Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: Machine Learning

Estudio de algoritmos de aprendizaje automático para el cálculo del LoRaWAN fingerprinting para posicionamiento en exteriores

---

Autor: Lucas de Torre Barrio

Tutor: Joaquín Torres-Sospedra

Profesor: Antonio Lozano Bagén

Madrid, 23 de junio de 2023

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Estudio de algoritmos de aprendizaje automático para el cálculo del LoRaWAN fingerprinting para posicionamiento en exteriores
Nombre del autor:	Lucas de Torre Barrio
Nombre del colaborador/a docente:	Joaquín Torres-Sospedra
Nombre del PRA:	Antonio Lozano Bagén
Fecha de entrega (mm/aaaa):	06/2023
Titulación o programa:	Máster en Ciencia de Datos
Área del Trabajo Final:	Machine Learning
Idioma del trabajo:	Español
Palabras clave	Outdoor positioning, LoRaWAN fingerprinting, clustering

Resumen

El objetivo de este Trabajo Fin de Máster (TFM) es explorar la posibilidad de predecir la posición en exteriores a partir de las señales enviadas a dispositivos LoRaWAN utilizando técnicas de aprendizaje automático similares a las utilizadas para predecir la posición en interiores a partir de las señales WiFi.

Se han utilizado diferentes algoritmos de aprendizaje automático para evaluar su capacidad para predecir la posición a partir de las señales LoRaWAN. Los resultados obtenidos muestran que es posible predecir la posición en exteriores con una precisión aceptable utilizando técnicas de aprendizaje automático y las señales LoRaWAN.

En conclusión, este TFM demuestra que las técnicas de aprendizaje automático pueden ser aplicadas con éxito para predecir la posición en exteriores a partir de las señales enviadas a dispositivos LoRaWAN. Esta investigación podría tener implicaciones importantes en el desarrollo de sistemas de posicionamiento en exteriores más precisos y eficientes para aplicaciones de IoT.

Palabras clave: Outdoor positioning, LoRaWAN fingerprinting

Abstract

The aim of this Master's Thesis is to explore the possibility of predicting outdoor positions based on signals sent to LoRaWAN devices using machine learning techniques similar to those used to predict indoor positions based on WiFi signals. To do this, signal data sent by LoRaWAN devices in a controlled outdoor environment has been collected, and signal processing and machine learning techniques have been used to build position prediction models.

Different machine learning algorithms have been used to evaluate their ability to predict position based on LoRaWAN signals. The results obtained show that it is possible to predict outdoor positions with acceptable accuracy using machine learning techniques and LoRaWAN signals.

In conclusion, this Master's Thesis demonstrates that machine learning techniques can be successfully applied to predict outdoor positions based on signals sent to LoRaWAN devices. This research could have significant implications in the development of more accurate and efficient outdoor positioning systems for IoT applications.

Keywords: Outdoor positioning, LoRaWAN fingerprinting

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Descripción general del problema

La creciente demanda de soluciones de IoT para aplicaciones de monitorización y seguimiento ha llevado a la necesidad de desarrollar sistemas de posicionamiento en exteriores precisos y eficientes. Aunque el sistema de posicionamiento global (GPS) es una tecnología comúnmente utilizada para la localización en exteriores, su alto consumo de energía es una limitación importante para muchos dispositivos IoT.

Por esta razón, se ha explorado la tecnología LoRaWAN, que utiliza señales de radio de baja potencia para la comunicación entre dispositivos y que tiene un consumo de energía significativamente menor que el GPS. Sin embargo, hasta ahora, no se ha investigado ampliamente el uso de LoRaWAN para el posicionamiento en exteriores haciendo uso de técnicas similares a las usadas en WI-Fi fingerprinting.

1.2. Motivación

La motivación detrás de este Trabajo Fin de Máster surge de la necesidad de desarrollar sistemas de posicionamiento en exteriores suficientemente precisos y eficientes para aplicaciones de IoT. Aunque existen tecnologías de posicionamiento en exteriores disponibles, como el GPS, su alto consumo de energía lo hace menos atractivo para dispositivos IoT de baja potencia.

En este sentido, la tecnología LoRaWAN se presenta como una alternativa prometedora para el posicionamiento en exteriores debido a su bajo consumo de energía y la capacidad de proporcionar cobertura en áreas remotas y de difícil acceso.

La investigación y el desarrollo de técnicas de predicción de posición precisas y eficientes utilizando señales LoRaWAN podría tener implicaciones significativas en el desarrollo de sistemas de IoT y abrir nuevas posibilidades para aplicaciones de seguimiento y monitorización en

exteriores.

Por lo tanto, la motivación principal de este Trabajo Fin de Máster es explorar el potencial de la tecnología LoRaWAN para el posicionamiento en exteriores. Además, se busca utilizar técnicas de aprendizaje automático para mejorar la precisión de la predicción de la posición en exteriores utilizando señales LoRaWAN.

1.3. Objetivos

1.3.1. Objetivo principal

El objetivo principal es utilizar técnicas de aprendizaje automático para mejorar la precisión de la predicción de la posición en exteriores utilizando señales LoRaWAN.

1.3.2. Objetivos secundarios

- Evaluar el rendimiento de diferentes algoritmos de aprendizaje automático para la predicción de la posición en exteriores utilizando señales LoRaWAN.
- Proponer y evaluar técnicas de mejora de la precisión en la predicción de la posición en exteriores utilizando señales LoRaWAN.

1.4. Metodología

1.4.1. Estrategia de investigación

La estrategia de investigación de este Trabajo Fin de Máster se divide en tres fases principales:

- **Análisis**: En esta fase, se analizarán y compararán las características de las señales LoRaWAN y las señales WiFi utilizadas para la predicción de la posición en interiores.
- **Evaluación**: Se evaluará el rendimiento de diferentes algoritmos de aprendizaje automático para la predicción de la posición en exteriores utilizando señales LoRaWAN.
- **Propuestas de mejora y validación**: En esta fase, se propondrán y evaluarán técnicas de mejora de la precisión en la predicción de la posición en exteriores utilizando señales LoRaWAN.

