untersuchung der maximalen Tiefe der Entscheidungsbäume wurden für den Parameter `max_depth` die Werte 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21 und `None` verglichen. Grundlage für diese Entscheidung ist das Paper *WiFi Indoor Localization with CSI Fingerprinting-Based Random Forest*, welches zu dem Ergebnis kam, dass die Genauigkeit der Ergebnisse ab einer bestimmten Tiefe der Entscheidungsbäume stagniert. Der Parameter `None` wurde zudem verwendet, da dies die Standardimplementierung von `scikit-learn` ist.<sup>2,3</sup>

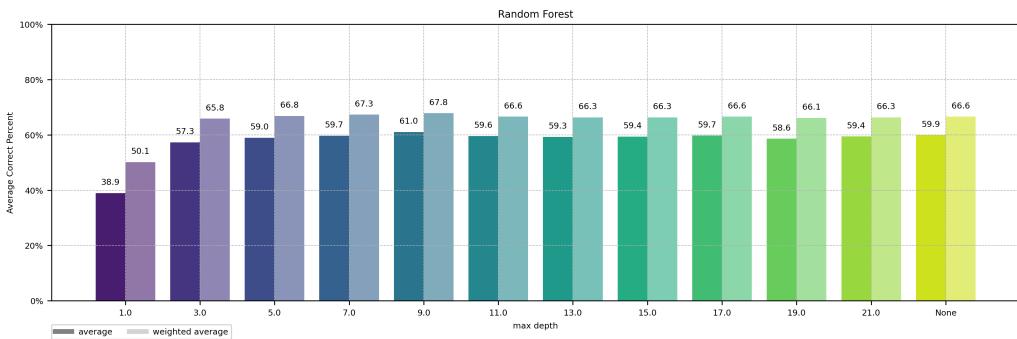


Abbildung 5.4: Vergleich der verschiedenen Werte des `max_depth` Parameters für den Random Forest Algorithmus

Wie in Abbildung 5.4 zu erkennen ist, hat die Wahl des Parameters nur einen geringen Einfluss auf die Genauigkeit der Ergebnisse, mit Ausnahme der Fälle, bei denen `max_depth` auf 1 gesetzt

<sup>1</sup> Wang u. a. (2018), S. 18

<sup>2</sup> Wang u. a. (2018), S. 19

<sup>3</sup> scikit-learn developers (2024b)

ist. Bei allen anderen Werten sind die Genauigkeitsunterschiede minimal. Bei den ungewichteten Ergebnissen beträgt die maximale Abweichung 3,7 %, während sie bei den gewichteten Ergebnissen bei 2 % liegt.

Da die Genauigkeit ab einem bestimmten Wert des `max_depth` Parameters keine signifikanten Unterschiede mehr aufweist und die besten Ergebnisse bei einer Tiefe von 9 erzielt werden konnte, wurde sich dafür entschieden für die weiteren Untersuchungen diesen Parameter auf 9 zu setzen.

Bei der Auswahl der Anzahl der Bäume (`n_estimators`) wurden die Werte in dem Wertebereich von 10 bis 100 in 10er-Schritten und die Werte in dem Wertebereich zwischen 100 und 1000 in 100er-Schritten untersucht. Diese Auswahl basiert auf den Erkenntnissen des Papers *WiFi Indoor Localization with CSI Fingerprinting-Based Random Forest*, welches zeigt, dass die Genauigkeit mit zunehmender Anzahl an Bäumen zwar steigt, die Rechenzeit jedoch ebenfalls linear mit der Anzahl der Bäume wächst. Das Paper stellt außerdem fest, dass ab einer bestimmten Anzahl an Bäumen keine signifikanten Verbesserungen der Genauigkeit mehr erzielt werden und dass die Unterschiede bei einer geringen Anzahl von Bäumen größer sind.<sup>4</sup>

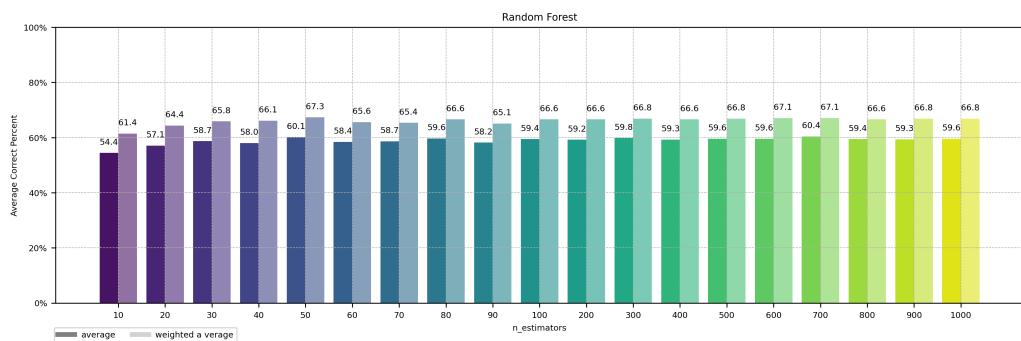


Abbildung 5.5: Vergleich der verschiedenen Werte des `n_estimators` Parameters für den Random Forest Algorithmus

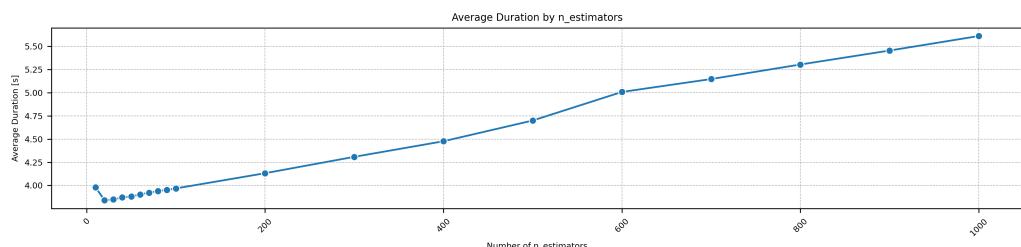


Abbildung 5.6: Vergleich der Dauer des Random Forest Algorithmus in Abhängigkeit der Anzahl der Bäume

Wie in Abbildung 5.5 zu erkennen ist, hat die Anzahl der Bäume, abgesehen von den kleineren Werten 10 und 20, keinen signifikanten Einfluss auf die Genauigkeit der Ergebnisse. Selbst bei den kleineren Werten ist der Unterschied der Genauigkeit nur minimal, wobei eine Steigerung der Anzahl der Bäume tendenziell zu einer höheren Genauigkeit führt. Die besten Ergebnisse wurden mit 50 Bäumen (60,1 % ungewichteter Durchschnitt und 67,3 % gewichteter Durchschnitt) und 700 Bäumen (60,4 % ungewichteter Durchschnitt und 67,1 % gewichteter Durchschnitt) erzielt.

<sup>4</sup> Wang u. a. (2018), S. 19

In Übereinstimmung mit den Ergebnissen des zitierten Papers, zeigt Abbildung 5.6, dass die Rechenzeit linear mit der Anzahl der Bäume ansteigt. Aufgrund dieser Tatsache und der Tatsache dass die Unterschiede in der Genauigkeit zwischen 50 und 700 Bäumen gering sind, wurde sich dafür entschieden, für die weiteren Untersuchungen die Anzahl der Bäume auf 50 zu setzen.

### 5.3.2 Untersuchung der im Detail betrachteten Parameter

Für die detaillierten Untersuchungen der Parameter wurden für jeden Algorithmus die drei Parameter ermittelt, die in den meisten Fällen eine richtige Vorhersage getroffen haben. Dafür wurden für den KNN Algorithmus der Parameter  $k$  untersucht, für den Random Forest Algorithmus die Parameter `max_features` und für den SVM Algorithmus der Parameter  $C$ .

Bei dem KNN Algorithmus mit der Sörensen und euklidischen Distanz wurden jeweils die Parameterwerte 1, 3, 5, 7, 9, 11, 13, 15 getestet, da empfohlen wird für  $k$  ungerade Werte zu verwenden. Für den SVM Algorithmus wurden für den linearen- und den RBF-Kernel die Parameterwerte 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5 für  $C$  verwendet, da diese in dem Paper *Supervised Learning-Based Indoor Positioning System Using WiFi Fingerprints* empfohlen wurden.<sup>5</sup> Für den Random Forest Algorithmus wurden die Parameterwerte *sqrt*, *log2*, *None*, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 verglichen, da für den Parameter `max_features` in der *scikit-learn* Implementierung des Random Forest Algorithmus die Anzahl der Features als absoluter Wert (Parameterwert 1 bis 10) oder als relativer Wert in Abhängigkeit der Gesamtzahl an Features angegeben werden kann und beides untersucht werden soll. Die Werte 0,1 bis 0,9 entsprechen dabei der Prozentszahl der betrachteten Features und *sqrt* und *log2* entsprechen der Quadratwurzel bzw. dem Logarithmus zur Basis 2 der Gesamtanzahl an Features.

