Machine Learning-Based Classification and Prediction of Context Using WiFi RSSI

Princy Chauhan
202311002@daiict.ac.in

Harsh Mistry
202311003@daiict.ac.in

Diti Soni
202311010@daiict.ac.in

I. Introduction

Indoor localization is an essential technology that enables accurate positioning and navigation within buildings, where GPS signals are often weak or unavailable. It plays a vital role in applications such as indoor navigation, asset tracking, and emergency response. However, achieving high localization accuracy in indoor environments is challenging due to signal attenuation, interference, and variability caused by factors like device differences and environmental changes. This project focuses on improving indoor localization accuracy using WiFi signal data, specifically Received Signal Strength Indicator (RSSI) values. To achieve this, we have used two machine learning models: Decision Tree and Random Forest. The Decision Tree provides a simple and interpretable model, while the Random Forest, being an ensemble method, combines multiple decision trees to deliver more accurate and reliable predictions.

The report outlines the problem, the methodology used, the implementation of these models, and the results obtained. The use of these techniques provides a more robust solution for indoor localization compared to traditional methods.

II. Problem Statement

Current indoor localization systems that use WiFi signals face several problems, making them less accurate in real-world situations. The main issue is that these systems are very sensitive to changes in the environment, such as:

- **Different Devices:** Different smartphones and other devices report WiFi signal strength (RSSI) differently because they have different hardware. This causes inconsistency in the location predictions.
- **Changes Over Time:** WiFi signals change over time due to factors like people moving around, obstacles appearing or disappearing, and WiFi access points being turned on or off. These changes can make location estimates less accurate.
- **Signal Variability:** The strength of WiFi signals can vary greatly depending on things like walls, furniture, or other electronic devices in the environment. This variability can lead to incorrect location predictions.

These issues make current indoor localization systems unreliable. The goal of this project is to create a more stable system by using a weighted ensemble classifier. This method combines multiple classifiers to handle different conditions and devices. It assigns more importance to classifiers that work better in certain situations, helping to reduce the impact of signal changes and context variations.

By doing this, the project aims to:

- Address the problem of different devices giving inconsistent signal data.
- Reduce the effect of environmental and time-based changes on the accuracy of the system.
- Provide a more accurate and reliable indoor localization system in real-world conditions.

III. App Overview

The *WiFiScanner* application was developed as a tool for collecting WiFi network data essential for the project on machine learning-based classification and prediction of context using WiFi RSSI. This application plays a crucial role in gathering the required information to develop a robust indoor localization system. By addressing challenges such as signal variability, *WiFiScanner* aims to improve data accuracy and reliability for classifier training.

A. Features of WiFiScanner

The *WiFiScanner* app is designed to perform the following tasks:

- Scan and identify all nearby WiFi networks.
- Collect and store detailed information from each network.
- Save the collected data in a structured CSV format for analysis and model training.

B. Data Collection Parameters

The app collects the following key parameters:

- **BSSID:** The Basic Service Set Identifier (BSSID) is a unique identifier for each WiFi network, crucial for distinguishing between networks.
- **SSID:** The Service Set Identifier (SSID) represents the network name, helping identify overlapping networks in the environment.
- **Timestamp:** Each entry includes a timestamp in UTC format, allowing temporal analysis of signal strength changes.
- **Frequency:** The frequency or channel on which each network operates, providing insights into network interference and characteristics.
- **Signal Strength:** The Received Signal Strength Indicator (RSSI) is recorded in dBm, crucial for proximity estimation and understanding signal variability.

- **Location:** If GPS is enabled, the app records the physical location where the data is collected, linking WiFi data to specific areas, which is essential for indoor localization.

C. App Workflow

The *WiFiScanner* application operates in the following manner:

1) Upon activation, the app initiates a WiFi scan to detect all nearby networks.
2) For each detected network, it collects the parameters listed above (BSSID, SSID, Timestamp, Frequency, Signal Strength, and Location).
3) The collected data is stored in a CSV file where each row represents the details of a single network scan.
4) The CSV file provides a structured dataset that is easy to import for further analysis and classifier training.

D. Data Storage Format

The data collected by *WiFiScanner* is stored in a CSV file with the following structure:

- **Columns:** BSSID, SSID, Timestamp, Frequency, Signal Strength, Location
- **Format:** The timestamp is recorded in UTC format, and signal strength is recorded in dBm. Each row corresponds to a single network scan.

This structured storage format ensures easy accessibility and usability of the data for further processing and analysis in the project.