1.4.2. Metodología de trabajo

La metodología de trabajo de este Trabajo Fin de Máster se basa en las siguientes fases:

- **Revisión bibliográfica**: Se realizará una revisión bibliográfica para conocer el estado actual de la investigación en el campo de la predicción de la posición en exteriores utilizando señales LoRaWAN y para identificar las técnicas de aprendizaje automático más adecuadas para el problema planteado.
- **Diseño experimental**: Se diseñará un experimento para recopilar datos de señales LoRaWAN en un entorno controlado en exteriores.
- **Análisis y evaluación**: Se analizarán y compararán las características de las señales LoRaWAN y las señales Wi-Fi utilizadas para la predicción de la posición en interiores. Además, se evaluará el rendimiento de diferentes algoritmos de aprendizaje automático para la predicción de la posición en exteriores utilizando señales LoRaWAN.
- **Propuestas de mejora y validación**: Se propondrán y evaluarán técnicas de mejora de la precisión en la predicción de la posición en exteriores utilizando señales LoRaWAN.
- **Análisis de resultados**: Se analizarán los resultados obtenidos en las diferentes fases de la investigación y se realizarán conclusiones sobre la capacidad de las señales LoRaWAN para la predicción de la posición en exteriores y sobre la eficacia de las técnicas propuestas para mejorar la precisión de la predicción de la posición.

1.5. Planificación del proyecto

La planificación consistirá en cuatro partes:

- **Definición y planificación del trabajo final (01/03/2023 - 12/03/2023)**: Se definirá de manera global el objetivo principal del trabajo y el área que se estudiará de cara a la realización del TFM
- **Análisis del estado del arte (13/03/2023 - 26/03/2023)**: Se llevará a cabo una revisión bibliográfica de los estudios realizados de Wi-Fi fingerprinting de cara a poder utilizarlos para las señales LoRaWAN.
- **Diseño e implementación del trabajo (27/03/2023 - 28/05/2023)**: Se realizará el preprocesamiento de datos y se implementarán diferentes algoritmos para comparar su rendimiento. Se llevarán a cabo pruebas y evaluaciones de los algoritmos y se analizarán los resultados obtenidos para extraer conclusiones.

- **Redacción de la memoria (29/05/2023 - 25/06/2023)**: Se estructurará y se escribirá cada uno de los capítulos específicos de la memoria del TFM. También se revisarán y corregirán los errores encontrados.

En la figura 1.1, podemos ver la planificación del proyecto mediante un diagrama de Gantt.

Figura 1.1: Diagrama de Gantt con la planificación temporal del proyecto

Capítulo 2

Estado del arte

2.1. Redes LoRaWAN

LoRaWAN (Long Range Wide Area Network) es un tipo de red de área amplia de baja potencia (LPWAN) que utiliza la tecnología de modulación de espectro ensanchado de difusión por secuencia directa (CSS) para proporcionar una comunicación inalámbrica de larga distancia y bajo consumo de energía para dispositivos de Internet de las cosas (IoT).

Según el sitio web oficial de LoRa Alliance [13], una organización que promueve y estandariza la tecnología LoRaWAN, la red LoRaWAN se basa en un protocolo de comunicación de capa de enlace de datos diseñado para conectar dispositivos de IoT con aplicaciones basadas en la nube a través de redes de operadores de telecomunicaciones.

Las redes LoRaWAN pueden soportar miles de dispositivos IoT en un solo gateway y ofrecen una cobertura de hasta varios kilómetros en áreas urbanas y hasta decenas de kilómetros en áreas rurales con baja densidad de población. Además, debido a su bajo consumo de energía, los dispositivos LoRaWAN pueden funcionar durante varios años con una batería de tamaño pequeño.

De esta forma, resulta muy interesante hacer uso de estas redes para el posicionamiento en exteriores, ya que su bajo consumo se contrapone al alto consumo de las señales GPS.

El estudio consistirá en aplicar técnicas de aprendizaje automático a partir de datos obtenidos mediante señales LoRaWAN. Concretamente, haremos uso de de LoRaWAN fingerprinting. Para ello, se estudiarán trabajos realizados de Wi-Fi fingerprinting, ya que es un área más explotada.

2.2. Wi-Fi fingerprinting

El Wi-Fi fingerprinting es una técnica de posicionamiento basada en la recolección y análisis de información de señales Wi-Fi, que fue introducida en el año 2000 [9], para determinar la ubicación de un dispositivo, principalmente utilizado en entornos interiores. Esta técnica implica la creación de un mapa de huellas digitales de señales Wi-Fi en una determinada área, permitiendo la identificación y comparación de la señal recibida en diferentes ubicaciones a través del análisis de características específicas de la señal, como la intensidad de la señal, el tiempo de propagación y la interferencia.

El Wi-Fi fingerprinting se ha utilizado en diversas aplicaciones, como la navegación interior, la publicidad basada en el contexto y la seguridad de redes, entre otras. Aunque se considera una técnica precisa, su precisión varía según la densidad de los puntos de acceso Wi-Fi y las características del entorno.

Aunque hay diversos enfoques de cara a estudiar la huella digital de Wi-Fi, nosotros estudiaremos el enfoque basado en la intensidad de señal recibida de cada punto de acceso Wi-Fi. De esta forma, los datos utilizados consisten en un conjunto de registros (cada uno correspondiente a una posición), teniendo para cada uno de ellos la intensidad recibida por cada punto de acceso Wi-Fi, además de la posición exacta. Por tanto, la idea es lograr predecir esa posición a partir de las intensidades de señales recibidas de cada punto de acceso Wi-Fi.

Se han estudiado distintos modelos de machine learning para la predicción del Wi-Fi fingerprinting, tales como el k-means, el KNN, redes neuronales... De estos, el que obtiene una mejor precisión son las redes neuronales, pero son computacionalmente costosas, por lo que la predicción es más lenta que con otras soluciones. De esta forma, el algoritmo más utilizado es el KNN, ya que tiene una buena relación entre la precisión y la eficiencia.

De hecho, es muy habitual que los estudios se hagan con 1-NN (es decir, teniendo en cuenta únicamente la huella digital más similar a la que se está estudiando). Sin embargo, en [5] se hacen distintas comparativas de cara a determinar las características óptimas del desarrollo de los modelos de aprendizaje automático utilizados para predecir la ubicación en interiores mediante el uso del Wi-Fi fingerprinting, y se determina que, para el conjunto de datos del estudio, el valor óptimo de k es 13. También, en ese mismo artículo, se comparan distintas métricas, las cuales se recogen en [2], y se concluye que, en el caso de estudio, la mejor es la distancia Sorensen.