In Abbildung 5.7 sind die Ergebnisse der verschiedenen Parameter für den KNN und SVM Algorithmus in Abhängigkeit von der Anzahl der Messungen pro Raum dargestellt. Es ist ersichtlich, dass alle Algorithmen bei 5 Messungen pro Raum verhältnismäßig gute Ergebnisse erzielen konnten, während hingegen bei 11 Messungen pro Raum vergleichsweise schlechtere Ergebnisse auftraten. Dies könnte auf einzelne Ausreißer oder Messfehler zurückzuführen sein. Ebenso ist es möglich, dass die Räume mit 5 Messungen unter besonders günstigen Bedingungen erfasst wurden. Nimmt man diese Ausreißer heraus, zeigt sich bei allen Algorithmen die Tendenz, dass die Genauigkeit der korrekten Raumvorhersage mit einer zunehmenden Anzahl an Messungen pro Raum steigt.

In Abbildung 5.8 sind die durchschnittlichen Genauigkeiten der Parameter dargestellt. Dabei wurde einmal der Durchschnitt aller Ergebnisse berechnet (*average*) und einmal der gewichtete Durchschnitt (*weighted average*), bei dem für jede Anzahl an Messungen pro Raum der Durchschnitt gebildet wurde und anschließend der Gesamtdurchschnitt. Dies soll einer möglichen Verzerrung entgegenwirken, da so Räume mit mehr Messungen nicht stärker ins Gewicht fallen.

---

<sup>5</sup> Suleiman u. a. (2023), S. 62

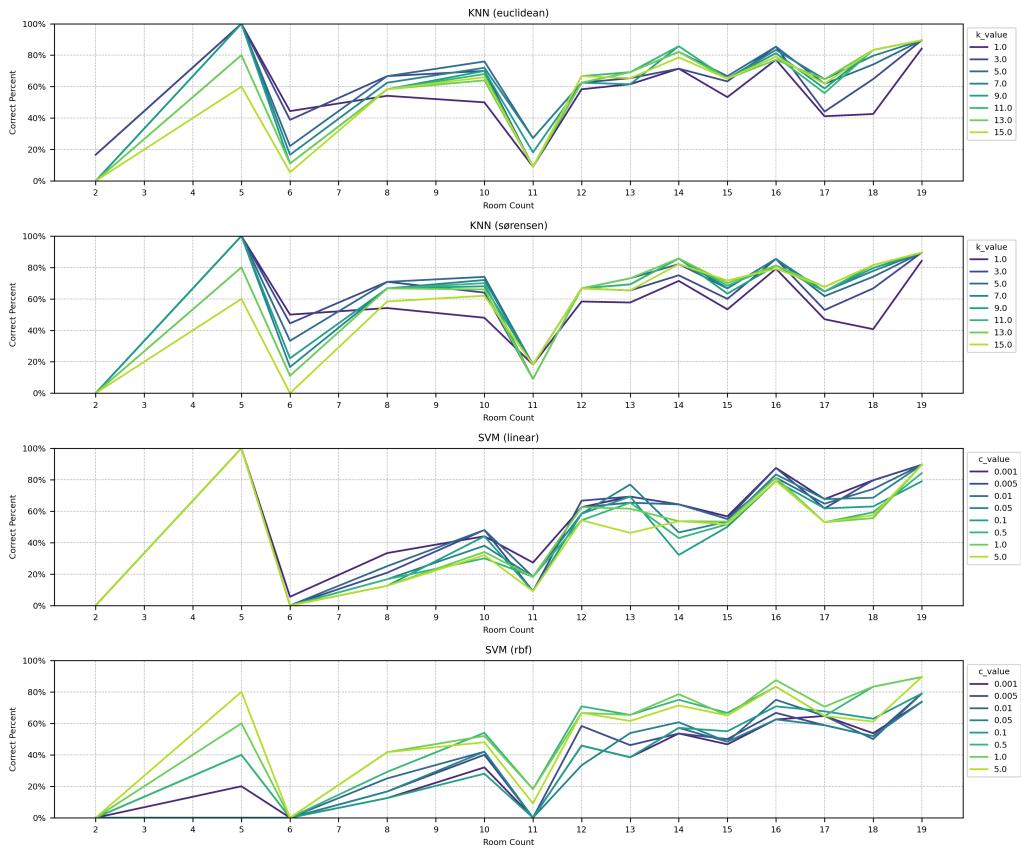


Abbildung 5.7: Vergleich der Parameter in Abhangigkeit der Anzahl der Messungen

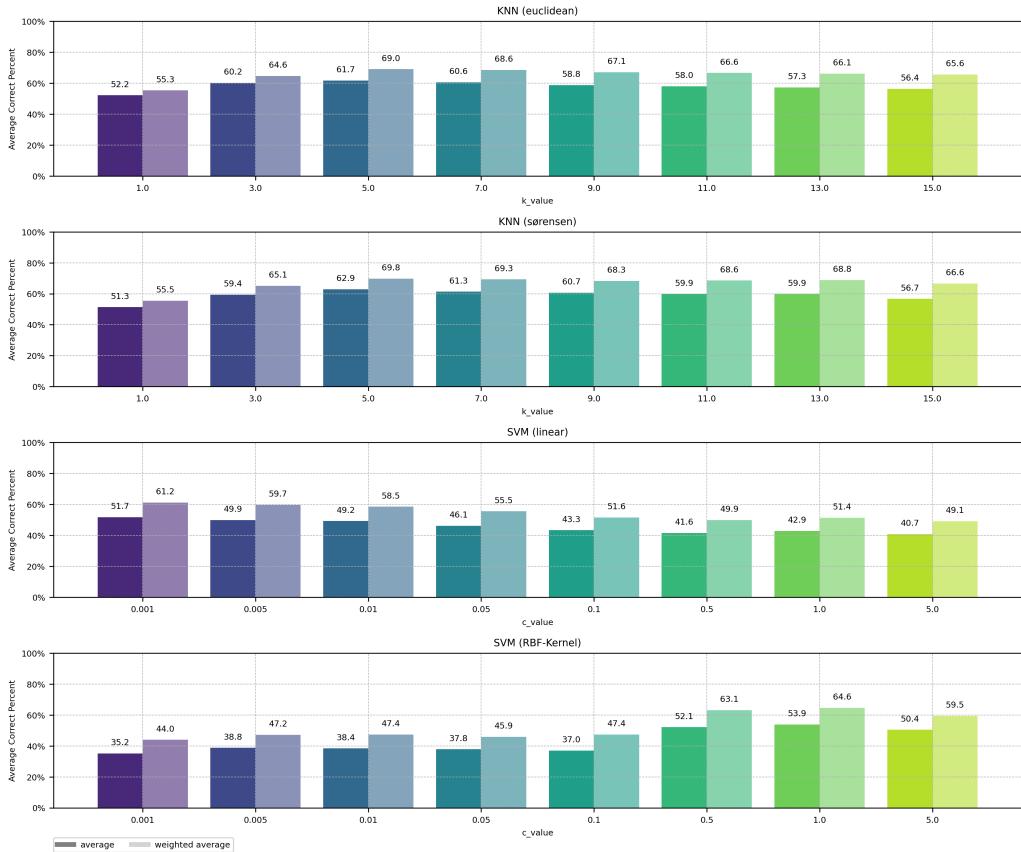


Abbildung 5.8: Vergleich der durchschnittlichen Genauigkeit der Parameter in Abhangigkeit der Anzahl der Messungen

**KNN** Wie in Abbildung 5.7 zu sehen ist, erzielen kleinere Werte für  $k$  sowohl bei der euklidischen als auch bei der Sørensen-Distanzmetrik bessere Ergebnisse in Räumen mit weniger Messungen, während größere Werte für  $k$  hingegen bei Räumen mit mehr Messungen von Vorteil sind. Im Durchschnitt konnten die besten Ergebnisse bei  $k = 5$ ,  $k = 7$  und  $k = 9$  erzielt werden. Daher wurden diese Werte für die weiteren Analysen ausgewählt.

**SVM (linear)** Abbildung 5.8 zeigt, dass die Genauigkeit des SVM-Algorithmus mit linearem Kernel mit zunehmendem Wert für  $C$  abnimmt, wobei die besten Ergebnisse bei  $C = 0,001$  erzielt wurden. Diese Ergebnisse gelten sowohl für den gewichteten (61,2 %) als auch für den ungewichteten Durchschnitt (51,7 %). Die zweit- und drittbesten Ergebnisse wurden für  $C = 0,005$  und  $C = 0,01$  erzielt. Daher wurden für die weitere Analyse die Werte  $C = 0,001$ ,  $C = 0,005$  und  $C = 0,01$  ausgewählt.