Data Collection Process

The data collection process was conducted in a systematic manner to ensure consistent labeling and comprehensive coverage of all designated locations, times, and grid cells. To maintain organization and streamline analysis, a standardized labeling format was implemented:

Label Format: BuildingName_Floor_Time_Column_CellNo

Examples of labels:

- `LAB_0_9AM_A_5`
- `LAB_1_3PM_B_7`

This labeling system allowed for clear differentiation of data based on location, floor, time slot, and specific grid cells. Data collection was performed across three buildings: the LAB Building, the CEP Building, and the HoR Men Building. Details about the grid design for these buildings are outlined in the previous section.

Steps for Data Collection:

1) Use the WiFiScanner Android application.
2) Assign a label to each grid cell using the standardized format.
3) Stay stationary in the selected grid cell for 15 seconds, during which the app automatically logged one data point per second. Data collection could also be manually halted before the 15-second interval if needed.
4) Update the label for the next grid cell and move to its location.
5) Repeat this process for all grid cells, floors, and time slots in each building.

This structured approach ensured uniform data collection across all locations and conditions.

IV. Data Preprocessing

Data preprocessing plays a crucial role in transforming raw data into a format suitable for analysis and model development. In this project, the preprocessing steps involved several stages aimed at cleaning, transforming, and preparing data for further processing.

A. Data Collection and Integration

The first step involved collecting data from over 80 CSV files. These files were obtained from various sources and contained different aspects of the dataset. The challenge was to merge these multiple CSV files into a single unified dataset. We employed a script to automate this process, ensuring that all files were integrated correctly while maintaining data consistency. During this step, we ensured that column names were standardized across the files and merged them based on common keys or identifiers.

B. Handling Missing Values

After merging the data from over 80 CSV files, the next critical step was addressing missing values within the dataset. To ensure the integrity of the analysis, rows containing any missing data were removed. This approach ensured that the dataset used for modeling was complete and reliable.

- Any row containing missing data was removed to avoid skewing the analysis and to maintain data quality.

C. Feature Scaling and Normalization

As part of the data preparation, we applied feature scaling and normalization to ensure that all numerical features were on the same scale. This was particularly important for machine learning models that are sensitive to the magnitude of input features. The following methods were used:

- *Min-Max Scaling* was applied to scale the numerical features into a range from 0 to 1.
- *Z-score Normalization* was used to transform the data to have a mean of 0 and a standard deviation of 1.

D. Categorical Data Encoding

Since many machine learning algorithms require numerical inputs, categorical variables were encoded into numeric representations:

- *Label Encoding* was applied to ordinal variables, where categories had a natural order, to convert them into numerical labels.

E. Dimensionality Reduction

In order to reduce the complexity of the dataset and improve model performance, we applied dimensionality reduction techniques. This included:

- *Principal Component Analysis (PCA)* was used to reduce the number of features while retaining the variance in the dataset.
- *Feature Selection* techniques were employed to remove redundant or irrelevant features.

F. Data Splitting

Once the data was cleaned and transformed, it was split into training and testing sets using an 80-20 split. This method is widely used to ensure that enough data is available for both training the model and evaluating its performance. The dataset was divided as follows:

- 80% of the data was allocated for training the model, ensuring that the model had sufficient data to learn patterns and relationships.
- 20% of the data was reserved for testing, providing an unbiased evaluation of the model's performance on unseen data.

This splitting strategy helped maintain the integrity of the data while allowing for effective training and evaluation of the machine learning models. The final step of preprocessing ensured that the data was properly prepared for analysis and model training, paving the way for obtaining accurate and reliable results.

After merging and removing all rows with missing values and combining all the csv files and preprocessing we formated dataset using below steps:

1) **Loading Data:** The dataset was loaded using the Pandas library from a CSV file named `merged_file.csv`.
2) **Extracting Features:** Several features were extracted from the `Location` column to create new attributes:
   - **Block:** The block number was extracted from the `Location` column using a regular expression to capture numeric values at the end of the string.
   - **Floor:** The floor number was extracted using a pattern that captures the number following the prefix `LAB_`.
   - **Area:** The area information was extracted and mapped to numerical values for consistency, using a dictionary that assigns each area code to a unique number.
3) **Cleaning Data:** The `BSSID` values were stripped of unnecessary whitespace, and unwanted SSIDs were removed from the dataset to ensure data quality.
4) **Handling Missing BSSID Values:** Not all Access Points (APs) are heard from every Location Point (LP) due to limited coverage range of Wi-Fi, nearby interfering devices, and the presence of obstacles. These missing BSSID values need to be filled before proceeding with the analysis. The BSSI values range between -11 dBm and -100 dBm. Missing entries were filled with a value of -110 dBm, which is outside the range of expected BSSID values, but serves as a placeholder. This approach is commonly used to handle missing data in Wi-Fi signal strength datasets, as Wi-Fi hotspots or APs of nearby buildings may also be heard during data collection. However, APs with very weak signal strength exhibit poor distance sensitivity and may not contribute meaningfully to the analysis. Furthermore, hotspots are movable and may remain active for a short duration, which can also lead to missing data.