Para mejorar el rendimiento, en [12], se consideran solo los elementos del conjunto de entrenamiento que reciben señal de un número determinado de gateways de las que también recibe señal el conjunto de test. Algo parecido se aplica en [8], donde se considera que las huellas están dominadas por una o dos gateways (de las que se reciba más señal).

Además del algoritmo KNN, se han utilizado otros algoritmos para el cálculo del WiFi

fingerprinting, como las redes neuronales [10] y los random forest [6], que permiten, en el estudio, trabajar en tiempo real.

2.3. Resultado previos LoRaWAN

Aunque el estudio del Wi-Fi fingerprinting está más avanzado que el del LorRaWAN fingerprinting, también hay estudios de este último. Por ejemplo, en [11], se utiliza un conjunto de datos de gran tamaño (unas 130.000 muestras) y se comparan diversos algoritmos, siendo el KNN y el random forest los que obtienen un mejor resultado. En [7] también se comparan varios algoritmos para otro conjunto de datos, siendo los mejores resultados aquellos obtenidos mediante la red neuronal multicapa y mediante un KNN con pesos.

Los autores del conjunto de datos utilizado en este estudio obtienen, mediante el uso del algoritmo KNN, un error medio de 398 metros ([1]).

En [4] se hace uso del mismo conjunto de datos y se aplican distintos algoritmos: se hace uso del algoritmo KNN, con la distancia euclídea, y se obtiene un error medio de 394 metros en el conjunto de test. Se hace uso del clasificador extra tree, con el que se obtiene un error medio de 380 metros. Finalmente, también utilizan un perceptrón multicapa, con el que obtienen un error de 358 metros.

De esta forma, vemos que los algoritmos más extendidos son KNN y las redes neuronales. En nuestro estudio, abordaremos estas opciones, aunque haremos estudios ya realizados en Wi-Fi fingerprinting, pero no en LoRaWAN fingerprinting, como son el uso y la comparativa de considerar solo algunas gateways en KNN y compararemos esos resultados con otros algoritmos extendidos en el cálculo del LoRaWAN fingerprinting: las redes neuronales y los random forest. Además, se probará a cambiar la función de pérdida de la red neuronal por la distancia Haversine, que calcula la distancia entre puntos de la tierra dadas su latitud y longitud.

Capítulo 3

Diseño e implementación del proyecto

En este capítulo se detalla el diseño y la implementación del estudio de algoritmos de aprendizaje no supervisado para el cálculo del LoRaWAN fingerprinting con el objetivo de aproximar el posicionamiento en exteriores. Los datos utilizados corresponden a las mediciones del rssi que podemos encontrar en [1]. El código desarrollado se encuentra en [3].

Tras preprocesar los datos, se utilizarán distintos algoritmos para este estudio:

- **KNN**: El algoritmo de vecinos más cercanos, el cual se utiliza con diferentes valores de K, comprendidos entre 1 y 15. Se ha desarrollado también una función personalizada para el algoritmo KNN, que permite considerar solo un subconjunto de vecinos en lugar de todos, con el fin de mejorar la eficiencia del algoritmo.

- **Red neuronal**: Definiremos una red neuronal completamente conectada con el error cuadrático medio como función de pérdida. Le variaremos sus hiperparámetros en busca de los mejores resultados. Usaremos de nuevo los hiperparámetros con mejor resultado para entrenar un modelo con función de pérdida la función Haversine, la más adecuada para nuestro caso de uso.

- **Random forest**: Utilizaremos este modelo por su buena capacidad de manejar grandes conjuntos de datos, así como por su aplicación en casos de WiFi fingerprinting. Para este algoritmo probaremos distintos hiperparámetros en busca de los mejores resultados.

3.1. Preprocesamiento de los datos

Esta etapa tiene como objetivo transformar los datos brutos recopilados en un formato adecuado para el estudio de diversos algoritmos de aprendizaje no supervisado utilizados en el cálculo del LoRaWAN fingerprinting para posicionamiento en exteriores.

El preprocesamiento de los datos consta de varias fases que se describen detalladamente a continuación:

3.1.1. Extracción de atributos

En primer lugar, se realiza la extracción de atributos clave de cada medición registrada en el entorno LoRaWAN. Se incluyen la latitud y longitud del lugar de la medición, la intensidad recibida de cada gateway y el momento de la ejecución en formato epoch. Estos atributos capturan información relevante sobre las mediciones y permiten establecer patrones y relaciones para el posicionamiento en exteriores.

3.1.2. Ordenación cronológica de los datos

Una vez que se han extraído los atributos, se procede a ordenar los datos de manera cronológica. Este paso es esencial para evitar dependencias entre los datos y garantizar que las mediciones más recientes se utilicen como referencia en los algoritmos de aprendizaje. Esto es necesario ya que puede suceder que dos mediciones consecutivas estén hechas en el mismo punto exacto, de forma que usar los valores de una para predecir el posicionamiento de la otra daría un error nulo, siendo un resultado falseado. La ordenación cronológica asegura que las mediciones estén organizadas en un orden secuencial basado en la marca de tiempo de cada medición, lo que facilita el análisis y la posterior implementación de los modelos de aprendizaje no supervisado.

3.1.3. División en conjuntos de entrenamiento y prueba

Una vez que los datos han sido ordenados cronológicamente, se realiza una división adecuada en conjuntos de entrenamiento y prueba. En este caso, se ha optado por asignar el 80 % de los datos ordenados al conjunto de entrenamiento y el 20 % restante al conjunto de prueba. Esta división permite evaluar el rendimiento de diferentes modelos y algoritmos y proporciona una estimación realista de su capacidad para generalizar y realizar el posicionamiento en exteriores.

El conjunto de entrenamiento se utiliza para entrenar los diferentes modelos y algoritmos de aprendizaje no supervisado. Estos modelos aprenden los patrones y relaciones presentes en los datos para realizar el cálculo del LoRaWAN fingerprinting y mejorar el posicionamiento. Por otro lado, el conjunto de prueba se reserva para evaluar el rendimiento de los modelos entrenados y medir su precisión en el cálculo del posicionamiento en exteriores. Esta evaluación proporciona información crítica sobre la capacidad de generalización y la eficacia de los modelos en situaciones reales.