**SVM (RBF)** Für den SVM Algorithmus mit dem RBF-Kernel wurden die besten Ergebnisse bei  $C = 1,0$  erzielt (64,6 % für den gewichteten und 53,9 % für den ungewichteten Durchschnitt). Die zweit- und drittbesten Ergebnisse wurden bei  $C = 0,5$  und  $C = 5,0$  erreicht. Daher wurden für die weitere Analyse die Werte  $C = 0,5$ ,  $C = 1,0$  und  $C = 5,0$  ausgewählt.

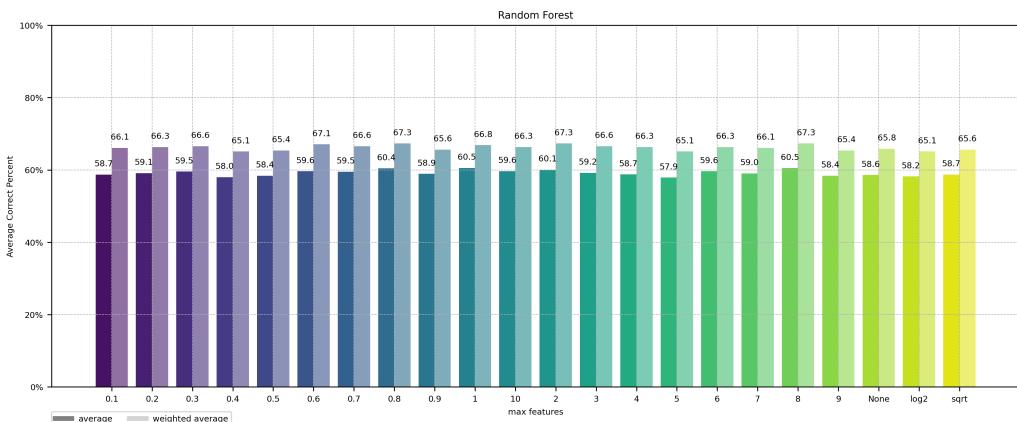


Abbildung 5.9: Vergleich der Parameter des Random Forest Algorithmus in Abhängigkeit der Anzahl der Messungen

In Abbildung 5.9 sind die durchschnittlichen Genauigkeiten der Parameter für den Random Forest Algorithmus dargestellt. Wie zu erkennen ist, hat die Wahl der Anzahl an Features keinen signifikanten Einfluss auf die Genauigkeit der Raumvorhersagen. Der Mittelwert gewichteten Durchschnitt liegt dabei bei 66,19 % und bei den ungewichteten Durchschnitten bei 59,15 %. Die besten Ergebnisse - auch wenn nur mit geringem Abstand - konnten für die Werte `max_features = 0,8` (entspricht 80 % der Features), `max_features = 2` und `max_features = 8` erzielt werden. Aus diesem Grund wurden für die weiteren Untersuchungen die Werte `max_features = 0,8`, `max_features = 2` und `max_features = 8` ausgewählt.

## 5.4 Untersuchungen des Einflusses verschiedener Daten-aufbereitungsmethoden

In dem folgenden Kapitel werden verschiedene Strategien zur Aufbereitung der Daten untersucht. Dafür wurde jede dieser Strategien in der API-Route `/measurements/predict` implementiert und kann über die Parameter dieses Endpunktes ausgewählt werden. Für die Analyse dieser Strategien wurde die in Kapitel 5.2 vorgestellte Anwendung verwendet.

### Strategien zum Umgang mit fehlenden Werten

Im ersten Schritt der Datenaufbereitung wurde untersucht, wie sich verschiedene Strategien zum Umgang mit fehlenden Werten auf die Genauigkeit der Algorithmen auswirken.

Für die Erstellung der KNN-, SVM- und Random-Forest-Modelle wurden die Trainingsdaten in Form einer Matrix übergeben, in der für jeden Raum und jeden Router die RSSI-Werte aufgeführt sind. Da jedoch in der Realität nicht in jedem Raum dieselben Router empfangen werden und die Modelle nicht mit fehlenden Werten umgehen können, ist eine Aufbereitung dieser Daten erforderlich. Hierfür wurden verschiedene Strategien entwickelt:

1. **Strategie -100:** Fehlenden Werten wird der Wert -100 dBm zugewiesen. Dieser Wert wurde gewählt, da das der kleinstmögliche RSSI-Wert ist (siehe Kapitel 2.2.3). Diese Strategie simuliert, dass der Router zwar erfasst wurde, aber nur ein sehr schwaches Signal gesendet hat.
2. **Strategie use\_received:** Fehlende Werte werden durch den entsprechenden Wert in den Testdaten ersetzt. Diese Methode wurde gewählt, weil bei nicht vorhandenen Routern die Distanz null ist und diese somit beim KNN-Algorithmus ignoriert werden.
3. **Strategie zero:** Fehlende Werte werden durch null ersetzt.

In Abbildung 5.10 ist die Genauigkeit der Algorithmen in Abhängigkeit von der gewählten Strategie zum Umgang mit fehlenden Werten dargestellt. Es zeigt sich, dass die zweite Strategie bei KNN und Random Forest zu keiner Verbesserung der Ergebnisse führt und auch bei zunehmender Anzahl an Messungen keine Tendenz zu besseren Ergebnissen erkennbar ist. Auffällig bei diesen beiden Modellen ist, dass bei der `use_received` Strategie eine leichte Steigerung der Genauigkeit bei 11 Messungen pro Raum zu beobachten ist, während die anderen beiden Strategien bei dieser Anzahl an Messungen eine entgegengesetzte Entwicklung zeigen. Die `use_received` Strategie konnte bei 11 Messungen pro Raum zum Teil bessere Ergebnisse erzielen als die anderen beiden Strategien. Im direkten Vergleich der ersten und dritten Strategie für KNN und Random Forest ist zu erkennen, dass die Zuweisung von -100 dBm bei fehlenden Werten zu besseren Ergebnissen führt als die Ersetzung durch 0 dBm, auch wenn die Unterschiede gering sind.

Bei dem SVM Algorithmus zeigt sich im Gegensatz zu den anderen beiden Modellen, dass die Genauigkeit bei der zweiten Strategie mit zunehmender Anzahl an Messungen tendenziell steigt. Dennoch sind auch hier die anderen beiden Strategien überlegen.

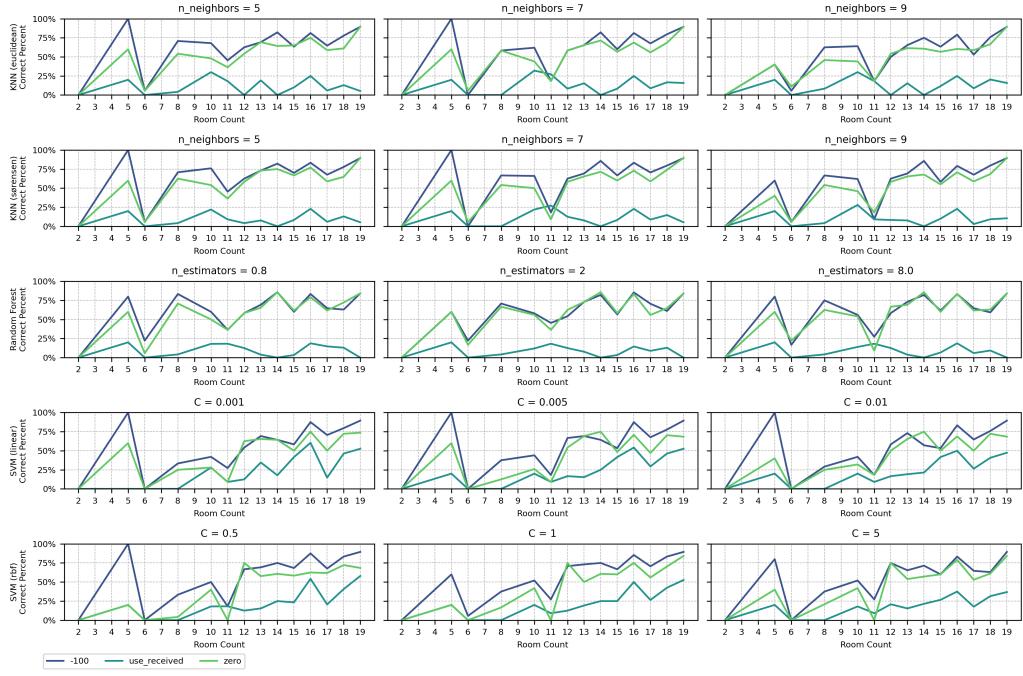


Abbildung 5.10: Vergleich der Genauigkeit in Abhängigkeit der Strategie zum Umgang mit fehlenden Werten

Wie in Abbildung 5.11 zu erkennen ist, führt die erste Strategie in allen Fällen zu den besten Ergebnissen, wenn die durchschnittliche Genauigkeit betrachtet wird. Aus diesem Grund werden in den weiteren Untersuchungen fehlende Werte durch -100 dBm ersetzt.