This format provides a clear representation of the essential attributes used in the analysis.

V. Overview of the Modeling Approach

TABLE I
Model Comparison Table

Models	Floor	Block	Area
Decision Tree	92.43	-	-
Random Forest	97.76	49.23	43.54
DNN	-	68.76	65.23
XGBoost	-	47.45	42.21

The dataset was divided into three distinct tasks — predicting **Floor**, **Block**, and **Area**. For each task, several machine learning models were employed, namely **Decision Trees (DT)**, **Random Forest (RF)**, **Extreme Gradient Boosting (XGB)**, and **Deep Neural Networks (DNN)**. To optimize model performance, **Grid Search** was applied to fine-tune the

hyperparameters of Random Forest and XGB for Block and Area predictions, aiming to improve the accuracy.

Below, we analyze each prediction target individually.

A. Floor Prediction

- **Models Used**: Decision Tree (DT), Random Forest (RF)
- **Accuracy Scores**:
  - **Decision Tree (DT)**: 92.43%
  - **Random Forest (RF)**: 97.76%
- The **Random Forest** model demonstrated the highest performance for predicting Floor, with an accuracy of **97.76%**, compared to **92.43%** achieved by the **Decision Tree**. The improvement in Random Forest over Decision Tree can be explained by the following reasons:
  - **Ensemble Technique**: Random Forest is an ensemble of multiple decision trees, which reduces the likelihood of overfitting and ensures better generalization. It tends to capture a variety of data patterns compared to a single decision tree.
  - **Feature Randomness**: Random Forest also selects random features for splitting nodes during tree construction, which leads to lower variance and better robustness against noise in the data.
- The problem of predicting Floor might involve less complex relationships and thus was successfully addressed by tree-based models. The **high accuracy (97.76%) of Random Forest** indicates that the dataset likely had features that could be easily captured by standard decision boundaries, resulting in high classification accuracy. Notably, **DNN** was not used here, which implies that simpler models were sufficient to obtain high accuracy.

B. Block Prediction

- **Models Used**: Random Forest (RF), Extreme Gradient Boosting (XGB), Deep Neural Networks (DNN)
- **Accuracy Scores**:
  - **Random Forest (RF)**: 49.23%
  - **Extreme Gradient Boosting (XGB)**: 47.45%
  - **Deep Neural Networks (DNN)**: 68.76%
- The **DNN** outperformed other models with an accuracy of **68.76%**. This suggests that the prediction of Block involves a **complex, non-linear relationship** between features that is better captured by a neural network with its multi-layer structure, which can learn complex patterns from the data.
- Both Random Forest and XGB struggled to achieve high accuracy, with scores of **49.23% (RF)** and **47.45% (XGB)**, respectively. Despite employing **Grid Search** for hyperparameter tuning, these models were unable to achieve a level of accuracy comparable to the DNN. This indicates that:
  - The **relationship between features** in the dataset was not well represented using simple decision boundaries.
  - **Block** prediction may depend on several interacting variables, which can lead to complex decision surfaces that tree-based models could not effectively capture, even with hyperparameter tuning.