3.2. KNN

Para el desarrollo de este TFM, se ha realizado una implementación detallada del algoritmo de vecinos más cercanos (KNN) con una función personalizada que hará uso de la distancia euclídea para medir la distancia entre los vecinos. KNN será el algoritmo o modelo en el que más se ahondará, ya que es el más utilizado en las tareas de WiFi fingerprinting. Esta función personalizada permite utilizar el KNN convencional, pero también proporciona una opción adicional para mejorar la eficiencia del algoritmo al considerar solo un subconjunto de los vecinos para cada elemento del conjunto de test dentro del conjunto de entrenamiento. Además, la función personalizada permitirá elegir qué métrica utilizar para calcular la distancia a los vecinos.

La función personalizada de KNN se ha diseñado de manera que se pueda utilizar como el KNN convencional, pero con la capacidad adicional de aceptar un vector de índices del mismo tamaño que el conjunto de prueba. Este vector de índices define qué vecinos de entrenamiento se utilizarán para cada elemento de prueba, de forma que para el elemento i-ésimo del conjunto de test, solo se considerarán los vecinos de entrenamiento cuyos índices estén presentes en la posición i-ésima del vector de índices.

Esta mejora en la implementación del algoritmo KNN busca reducir significativamente el tiempo de ejecución, ya que se evita el cálculo de distancias y la consideración de todos los vecinos de entrenamiento. En lugar de ello, se selecciona un subconjunto específico de vecinos, lo que conlleva una reducción en la carga computacional.

A continuación, se muestra el código correspondiente al algoritmo KNN personalizado:

```
def knn_algorithm(k, train_X, test_X, train_y, test_y, considered_index =
    [], distancia = 'euclidea'):
    min_distances = []
    min_distances_index = []

    # Recorremos cada elemento de test
    for i in range(len(test_X)):
        min_distances.append([])
        min_distances_index.append([])
        # Recorremos cada elemento de train que consideramos
        if considered_index != []:
            index_to_consider = considered_index[i]
        else:
            index_to_consider = range(len(train_X))
        for j in index_to_consider:
            # Seleccionamos la metrica para medir la distancia entre vecinos
            if distancia == 'sorensen':
```

```
17                current_distance = sorensen_distance(train_X[j], test_X[i])
18            elif distancia == 'manhattan':
19                current_distance = cityblock(train_X[j], test_X[i])
20            else: # distancia euclidea
21              current_distance = math.dist(train_X[j], test_X[i])
22            if len(min_distances[-1]) < k:
23                min_distances[-1].append(current_distance)
24                min_distances_index[-1].append(j)
25                min_distances_index[-1] = [x for _, x in sorted(zip(
    min_distances[-1], min_distances_index[-1]))]
26                min_distances[-1].sort()
27
28            elif min_distances[-1][-1] > current_distance:
29                min_distances[-1][-1] = current_distance
30                min_distances_index[-1][-1] = j
31                min_distances_index[-1] = [x for _, x in sorted(zip(
    min_distances[-1], min_distances_index[-1]))]
32                min_distances[-1].sort()
33
34    predict_y = []
35
36    # Hallamos las predicciones del conjunto de test
37    for i in range(len(min_distances_index)):
38        punto = [0.0,0.0]
39        for j in min_distances_index[i]:
40            punto[0] += train_y[j][0]
41            punto[1] += train_y[j][1]
42        num_elems = len(min_distances_index[i])
43        punto[0] =1.0*punto[0]/(1.0*num_elems)
44        punto[1] =1.0*punto[1]/(1.0*num_elems)
45        predict_y.append(deepcopy(punto))
46
47    distances_test_predict = []
48    for i in range(len(predict_y)):
49        distances_test_predict.append(haversine(test_y[i][0],test_y[i][1],
    predict_y[i][0],predict_y[i][1]))
50
51    # Calculamos el error medio
52    mean_error = np.mean(distances_test_predict)
53    mean_error
54
55    return predict_y, mean_error, min_distances_index, min_distances
```

No obstante, es importante analizar y evaluar la pérdida de precisión que puede surgir al

utilizar esta función personalizada de KNN en comparación con el KNN convencional. Para realizar esta evaluación, se llevarán a cabo experimentos exhaustivos utilizando conjuntos de datos de prueba y se compararán los resultados obtenidos con ambas implementaciones.

La comparación se centrará en la precisión del algoritmo KNN y se evaluará el error de estimación en el posicionamiento en exteriores. Se analizará el impacto de utilizar diferentes conjuntos de índices para seleccionar los vecinos de entrenamiento, y se buscará encontrar un equilibrio entre la ganancia de eficiencia y la precisión del algoritmo.

Además de evaluar la precisión, también se llevarán a cabo pruebas de rendimiento para medir el tiempo de ejecución y comparar la eficiencia de la función personalizada de KNN con la implementación convencional. Estas pruebas permitirán cuantificar la mejora en la velocidad de procesamiento y validar la utilidad de la función personalizada en términos de eficiencia.

Una vez seleccionado uno de los vectores de índices teniendo en cuenta su precisión y eficiencia, se probarán otras distancias (además de la euclídea) para comprobar si es posible mejorar la precisión del algoritmo.

En todos los casos, el error se haya comparando la latitud y la longitud que se han predicho y las que realmente eran. Para ello, se hará uso de la función Haversine, que permite hallar la distancia en metros entre dos puntos de la tierra dados su latitud y longitud. La función utilizada es la función *geodesic*, del paquete *geopy.distance*.

3.2.1. KNN convencional

Primero se utilizará el algoritmo KNN convencional. Utiliza la distancia euclídea para hallar la distancia entre vecinos y se calculará con valores de K entre 1 y 15.

Para optimizar el algoritmo, se realizará con K = 15 y se extraerán los vecinos más cercanos ordenados de menor a mayor distancia. De esta forma, podemos utilizar los algoritmos más cercanos para calcular el KNN para K entre 1 y 15 sin necesidad de volver a utilizar el algoritmo.

Por último, además de la precisión, se tendrá en cuenta el tiempo de ejecución del algoritmo para poder compararlo con las ejecuciones con vectores de índices.

3.2.2. KNN con vector de índices

Se tendrán en cuenta cuatro casos distintos, para los que crearemos los vectores de índices correspondientes a las siguientes casuísticas:

- **Señal recibida de al menos una gateway coincidente**: Para cada elemento del conjunto de test, se consideran los elementos del conjunto de entrenamiento que verifican que reciben señal de al menos una de las gateways de las que también recibe señal el conjunto de entrenamiento. Es decir, dado un elemento del conjunto de test, cada elemento

del conjunto de entrenamiento será considerado si y solo si existe al menos una gateway para la que ambos elementos han recibido señal (en nuestro caso, esto implica que el rssi sea distinto de -200). En este caso, además, el conjunto de índices tendrá un gran tamaño (superior al que nos permitirá el entorno en el que vamos a realizar las pruebas), por lo que habrá que conseguir trabajar con él a pesar de que nuestro entrono de ejecución (Kaggle) no permite variables tan grandes. Para ello, realizaremos el KNN en dos partes, de forma que cada parte sí que tenga un tamaño razonable para realizar las pruebas.