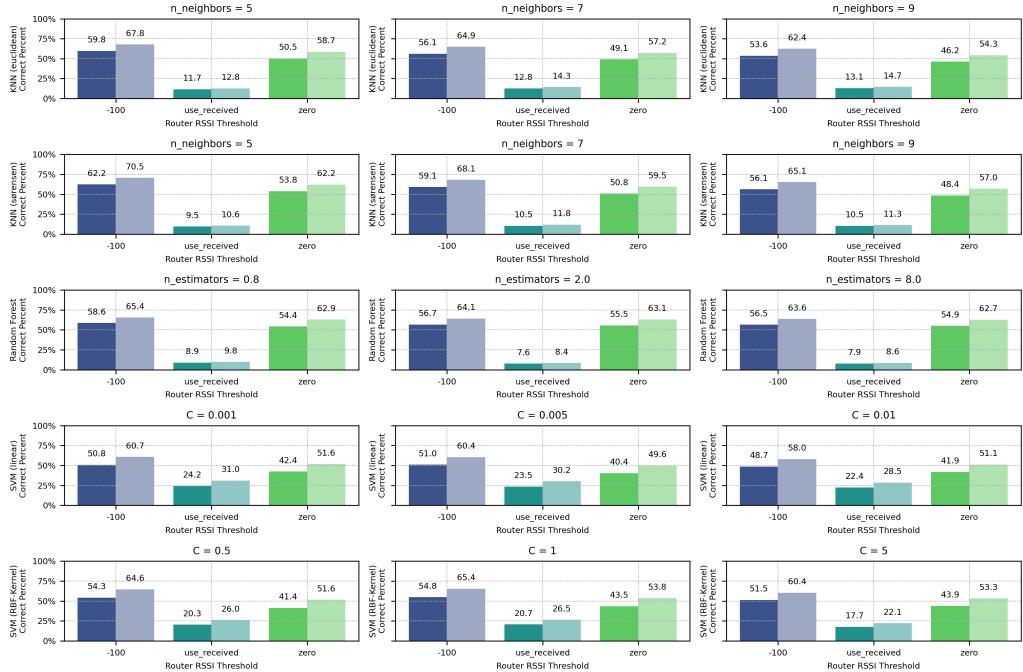


Abbildung 5.11: Vergleich der durchschnittlichen Genauigkeit in Abhängigkeit der Strategie zum Umgang mit fehlenden Werten

## Einfluss der verwendeten Router

Im zweiten Schritt der Datenaufbereitung wird untersucht, ob die Auswahl der Router einen Einfluss auf die Genauigkeit der Modelle hat. Der Gedanke hinter diesem Aufbereitungsschritt ist, dass nicht alle empfangenen Access Points ortsgebunden sind (z. B. Hotspots von mobilen Endgeräten) und somit einen negativen Einfluss auf die korrekte Raumvorhersage haben können. Deswegen könnte es von Vorteil sein, wenn nur die Access Points der eduroam-Infrastruktur (*eduroam*, *HowToUseEduroam* und *Gast@HTW*) für die Vorhersagen verwendet werden.

Aus diesem Grund wird in diesem Aufbereitungsschritt untersucht, wie sich die Genauigkeit der Modelle verhält, wenn nur die Router der *eduroam*-Infrastruktur für die Vorhersagen verwendet werden.

Wie in Abbildung 5.12 zu erkennen ist, sind die Ergebnisse für alle Modelle und alle Parameterkonfigurationen - mit Ausnahme von den Ergebnissen bei 5 Messungen pro Raum - fast identisch. Dies deutet darauf hin, dass die Auswahl der Router bei diesen Daten keinen signifikanten Einfluss auf die Genauigkeit der Modelle hat und dass in dem Raum in dem 5 Messungen durchgeführt wurden die nicht-*eduroam* Access Points einen besonders großen Einfluss haben bzw. die *eduroam* Access-Points nicht aussagekräftig genug sind. Für die weiteren Untersuchungen wurde sich daher dafür entschieden nur die *eduroam* Router zu betrachten, da bei diesen Routern davon ausgegangen werden kann, dass diese ihre Position nicht verändern und die Unterschiede - abgesehen von dem Raum mit 5 Messungen - minimal sind.

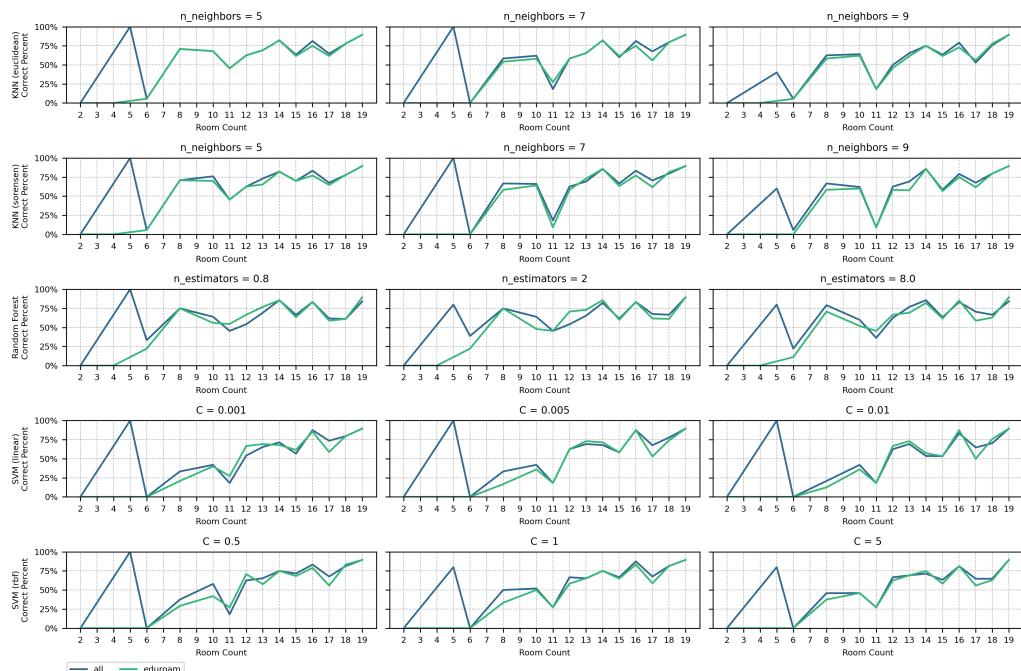


Abbildung 5.12: Vergleich der Genauigkeit in Abhängigkeit der Strategie zur Auswahl der Router

## Filterung von RSSI-Werten

### Berücksichtigung nur häufig auftretender Router

Im dritten Schritt der Datenaufbereitung wird analysiert, ob die Genauigkeit verbessert werden kann, wenn nur Access Points berücksichtigt werden, die in mehreren Messungen eines Raums erfasst wurden. Die zugrunde liegende Annahme ist, dass Router, die häufiger in einem Raum empfangen werden, repräsentativer sind als solche, die nur selten erfasst werden. Aus diesem Grund wurden verschiedene Schwellenwerte getestet, die den Prozentsatz der berücksichtigten Router angeben. Dabei wurde unter anderem die bisherige Einstellung, bei der alle Router einbezogen werden, sowie die Schwellenwerte von 25 %, 50 % und 75 % untersucht. Das bedeutet, dass beispielsweise nur die Router berücksichtigt werden, die in mindestens 25 % der Messungen eines Raums vorhanden sind.