- **Señal recibida de al menos dos gateways coincidentes**: Para cada elemento del conjunto de test, se consideran los elementos del conjunto de entrenamiento que verifican que reciben señal de al menos dos de las gateways de las que también recibe señal el elemento del conjunto de entrenamiento. Es decir, dado un elemento del conjunto de test, cada elemento del conjunto de entrenamiento será considerado si y solo si existen al menos dos gateways para la que ambos elementos han recibido señal.

- **Señal recibida de al menos tres gateways coincidentes**: Para cada elemento del conjunto de test, se consideran los elementos del conjunto de entrenamiento que verifican que reciben señal de al menos tres de las gateways de las que también recibe señal el elemento del conjunto de entrenamiento. Es decir, dado un elemento del conjunto de test, cada elemento del conjunto de entrenamiento será considerado si y solo si existen al menos tres gateways para la que ambos elementos han recibido señal.

- **Señal recibida de exactamente las mismas gateways**: Para cada elemento del conjunto de test, se consideran los elementos del conjunto de entrenamiento que verifican que reciben señal de exactamente las gateways de las que también recibe señal el elemento del conjunto de entrenamiento. Es decir, dado un elemento del conjunto de test, cada elemento del conjunto de entrenamiento será considerado si y solo si todas las gateways para las que uno de los elementos recibe señal son las mismas gateways para las que el otro elemento también recibe señal.

Para los tres primeros casos, de cara a aumentar la eficiencia, se crea otro vector de índices que contiene las posiciones de las gateways para las que cada elemento del conjunto de test ha recibido señal. En el último caso, se hace lo mismo para el conjunto de entrenamiento. De esta forma, se ahorran muchas comparaciones y se optimiza el algoritmo.

Por último, además de la precisión, se tendrá en cuenta el tiempo de ejecución del algoritmo para poder compararlo con las ejecuciones con vectores de índices.

3.2.3. KNN con vector de índices y distintas métricas

En este último apartado del KNN, se probarán las siguientes distancias haciendo uso del vector de índices que consideremos más apropiado. Las métricas que se utilizarán son:

- **Distancia euclídea**: Es la utilizada en el apartado anterior. Es una de las métricas más utilizadas para el fingerpinting. Se define como:

$$distance_{euclidean}(A, B) = \sqrt{\sum_i |A_i - B_i|^2}$$

- **Distancia Manhattan**: Al igual que la distancia euclídea, es una de las métricas más utilizadas para el fingerpinting. Se define como:

$$distance_{manhattan}(A, B) = \sum_i |A_i - B_i|$$

- **Distancia Sorensen**: Hemos visto que en [5] es la métrica que obtiene mejores resultados. De esta forma, podremos comprobar si ese resultado obtenido para el WiFI fingerprinting es extrapolable al LoRaWAN fingerprinting. Se define como:

$$distance_{sorensen}(A, B) = \frac{\sum_i |A_i - B_i|}{\sum_i (A_i + B_i)}$$

De esta forma, se pretende encontrar métodos ejecutar el algoritmo KNN que tengan un buen equilibrio entre precisión y eficiencia y, posteriormente, podremos hacer una comprobación de si los resultados mostrados en el artículo [5] pueden generalizarse al caso del LoRaWAN fingerprinting, verificando si la distancia Sorensen es o no la más precisa.

3.3. Red neuronal

Se procederá a estudiar la precisión de una red neuronal artificial para el cálculo del LoRaWAN fingerprinting. Para ello, se definirá una red neuronal a la que se le variarán alguno de los hiperparámetros para encontrar la combinación de los mismos que genera los mejores resultados. Se utilizarán las librerías de *keras* y de *tensorflow*.

Concretamente, la red es una red neuronal completamente conectada que permite resolver problemas de regresión. La arquitectura del modelo consiste en varias capas densas. Cada capa densa está compuesta por un conjunto de neuronas que se conectan con todas las neuronas de la

capa anterior. En este caso, se han utilizado múltiples capas densas para aumentar la capacidad de representación del modelo y permitir la extracción de características más complejas.

La estructura de capas utilizada será: capa densa de entrada, capa de dropout, dos capas densas, otra capa de dropuot, otra capa densa y una capa densa de salida. Todas las capas densas tienen activación ReLU para introducir no linealidad en el modelo y permitir la representación de relaciones no lineales entre las variables de entrada y la salida. Se describen las capas a continuación:

- **Capa de entrada**: Esta capa representa la entrada de los datos y tiene 68 neuronas, correspondientes a las variables de entrada del conjunto de datos.
- **Capas ocultas**: Se han añadido varias capas densas (además de la de entrada). Estas capas son responsables de aprender representaciones intermedias y extraer características de los datos.
- **Capas de dropout**: En total hay dos capas de dropout. El dropout es una técnica de regularización que ayuda a prevenir el sobreentrenamiento al desactivar aleatoriamente un porcentaje de las neuronas durante el entrenamiento.
- **Capa de salida**: Esta capa consta de 2 neuronas, que representan las coordenadas de latitud y longitud en el problema de regresión.

El modelo se entrena utilizando el optimizador Adam y se utiliza el error cuadrático medio como función de pérdida. El entrenamiento se realizará durante 500 épocas. El objetivo es minimizar la función de pérdida durante el entrenamiento para ajustar el modelo a los datos de entrenamiento y lograr la mejor predicción posible en los datos de validación.

Para ello, se probarán varios hiperparámetros con el objetivo de buscar la combinación que nos de mejores resultados. Los hiperparámetros probados serán:

- **learning_rate**: Tomará valores 0.001, 0.01, 0.1. Sirve para controlar la velocidad de convergencia del modelo durante el entrenamiento. Si el learning rate es demasiado pequeño, el modelo puede tardar mucho tiempo en converger y si el learning rate es demasiado grande, los pasos pueden ser demasiado grandes y el modelo puede tener dificultades para converger, o incluso divergir.
- **num_neuronas_list**: Tomará valores 32 y 128. Determina el número de neuronas en cada capa oculta del modelo. Cuantas más neuronas tenga una capa oculta, más capacidad tendrá el modelo para aprender representaciones complejas y capturar relaciones no lineales en los datos, aunque, un número excesivo de unidades puede llevar al sobreentrenamiento del modelo.

- **dropout_valores**: Tomará valores 0.001, 0.01, 0.1. Determina la fracción de neuronas que se desactivarán aleatoriamente durante el entrenamiento para evitar el sobreentrenamiento.

Se escogerán los hiperparámetros que den un mejor resultado y serán utilizados para entrenar un modelo en el que se definirá la función de pérdida como la función Haversine. Esta función será implementada con el objetivo de minimizar la distancia en metros entre los valores predichos y los valores reales. Para ello, se hará uso del paquete *backend* del módulo *keras*.

Por un lado, será útil para comparar el uso de esta función de pérdida con el uso del error cudrático medio. Por otro lado, se utilizará la red neuronal con mejores resultado y estos resultados serán comparados con el KNN con mejores resultados.

3.4. Random forest

Por último, se entrenará un modelo de random forest para aproximar la latitud y la longitud de los elementos de test de nuestro conjunto. Para esto, el random forest se basa en la creación de múltiples árboles de decisión, donde cada árbol se entrena con una muestra aleatoria de los datos de entrenamiento y realiza predicciones independientes. Luego, las predicciones de cada árbol se combinan para obtener una predicción final.

Tiene la ventaja de tener la capacidad de manejar conjuntos de datos grandes, lo cual es muy útil en nuestro caso, donde la cantidad de datos es de varias decenas de miles de elementos.

Para el desarrollo del modelo de random forest se utilizará la librería *RandomForestRegressor*. Se procederá de forma similar a la de la red neuronal, definiendo una función en la que se podrán variar los hiperparámetros.

Los hiperparámetros probados serán:

- **n_estimators**: Tomará valores 50, 100 y 200. Define el número de árboles del random forest.
- **max_depth**: Tomará valores None, 5 y 10. Determina la profundidad máxima de cada árbol (en el caso del valor None no se impone un máximo).
- **min_samples_split**: Tomará valores 2, 5 y 10. Determina el mínimo número de muestras que puede tener una hoja para dividirse en más ramas del árbol.

3.5. Comparativa

Por último, se realizará una comparativa general. Se comprobará cuál de los tres métodos ha tenido mejores resultados, si el KNN, la red neuronal o el random forest. Se tendrán en

cuenta tanto los resultados obtenidos como el tiempo de ejecución. Si hay uno de los métodos que obtiene unos resultados mucho mejores que el resto, el tiempo de ejecución no será tan prioritario, pero, si por el contrario, tenemos casos con resultados parecidos, el tiempo de ejecución servirá para discernir qué método es más útil.

Además, se comprobará si los cambios realizados en cada uno de los modelos (variaciones de K y uso de vector de índices en el KNN, distintos hiperparámetros y uso de la función de Haversine como función de pérdida en la red neuronal y distintos hiperparámetros en el random forest) han dado lugar a cambios significativos o, por el contrario, únicamente producen pequeñas variaciones.

Capítulo 4

Conjunto de datos y resultados

En este apartado se describirá detalladamente el conjunto de datos utilizado durante el proyecto, así como los resultados obtenidos a través de la implementación de los algoritmos de aprendizaje no supervisado para el cálculo del LoRaWAN fingerprinting y su aplicación en el posicionamiento en exteriores.

4.1. Conjunto de datos

El conjunto de datos se ha obtenido de [1]. Este conjunto consiste en 123.529 registros con 73 variables. Cada uno de los registros se corresponde con una medida de la señal LoRaWAN recibida por cada uno de los 68 emisores de señal y viene especificada por las siguientes variables:

- **rssi**: Las primeras 68 variables se corresponden con el rssi de cada uno de las estaciones LoRaWAN (estas variables están siempre igual ordenadas). Son números enteros en los que el -200 indica que no se ha recibido señal de ese punto LoRaWAN.
- **RX time**: Específica el timestamp del momento en el que se captan las señales LoraWAN correspondientes a esa muestra.
- **Spreading factor**: Es el factor de dispersión. Se representa mediante un número entero. No utilizado en este estudio.
- **HDOP**: Horizontal Dilution Of Precision, una medida de la calidad de señal GPS. Número real entre 0 y 1. No utilizado en este estudio.
- **Latitude**: Representa la latitud desde la que se han recibido las señales LoraWAN correspondientes a esta muestra. Está en grados.

- **Longitude**: Representa la longitud desde la que se han recibido las señales LoraWAN correspondientes a esta muestra. Está en grados.

Para nuestro estudio, se ha utilizado la variable RX time para ordenar conológicamente los datos. De esta forma, se ha usado el primer 80 % como conjunto de entrenamiento para predecir las mediciones posteriores, ya que una selección aleatoria puede dar lugar a muestras muy cercanas con el consecuente falseo de resultados. Se han utilizado también las variables correspondientes a las rssi, que se han utilizado como variables de entrada para predecir la posición final. Esta posición final se ha calculado como latitud y longitud y se ha comparado con las reales (las que aparecen en la muestra). De esta forma, las variables que no se han utilizado han sido el Spreading factor y el HDOP.

4.2. Resultados KNN

En primer lugar, se ha utilizado un modelo KNN con K natural entre 1 y 15. Posteriormente, se han utilizado modelos KNN con los mismos valores de K, pero (con intención de mejorar el tiempo de entrenamiento) obligando a que los vecinos hayan recibido señal de exactamente las mismas gateways, de al menos tres gateways coincidentes, de al menos dos gateways coincidentes y de al menos una gateway coincidente.

Se pueden ver los tiempos de ejecución en la tabla resultados obtenidos en la tabla 4.1 y los resultados obtenidos en la tabla 4.2.