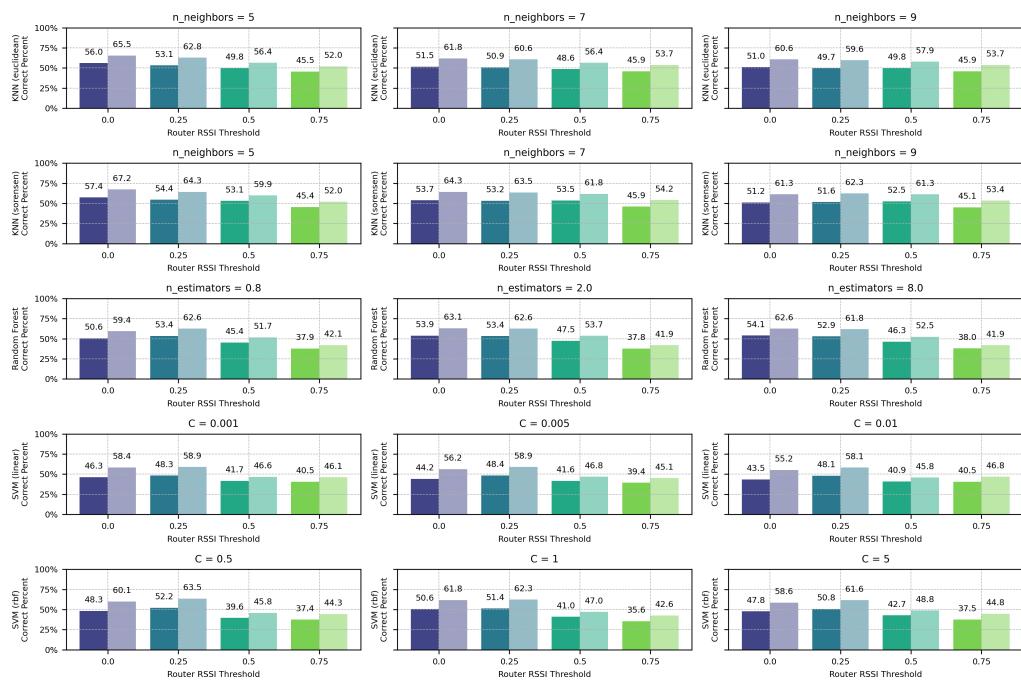


Abbildung 5.13: Vergleich der durchschnittlichen Genauigkeit in Abhängigkeit des Schwellenwerts für die Anzahl der Messungen eines Routers

Wie in Abbildung 5.13 zu sehen ist, sinkt die Genauigkeit bei beiden KNN-Modellen, mit Ausnahme der Sørensen Distanzmetrik für den Wert  $k = 9$ , sowie bei dem Random Forest Modell, mit Ausnahme der Werte für `max_features = 0.8`, mit steigendem Schwellenwert. Bei dem KNN Algorithmus unter Verwendung der Sørensen Distanz und  $k = 9$  und dem Random Forest Algorithmus mit `max_features = 0.8` zeigt der Schwellenwert von 25 % jedoch eine etwas bessere Genauigkeit als der Schwellenwert von 0 %. Bei dem SVM Modell zeigt sich unabhängig vom Kernel, dass die Ergebnisse bei einem Schwellenwert von 25 % bei allen Parametern am besten sind und die Genauigkeit mit zu- und abnehmendem Schwellenwert sinkt. Aus diesem Grund werden für das KNN- und Random Forest-Modell bei den folgenden Untersuchungen alle Router berücksichtigt (Schwellenwert von 0 %) und bei den SVM-Modellen nur die Access Points, die in mindestens 25 % der Messungen eines Raums vorhanden sind.

## Ausschluss von Routern mit schwachem Signal

Im vierten Schritt der Datenaufbereitung wird untersucht, ob die Genauigkeit der Modelle verbessert werden kann, indem Router mit einem schwachen Signal ausgeschlossen werden. Die Annahme dahinter ist, dass Router mit einem schwachen Signal weniger aussagekräftig sind als Router mit einem stärkeren Signal. Aus diesem Grund wurden verschiedene Schwellenwerte getestet, die den minimalen RSSI-Wert angeben, den ein Router haben muss, um berücksichtigt zu werden. Dabei wurden die Schwellenwerte von -100 dBm, -90 dBm, -80 dBm, -70 dBm, -60 dBm, -50 dBm und -40 dBm untersucht. Diese Idee basiert auf dem Paper *Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems.*<sup>6</sup>

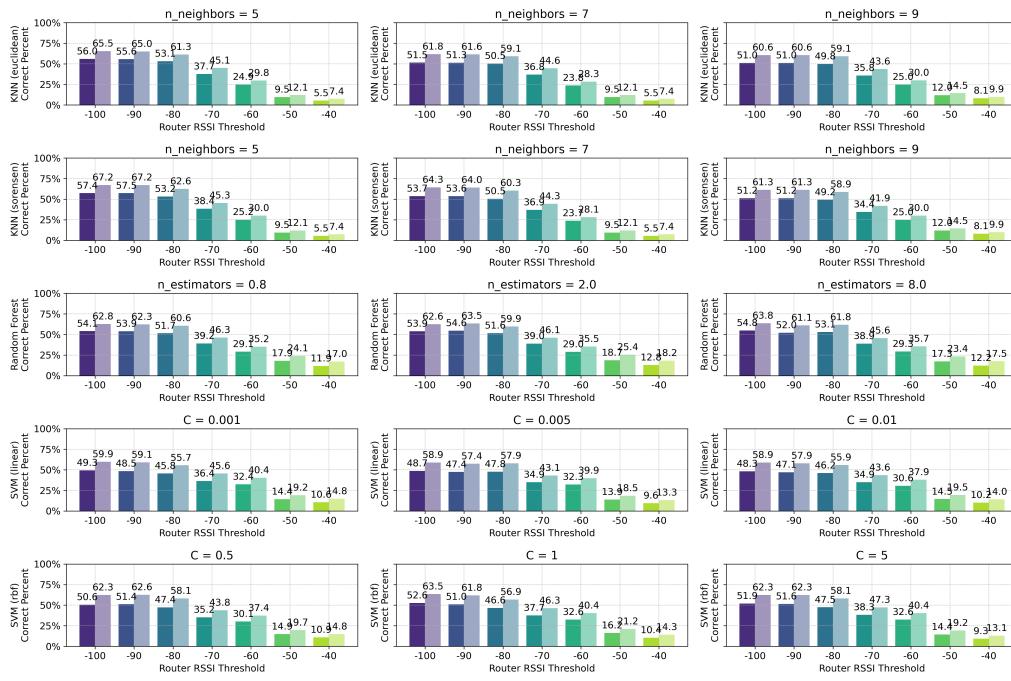


Abbildung 5.14: Vergleich der durchschnittlichen Genauigkeit in Abhängigkeit des Schwellenwerts für die Mindestsignalstärke eines Routers

Wie in Abbildung 5.14 zu erkennen ist, führt die Ignorierung von Routern mit einem schwachen Signal zu keiner Verbesserung der Ergebnisse und die Genauigkeit der Vorhersagen nimmt in den meisten Fällen - abgesehen von ein paar Ausnahmen - linear ab. Bei den Ausnahmen (SVM mit RBF-Kernal und  $C = 0,5$ , Random Forest mit einer maximalen Anzahl von 2 Features und KNN mit der Sørensen-Distanz und  $k = 5$ ) zeigt sich, dass die Genauigkeit bei einem Schwellenwert von -90 dBm leicht besser ist als die Genauigkeit bei einem Schwellenwert von -100 dBm. Insgesamt ist jedoch zu erkennen, dass die Ergebnisse bei einem Schwellenwert von -100 dBm am besten sind. Aus diesem Grund wird für die weiteren Untersuchungen ein Schwellenwert von -100 dBm verwendet.

<sup>6</sup> Torres-Sospedra u. a. (2015), S. 9272

## Skalierung von RSSI-Werten

Im fünften Schritt der Datenaufbereitung wird untersucht, ob die Skalierung der RSSI-Werte die Genauigkeit der Modelle verbessern kann. Die Idee hinter der Werteskaliierung stammt aus dem Paper *Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems* und basiert auf der Erkenntnis, dass die RSSI-Werte nicht linear verteilt sind (siehe Kapitel 2.3). Dafür werden wie in dem Paper beschrieben drei verschiedene Skalierungsmethoden implementiert und mit den bisher nicht skalierten Werten verglichen.<sup>7</sup>

Die drei Skalierungsmethoden sind:

1. **Lineare Normalisierung**
2. **Exponentielle Skalierung**
3. **Potenzierte Skalierung**

Grundlage jeder Skalierung ist die positive Darstellung der Werte. Hierfür wird jeder Wert in den Trainings- und Testdaten mit dem niedrigsten gemessenen RSSI-Wert minus 1 dBm (min) subtrahiert:

$$\text{Positiv}_i(x) = \text{RSS}_i - (\text{min}) \quad (5.3)$$

Durch diese Skalierung werden alle RSSI-Werte positiv dargestellt und der niedrigste Wert ist 1 dBm.

### Lineare Normalisierung:

Die linear normalisierten Werte werden berechnet mit:

$$\text{Normiert}_i(x) = \frac{\text{Positiv}_i(x)}{-\text{min}} \quad (5.4)$$

und befinden nach der Skalierung in dem Wertebereich [0, 1].