	Usual	**1 coincidente**	**2 coincidentes**	**3 coincidentes**	**Todos coincidentes**
Tiempo ejecución (segundos)	12944	3526	1411	137	**134**

Cuadro 4.1: Tiempos de ejecución KNN

K	Usual	1 coincidente	2 coincidentes	3 coincidentes	Todos coincidentes
1	432.0	432.0	432.0	440.0	440.0
2	293.4	293.4	293.4	297.4	297.4
3	**245.5**	**245.5**	**245.4**	**248.1**	**248.1**
4	288.5	288.5	288.5	292.6	290.4
5	316.0	316.0	316.0	291.3	317.5
6	311.0	311.0	311.0	316.1	312.4
7	329.4	329.4	329.4	337.8	330.6
8	346.8	346.8	346.8	315.5	347.8
9	326.0	326.0	326.0	319.3	326.8
10	366.0	366.0	366.0	306.1	366.8
11	364.5	364.5	364.5	325.4	365.2
12	349.5	349.5	349.5	320.0	350.4
13	362.5	362.5	362.5	317.5	363.3
14	355.2	355.2	355.2	312.6	355.9
15	350.7	350.7	350.7	298.5	351.4

Cuadro 4.2: Error en metros con los algoritmos KNN

Se observa que los tiempos de ejecución mejoran sustancialmente cuando restringimos el conjunto sobre el que buscar los vecinos más cercanos. Respecto al error en los resultados, en todos los casos se logra con K = 3 y, además, todos tienen valores similares. Sin embargo, en el caso en el que se obliga a que haya al menos 3 gateways coincidentes, se obtienen mejores resultados para K entre 8 y 15 que en el resto de casos.

Posteriormente, se utiliza el KNN con 3 gateways conincidentes, pero con distancias distintas: la Sorensen y la Manhattan. se vuelven a obtener los mejores resultados para K = 3 en ambos casos. Los resultados se pueden ver en 4.3.

	3 coincidentes euclidea	**3 coincidentes Sorensen**	**3 coincidentes Manhattan**
Tiempo ejecución (segundos)	137	415	329
Error medio K=3 (mse)	248.1	402.3	234.6

Cuadro 4.3: Resultados KNN Sorensen y Manhattan

Se observa que la distancia sorensen empeora tanto en tiempo como en precisión a la dis-

tancia euclídea. Sin embargo, aunque la distancia Manhattan empeora el tiempo de ejecución de la distancia euclídea, mejora el error medio, siendo el mejor que encontramos en las distintas variantes de KNN probadas.

4.3. Resultados red neuronal

Se han entrenado redes neuronales (con capa densa de entrada, capa de dropout, dos capas densas, otra capa de dropuot, otra capa densa y una capa densa de salida) con distintos hiperparámetros: el número de neuronas ha sido 32 y 128, el learning rate 0.001, 0.01 y 0.1 y el dropout rate ha sido 0.2, 0.4 y 0.6.

Para las redes neuronales de 32 neuronas, los resultados se pueden ver en 4.4, y para las de 128 neuronas, en 4.5.

	learning rate = 0.001	**learning rate = 0.01**	**learning rate = 0.1**
dopout rate = 0.2	1971	37401	2342
dopout rate = 0.4	1683	2890	2418
dopout rate = 0.6	1907	3158	1692

Cuadro 4.4: Error en metros con redes de 32 neuronas

	learning rate = 0.001	**learning rate = 0.01**	**learning rate = 0.1**
dopout rate = 0.2	1384591	3271	1833
dopout rate = 0.4	12395	1786	2129
dopout rate = 0.6	3770	3417	1724

Cuadro 4.5: Error en metros con redes de 128 neuronas

Se ve que el mejor resultado se obtiene con la red de 32 neuronas, un learning rate de 0.001 y un valor de dropout de 0.4.

De esta forma, se utilizan esos hiperparámetros para entrenar una red neuronal en la que la función de pérdida sea la función Haversine (que calcula la distancia entre dos puntos de la tierra dada su latitud y longitud).

En este caso, el error obtenido es mucho mayor, de 2643059 metros.

En todos los casos el tiempo de ejecución aproximado es de una hora.

4.4. Resultados random forest

Se ha entrenado random forest con distintos hiperparámetros: el número de árboldes ha sido 20, 100 y 200, la profudidad máxima ha sido ilimitada, 5 y 10 y el mínimo de muestras para hacer split ha sido de 2, 5 y 10.

Para las random forest de 50 árboles, los resultados se pueden ver en 4.6, para los de 100 árboles en 4.7, y para los de 200 en 4.8.

	profundidad $= \infty$	**profundidad** $= 5$	**profundidad** $= 10$
min samples split $= 2$	489	965	648
min samples split $= 5$	479	969	650
min samples split $= 10$	471	965	651

Cuadro 4.6: Error en metros con random forest de 50 árboles

	profundidad $= \infty$	**profundidad** $= 5$	**profundidad** $= 10$
min samples split $= 2$	487	964	647
min samples split $= 5$	477	966	645
min samples split $= 10$	470	966	648

Cuadro 4.7: Error en metros con random forest de 100 árboles

	profundidad $= \infty$	**profundidad** $= 5$	**profundidad** $= 10$
min samples split $= 2$	487	966	649
min samples split $= 5$	476	965	649
min samples split $= 10$	470	966	649

Cuadro 4.8: Error en metros con random forest de 200 árboles

Se observa que el número de árboles no afecta notablemente en los resultados. También vemos que los mejores resultados se obtienen cuando no se limita la profundidad máxima y cuando el número mínimo de muestras para dividir un nodo es 10. En todos los casos, el tiempo de ejecución es de aproximadamente 20 segundos.

4.5. Comparativa entre algoritmos

Se ha visto que los resultados obtenidos con KNN y random forest son mucho mejores que los que se han obtenido con las redes neuronales artificiales. En la tabla 4.9 podemos ver una

comparativa del error medio en metros y el tiempo total de entrenamiento y predicción entre el algoritmo KNN con mejores resultados (con K=3, vector de índice con al menos 3 coincidentes y distancia Manhattan), la red neuronal con mejores resultados (32 neuronas, learning rate de 0.001 y dropout de 0.4) y el random forest con mejor resultado (100 árboles, un mínimo de 10 muestras para hacer split y sin límite de profundidad).

	KNN	**Red neuronal**	**Random forest**
Tiempo ejecución (segundos)	329	3547	37
Error medio (mse)	235	1683	470

Cuadro 4.9: Comparativa entre algoritmos

Esto se puede deber a que KNN y random forest son más adecuados en situaciones donde los datos son relativamente simples (en este caso tenemos 68 variables del mismo tipo), mientras que las redes neuronales pueden ser más efectivas cuando los datos son complejos.

Además, puede ocurrir que, con otros hiperparámetros, los resultados obtenidos con redes neuronales se vieran mejorados, así como aplicando estas redes neuronales en otro conjunto de datos.

Capítulo 5

Conclusiones

Con el estudio realizado, se observa que los mejores resultados se han obtenido con el algoritmo KNN con K = 3, usando la distancia Manhattan y teniendo en cuenta sólo aquellos elementos del conjunto de entrenamiento en los que se recibe señal de al menos tres gateways de las que también recibe señal el elemento del conjunto de test. En este caso, el error ha sido de 235 metros y el tiempo de ejecución ha sido de 329 segundos (poco más de 5 minutos).

De hecho, este error es mejor que el que logran los autores del dataset con KNN (en [1]), que es de 398 metros y que el logrado en [4], de 358 metros. Hay que tener en cuenta que la partición realizada en nuestro estudio es temporal, mientras que la realizada en [1] es aleatoria, algo que, en principio, da lugar a mejores resultados.

Respecto a la variable de índices con un tamaño superior al permitido por el entorno (que ha obligado a dividir ese conjunto de índices en dos), se ha visto posteriormente que era posible optimizar dicha variable. Para ello, habría que haber considerado, para cada gateway (de las 68 disponibles), qué elementos reciben señal de dicha gateway, en lugar de tener, para cada elemento, los elementos que reciben señal de al menos una gateway coincidente con dicho elemento. De esta forma, se reduce en gran medida la dimensión de la variable y es posible trabajar con ella más fácilmente.

Además, se ha visto que, en el mejor caso del KNN, la mejor distancia ha sido la distancia Manhattan, y que la Sorensen no mejoraba ni a esta ni a la euclídea. Esto supone que el resultado de que la mejor distancia es la distancia Sorensen, obtenido en este estudio [5] para un caso de posicionamiento en interiores usando WiFi fingerprinting, no es generalizable para el caso de posicionamiento en exteriores.

Por otro lado, el mejor tiempo de ejecución se logra con el random forest, en el que cualquier combinación de parámetros tenía una duración inferior al minuto. Además, aunque los errores eran claramente peores que en el mejor caso del KNN, el buen tiempo de ejecución puede hacer que este método se pueda considerar en casos de necesitar aproximar una localización con gran

velocidad, ya que el error, aún siendo más alto, sigue siendo razonable para una localización en exteriores.

El peor caso ha sido el de las redes neuronales, que han conseguido los peores resultados en error medio, y el tiempo de entrenamiento ha sido muy lento (aunque, una vez entrenadas, las predicciones son rápidas). Además, el uso de la distancia Haversine como función de pérdida empeora enormemente los resultados.

Por tanto, podemos concluir que, en este caso, la mejor opción es usar el algoritmo KNN (con las propiedades descritas), que es uno de los algoritmos más utilizados para el posicionamiento en interiores, por lo que su uso es generalizable.

Capítulo 6

Trabajo futuro

Con los resultados obtenidos a lo largo de este estudio, se puede explorar la posible generalización de los mismos a otros conjuntos de datos, tanto similares como de un tamaño menor, ya que el conjunto utilizado es particularmente grande. De esta forma, podríamos observar lo robustos y escalables que son estos resultados.

Además, aunque se han explorado algunos hiperparámetros para las redes neuronales y los random forest, existe todavía un abanico para su exploración y optimización. Del mismo modo, se puede hacer uso de otros algoritmos para comprobar su eficacia, como puede ser una red neuronal convolucional.

Bibliografía

[1] Aernouts, Michiel, Berkvens, Rafael, Van Vlaenderen, Koen, Weyn, Maarten. (2018). Sigfox and LoRaWAN Datasets for Fingerprint Localization in Large Urban and Rural Areas (1.1) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.1212478

[2] Cha, S. H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. International Journal of Mathematical Models and Methods in Applied Sciences, 1, 300–307

[3] de Torre, Lucas (2023). TFM. Github. https://github.com/ldetorreUOC/TFM

[4] G. G. Anagnostopoulos and A. Kalousis, "A Reproducible Comparison of RSSI Fingerprinting Localization Methods Using LoRaWAN," 2019 16th Workshop on Positioning, Navigation and Communications (WPNC), Bremen, Germany, 2019, pp. 1-6, doi: 10.1109/WPNC47567.2019.8970177.

[5] Joaquín Torres-Sospedra, Raúl Montoliu, Sergio Trilles, Óscar Belmonte, and Joaquín Huerta. 2015. Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems. Expert Syst. Appl. 42, 23 (December 2015), 9263–9278. https://doi.org/10.1016/j.eswa.2015.08.013

[6] Luca Calderoni, Matteo Ferrara, Annalisa Franco, Dario Maio, Indoor localization in a hospital environment using Random Forest classifiers, Expert Systems with Applications, Volume 42, Issue 1, 2015, Pages 125-134, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2014.07.042

[7] Messaoud Ahmed Ouameur, Manouane Caza-Szoka, Daniel Massicotte, Machine learning enabled tools and methods for indoor localization using low power wireless network, Internet of Things, Volume 12, 2020, 100300, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2020.100300.

[8] N. Marques, F. Meneses, and A. Moreira, "Combining similarity functions and majority

rules for multi-building, multi-floor, WiFi positioning" in Proc. Int. Conf. Indoor Positioning Indoor Navigat., 2012.

[9] P. Bahl and V. N. Padmanabhan, "RAFAR: An in-building RFbased user location and tracking system," in Proc. 19th Annu. Joint Conf. IEEE Comput. Commun. Societies, 2000, pp. 775–784.

[10] Rafael Saraiva Campos, Lisandro Lovisolo, Marcello Luiz R. de Campos, Wi-Fi multi-floor indoor positioning considering architectural aspects and controlled computational complexity, Expert Systems with Applications, Volume 41, Issue 14, 2014, Pages 6211-6223, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2014.04.011.

[11] Thomas Janssen, Rafael Berkvens, Maarten Weyn, Benchmarking RSS-based localization algorithms with LoRaWAN, Internet of Things, Volume 11, 2020, 100235, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2020.100235.

[12] T. J. Gallagher, B. Li, A. G. Dempster, and C. Rizos, "A sectorbased campus-wide indoor positioning system" in Proc. Int. Conf. Indoor Positioning Indoor Navigation, 2010.

[13] What is LoRaWAN® Specification - LoRa Alliance®. Retrieved March 25, 2023, from https://lora-alliance.org/about-lorawan/.