### Exponentielle Skalierung:

Die exponentiell skalierten Werte werden berechnet mit:

$$\text{Exponential}_i(x) = \frac{\exp\left(\frac{\text{Positiv}_i(x)}{\alpha}\right)}{\exp\left(\frac{-\text{min}}{\alpha}\right)} \quad (5.5)$$

und  $\alpha = 24$ .

Die Auswahl dieses Wertes für den Parameter  $\alpha$  basiert auf den Ergebnissen aus dem Paper *Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems*.<sup>8</sup>

<sup>7</sup> Torres-Sospedra u. a. (2015), S. 9269

<sup>8</sup> Torres-Sospedra u. a. (2015), S. 9269

## Potenzierte Skalierung:

Die potenzierten Werte werden unter Verwendung von  $\beta = e$  und der Formel:

$$\text{Potenziert}_i(x) = \left( \frac{\text{Positiv}_i(x)}{-\min} \right)^e \quad (5.6)$$

berechnet. Diese Auswahl basiert ebenfalls auf den Ergebnissen aus dem Paper *Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems.*<sup>9</sup>

In Abbildung 5.15 sind die verschiedenen Skalierungsmethoden dargestellt. Für diese exemplarische Darstellung wurden RSSI-Werte zwischen -100 dBm und -1 dBm in einem Intervall von 1 dBm betrachtet.

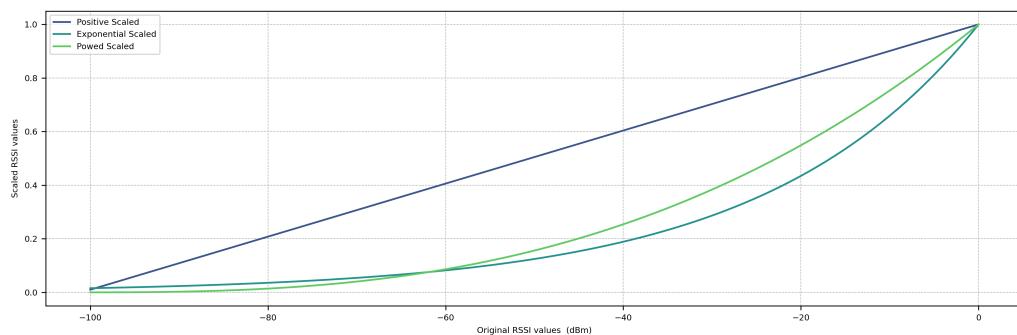


Abbildung 5.15: Darstellung der verschiedenen Skalierungsmethoden

Da in dem KNN-Modell die Distanzen zur Bestimmung der Räume auf den RSSI-Werten basieren und die Skalierung dieser Werte den Wertebereich beeinflusst hat, wird zunächst überprüft, ob die gewählte Gewichtung (`weights = distance`) weiterhin optimal ist. Dazu wurde der KNN-Algorithmus mit beiden Gewichtungsfunktionen und beiden Distanzmetriken jeweils mit linearer Normalisierung, exponentieller Skalierung, potenzierte Skalierung und den unskalierten Werten verglichen.

---

<sup>9</sup> Torres-Sospedra u. a. (2015), S. 9269

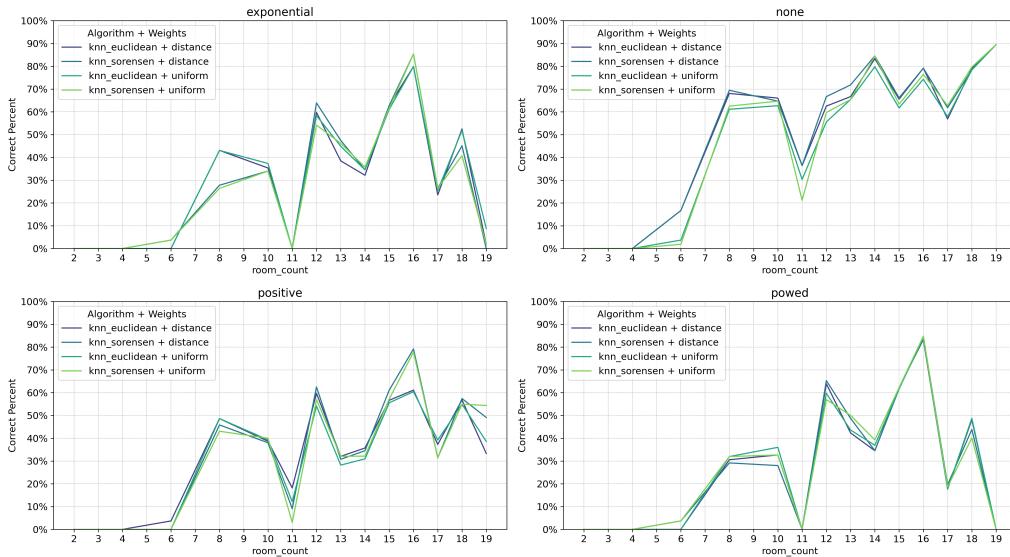


Abbildung 5.16: Vergleich des KNN-Algorithmus mit den verschiedenen Skalierungsmethoden und beiden Gewichtungsfunktionen

Wie in Abbildung 5.16 zu erkennen ist, zeigen die Ergebnisse unabhängig von der gewählten Distanzmetrik und der Gewichtungsfunktion einen ähnlichen Verlauf innerhalb jeder Skalierungsstrategie. Auffällig ist jedoch, dass bei der exponentiellen und der potenzierten Skalierung die Genauigkeit in Räumen mit mehr als 16 Messungen deutlich abnimmt, wobei der Rückgang bei der potenzierten Skalierung weniger stark ausgeprägt ist. Zudem gibt es leichte Unterschiede zwischen den Distanzmetriken: Bei der exponentiellen Skalierung schneidet die Sørensen-Distanz in Räumen mit 8 Messungen schlechter ab und bei der positiven Skalierung in Räumen mit 16 Messungen, jeweils im Vergleich zur euklidischen Distanz.

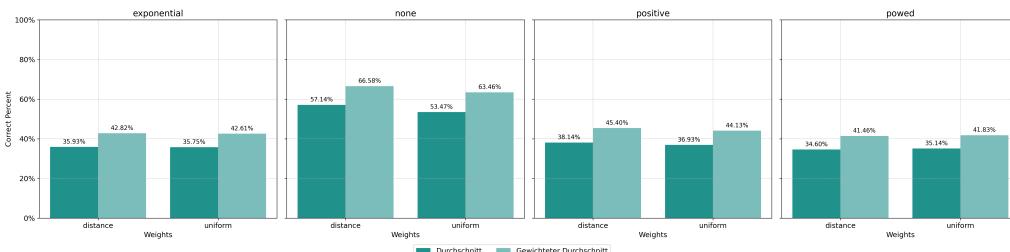


Abbildung 5.17: Vergleich der durchschnittlichen Genauigkeit des KNN-Algorithmus in Bezug auf die verschiedenen Skalierungsmethoden und beide Gewichtungsfunktionen

Wenn man sich die durchschnittliche Genauigkeit der beiden Distanzmetriken in Abhängigkeit der Skalierungsstrategie ansieht, zeigt sich in Abbildung 5.17, dass abgesehen von der potenzierten Skalierung in jedem Fall die gewichtete Distanzfunktion (`weights = distance`) besser abschneidet und dass die besten Ergebnisse mit den unskalierten Werten erzielt werden konnten. Aus diesem Grund wird für die weiteren Untersuchungen weiterhin gewichtete Distanzfunktion verwendet.

Bei der Untersuchung der weiteren Modelle (siehe Abbildungen 5.18 und 5.19) zeigt sich, dass die nicht skalierten Werte in den meisten Fällen bessere Ergebnisse liefern als die skalierten. Es ist auch ersichtlich, dass bei den skalierten Werten die Genauigkeit zwar zunächst mit zunehmender Anzahl an Messungen pro Raum ansteigt, aber ab einem bestimmten Punkt (ungefähr

bei 16 Messungen pro Raum) wieder deutlich abnimmt. Aufgrund dieser Beobachtungen wird in den weiteren Untersuchungen auf die Skalierung der RSSI-Werte verzichtet.

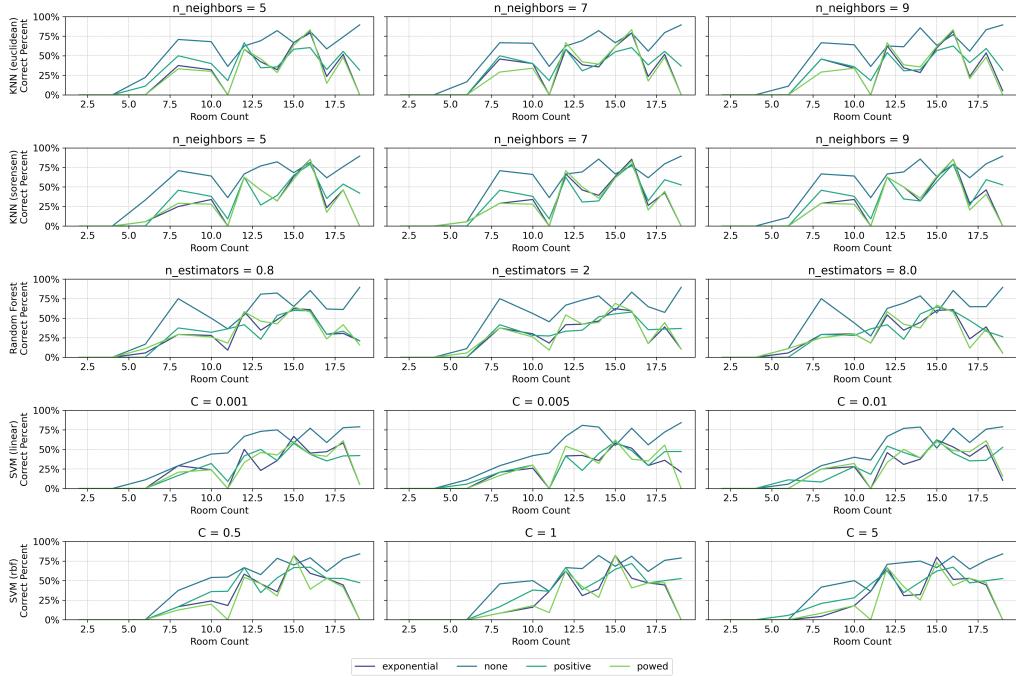


Abbildung 5.18: Vergleich der Genauigkeit in Abhängigkeit der Skalierungsstrategien

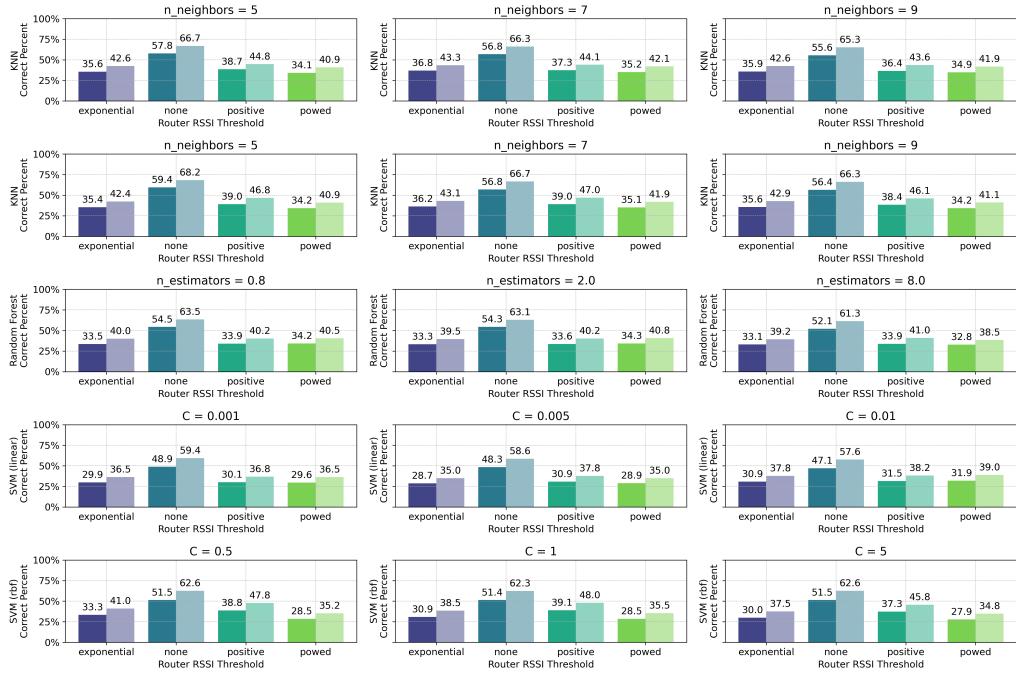


Abbildung 5.19: Vergleich der durchschnittlichen Genauigkeit in Abhängigkeit der Skalierungsstrategien

Insgesamt konnten die besten Ergebnisse mit dem KNN-Modell unter Verwendung der Sørensen-Distanzmetrik, der gewichteten Distanzfunktion und  $k = 5$  erzielt werden.

## 5.5 Erweiterte Untersuchungen

In den Kapiteln 5.3 und 5.4 konnte ermittelt werden, dass bei den Testdaten der KNN-Algorithmus mit der Sørensen Distanzmetrik und einem Wert von  $k = 5$  die besten Ergebnisse erzielen konnte. In dem folgenden Kapitel wird nun konkreter untersucht, wie sich die Anzahl der Messungen auf die Genauigkeit auswirkt und ob eine Verbesserung der Vorhersagegenauigkeit erzielt werden kann, wenn zusätzliche Messungen an Orten durchgeführt werden, die nicht zu den Räumen gehören. Die Idee dabei ist, dass durch Messungen an Orten, die nicht zu den Räumen gehören, die Unterscheidung zwischen den Räumen verbessert werden kann und falsche Vorhersagen reduziert werden können. Für die Orte außerhalb der Räume wurde der Flur zwischen den Räumen ausgewählt. Sollte bei der Raumbestimmung ein Flur vorhergesagt werden, so wird die Vorhersage als nicht falsch gewertet.

Um diese Hypothese zu überprüfen, wurden in vier nebeneinander liegenden Räumen (WH C 335, WH C 351, WH C 352 und WH C 353) sowie in dem Flur zwischen diesen Räumen, weitere Messungen in dem Zeitraum vom 7. Juli 2024 bis zum 19. Juli 2024 durchgeführt, sodass für jeden dieser Räume und den Flurbereich 20 Fingerprints vorhanden sind. Um den Einfluss der Anzahl der Messungen auf die Genauigkeit der Vorhersagen zu untersuchen, wurde für jeden der vier Räume eine Raumvorhersage erstellt. Dabei wurden für jede Messung eine Vorhersage getroffen und alle möglichen Kombinationen aus der Anzahl der Fingerprints pro Raum (1, 3, 5, ..., 19) und der Anzahl der Messungen im Flur (0, 1, 3, 5, ..., 19) berücksichtigt.

In der Abbildung 5.20 ist dargestellt in wie vielen Fällen und unter welcher Anzahl der Messungen pro Raum und auf dem Flur die Raumvorhersage korrekt war (linke Heatmap), in wie vielen Fällen Vorhersage korrekt oder nicht falsch war (mittlere Heatmap) und in wie vielen Fällen der Flur vorhergesagt wurde (rechte Heatmap).

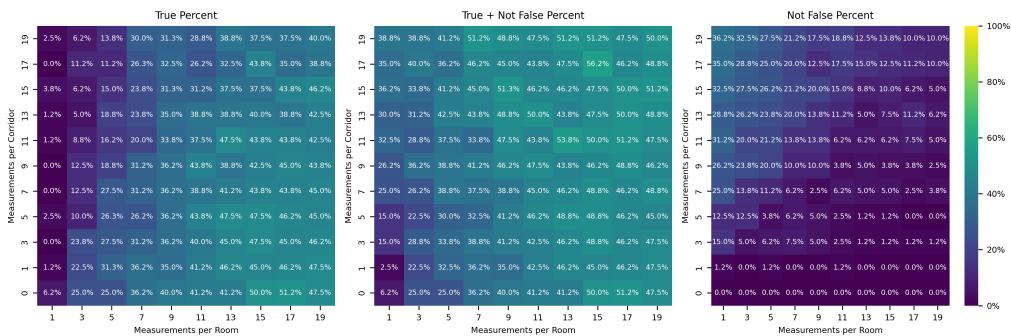


Abbildung 5.20: Vergleich der Genauigkeit in Abhängigkeit der Anzahl an Messungen pro Raum und auf dem Flur

Wie zu erkennen ist, steigt die Anzahl der korrekten Vorhersagen tendenziell mit einer zunehmenden Anzahl an Messungen pro Raum und verringert sich mit einer zunehmenden Anzahl an Messungen auf dem Flur. Die besten Ergebnisse konnten dabei erzielt werden, wenn in jedem Raum 17 Messungen und keine Messungen auf Flur verwendet wurden (51,2 %). Bei der Betrachtung der Fälle in denen keine falsche Vorhersage getroffen wurde, wurden bei 15 Messungen pro Raum und 17 Messungen auf dem Flur in den meisten Fällen keine falschen Vorhersagen getroffen (56,2 %). Von diesen 56,2 % waren entsprachen dabei 77,96 % einer korrekten Vorhersage und 22,04 % einer Vorhersage des Flurs.

# 6 Fazit

## 6.1 Diskussion der Ergebnisse

Die Ergebnisse dieser Arbeit haben gezeigt, dass unter den untersuchten Algorithmen der gewichtete KNN Algorithmus mit der Sørensen-Distanzfunktion und  $k = 5$  die besten Ergebnisse erzielen konnte und dass die Datenaufbereitungsmethoden zu keiner Verbesserung der Ergebnisse geführt haben. Bei der Analyse von benachbarten Räumen in Abhängigkeit von der Anzahl an Messungen und der Verwendung von Messungen außerhalb der Räume ist diese Arbeit zu dem Ergebnis gekommen, dass dort in nur maximal 51,2 % der Fälle eine korrekte Ortung möglich war. Aus diesem Grund kommt die Arbeit zu dem Schluss, dass die WiFi-Fingerprint-basierte Raumortung zwar als Unterstützung für die Raumbestimmung genutzt werden kann, jedoch nicht als alleiniges Mittel zur Raumortung verwendet werden sollte.

## 6.2 Ausblick auf zukünftige Arbeiten

In zukünftigen Untersuchungen könnte dementsprechend überprüft werden, ob mit anderen Techniken wie der Indoor-Ortung via Bluetooth oder Ultra-wideband bessere Ergebnisse erzielt werden können und ob diese vielleicht in Kombination mit der WiFi-Fingerprint-basierten Raumortung genutzt werden können. Ein weiterer Ansatz für zukünftige Arbeiten könnte sein, dass die Geräte, die ihren Raum bestimmen miteinander kommunizieren und die Raumbestimmung nicht allein auf den selbst gemessenen Daten, sondern auch in Abhängigkeit von den Messungen anderer Geräte, basiert, indem die Wahrscheinlichkeiten der Vorhersagen miteinander verglichen werden.

# Literatur

- [1] Shuang Shang und Lixing Wang. „Overview of WiFi fingerprinting-based indoor positioning“. In: *IET Communications* 16 (Apr. 2022). DOI: 10.1049/cmu2.12386.
- [2] Samaneh Amirsoori u. a. „WI-FI Based Indoor Positioning Using Fingerprinting Methods (KNN Algorithm) in Real Environment“. In: *International Journal of Future Generation Communication and Networking* 10.9 (Sep. 2017), 23–36. ISSN: 2233-7857. DOI: 10.14257/ijfgcn.2017.10.9.03. URL: <http://dx.doi.org/10.14257/ijfgcn.2017.10.9.03>.
- [3] Mohammad Masoud, Yousef Jaradat und Mohammad Alia. „IEEE802.11 Access Point’s Service Set Identifier (SSID) for Localization and Tracking“. In: *Computers, Materials & Continua* 71 (Jan. 2022), S. 5459–5476. DOI: 10.32604/cmc.2022.023781.
- [4] IoT [mesh]. *RSSI: "Received Signal Strength Indication: Was ist ein guter Wert? Wie wird der RSSI ermittelt?* <https://www.iot-mesh.de/rssi>. [Accessed 19-08-2024]. 2024.
- [5] Sebastian Sadowski und P. Spachos. „RSSI-Based Indoor Localization with the Internet of Things“. In: *IEEE Access* PP (Juni 2018), S. 1–1. DOI: 10.1109/ACCESS.2018.2843325.
- [6] Xu Han und Zunwen He. „A Wireless Fingerprint Location Method Based on Target Tracking“. In: *2018 12th International Symposium on Antennas, Propagation and EM Theory (ISAPE)*. 2018, S. 1–4. DOI: 10.1109/ISAPE.2018.8634177.
- [7] Yasmine Rezgui u. a. „An Efficient Normalized Rank Based SVM for Room Level Indoor WiFi Localization with Diverse Devices“. In: *Mobile Information Systems* 2017.1 (2017), S. 6268797. DOI: <https://doi.org/10.1155/2017/6268797>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1155/2017/6268797>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2017/6268797>.
- [8] IBM. *Was ist der „k-nearest neighbors algorithm“?* / IBM. <https://www.ibm.com/de-de/topics/knn>. [Accessed 17-08-2024]. 2024a.
- [9] Joaquín Torres-Sospedra u. a. „Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems“. In: *Expert Systems with Applications* 42.23 (2015), S. 9263–9278. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2015.08.013>. URL: <https://www.sciencedirect.com/science/article/pii/S0957417415005527>.
- [10] K. Hechenbichler und Klaus Schliep. „Weighted k-Nearest-Neighbor Techniques and Ordinal Classification“. In: *Discussion Paper* 399 (Jan. 2004).
- [11] scikit-learn developers. *KNeighborsRegressor*. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor>. [Accessed 17-08-2024]. 2024a.
- [12] IBM. *What Is Support Vector Machine?* / IBM. <https://www.ibm.com/topics/support-vector-machine>. [Accessed 17-08-2024]. 2024b.
- [13] Data Basecamp. *Support Vector Machine (SVM) - einfach erklärt!* / Data Basecamp. <https://databasecamp.de/ki/support-vector-machine-svm>. [Accessed 19-08-2024]. 2021.
- [14] Bahaa El-Din Helmy. *The C Parameter in Support Vector Machines / Baeldung on Computer Science*. <https://www.baeldung.com/cs/ml-svm-c-parameter>. [Accessed 20-08-2024]. 2024.

- [15] GeeksforGeeks. *Gamma Parameter in SVM* - GeeksforGeeks. <https://www.geeksforgeeks.org/gamma-parameter-in-svm/>. [Accessed 20-08-2024]. 2024.
- [16] Niklas Donges. *Random Forest: A Complete Guide for Machine Learning / Built In*. <https://builtin.com/data-science/random-forest-algorithm>. [Accessed 17-08-2024]. 2024.
- [17] Yanzhao Wang u. a. „WiFi Indoor Localization with CSI Fingerprinting-Based Random Forest“. In: *Sensors* 18.9 (2018). ISSN: 1424-8220. DOI: 10.3390/s18092869. URL: <https://www.mdpi.com/1424-8220/18/9/2869>.
- [18] Johann Winter. „Implementierung eines Systems zur Indoor-Wegesuche basierend auf dem Fingerprintverfahren“. Bachelorarbeit. Hochschule für Technik und Wirtschaft Berlin, 6. Okt. 2017.
- [19] Zoo Codes. *Getting Started: Monitoring a FastAPI App with Grafana and Prometheus - A Step-by-Step Guide*. [https://dev.to/ken\\_maura1/getting-started-monitoring-a-fastapi-app-with-grafana-and-prometheus-a-step-by-step-guide-3fbn](https://dev.to/ken_maura1/getting-started-monitoring-a-fastapi-app-with-grafana-and-prometheus-a-step-by-step-guide-3fbn). [Accessed 20-08-2024]. 2023.
- [20] Google. *Android Gradle plugin 8.5 release notes / Android Studio / Android Developers*. <https://developer.android.com/build/releases/gradle-plugin>. [Accessed 20-08-2024]. 2024a.
- [21] Google. *Support Library Packages / Android Developers*. <https://developer.android.com/topic/libraries/support-library/packages>. [Accessed 20-08-2024]. 2024b.
- [22] Google. *Wi-Fi scanning overview / Connectivity / Android Developers — developer.android.com*. <https://developer.android.com/develop/connectivity/wifi/wifi-scan>. [Accessed 21-08-2024]. 2024c.
- [23] Kenzo Takahashi. *K-Nearest Neighbor from Scratch in Python*. <https://kenzotakahashi.github.io/k-nearest-neighbor-from-scratch-in-python.html>. [Accessed 20-08-2024]. 2016.
- [24] Florian Zyprian. *Random Forest Developer Guide: 5 ways to implement in Python*. <https://konfuzio.com/en/random-forest>. [Accessed 20-08-2024]. 2023.
- [25] scikit-learn developers. *RandomForestClassifier — scikit-learn.org*. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. [Accessed 21-08-2024]. 2024b.
- [26] Basem Suleiman u. a. „Supervised Learning-Based Indoor Positioning System Using WiFi Fingerprints“. In: *Proceedings of the 2023 International Conference on Advances in Computing Research (ACR'23)*. Hrsg. von Kevin Daimi und Abeer Al Sadoon. Cham: Springer Nature Switzerland, 2023, S. 56–71. ISBN: 978-3-031-33743-7.



# Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter angefertigt habe. Sämtliche Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Quellen entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Berlin, 22. August 2024

Ort, Datum



Friedrich Völkers