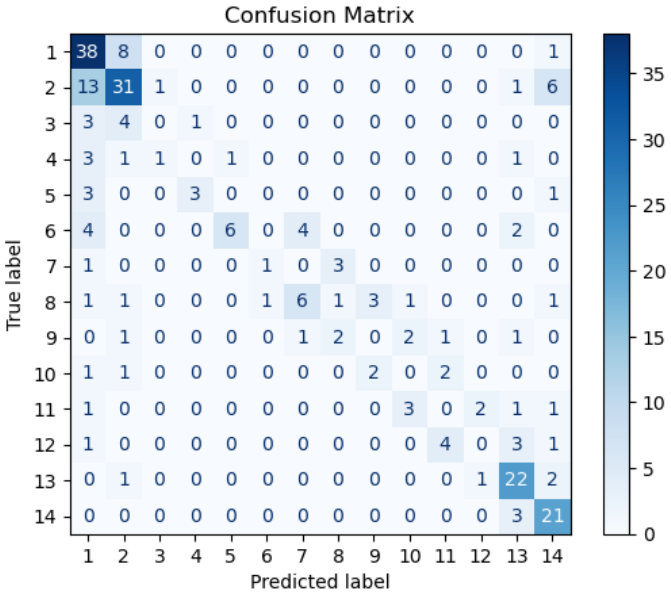
C. Area Prediction

- **Models Used**: Random Forest (RF), Extreme Gradient Boosting (XGB), Deep Neural Networks (DNN)
- **Accuracy Scores**:
  - **Random Forest (RF)**: 43.54%
  - **Extreme Gradient Boosting (XGB)**: 42.21%
  - **Deep Neural Networks (DNN)**: 65.23%
- Again, the **DNN** outperformed the other models, achieving an accuracy of **65.23%**. This indicates that the **Area** variable also involved highly **complex, non-linear relationships**. Neural networks with multiple layers (hidden units) are well-suited for capturing these intricate dependencies, especially if the relationships between features are not straightforward.
- Both Random Forest and XGB achieved similar and relatively low accuracies of **43.54%** and **42.21%**, respectively. This result reflects the **limitations** of tree-based models in capturing the non-linear relationships inherent in this prediction task. These models likely struggled because:
  - **Grid Search** was applied, but the overall model capacity was not enough to match the DNN's performance, as the **interaction effects** and **non-linear dependencies** were likely too intricate.

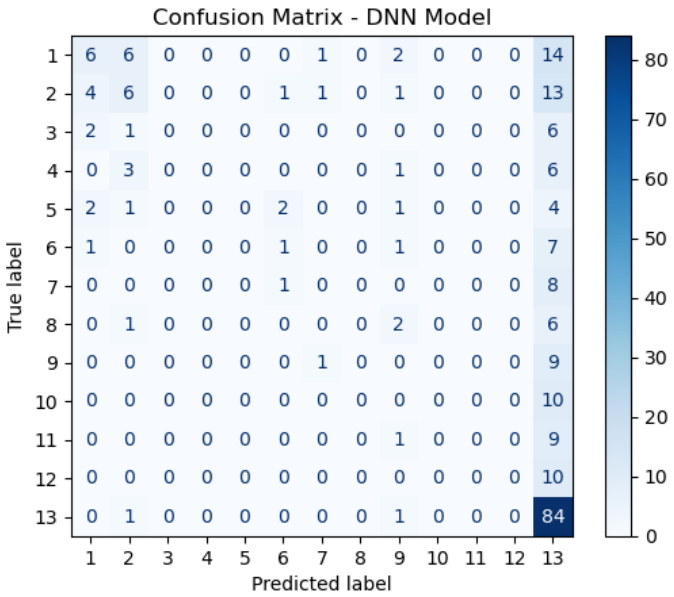
VI. Team Contributions

This project was a collaborative effort by three team members, with each member contributing significantly to various aspects of the work. Below is a detailed breakdown of their individual contributions:

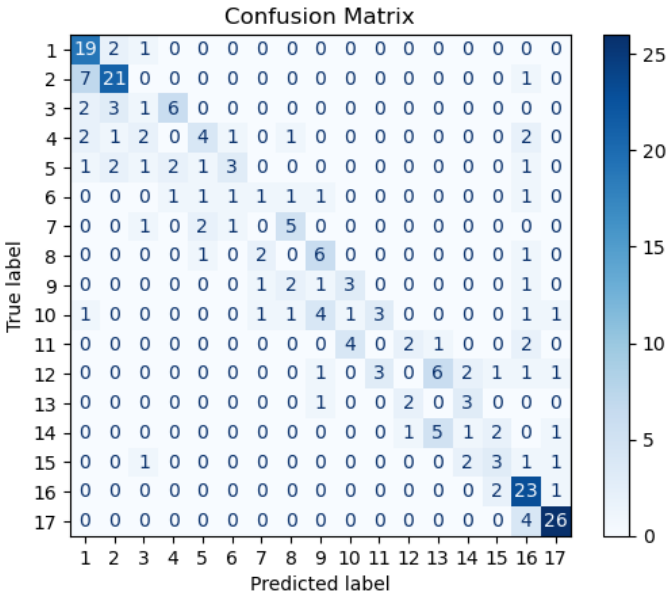
- **Princy Chauhan (202311002)**:
  - Contributed to data collection with other groups.
  - Contributed to the Data Preprocessing and Model Training.
  - Contributed to the documentation of the project.
- **Harsh Mistry (202311003)**:
  - Understood the requirement and data collection strategy and developed the WifiScanner Application.
  - Contributed to data collection with other groups.
  - Contributed to Model Training.
  - Contributed to the documentation of the project.
- **Diti Soni (202311010)**:
  - Contribted to Data preprocessing.
  - Contributed to the statistical analysis of the data.
  - Contributred to Model Training and enhancement of accuracy.
  - Contributed to data collection with other groups.



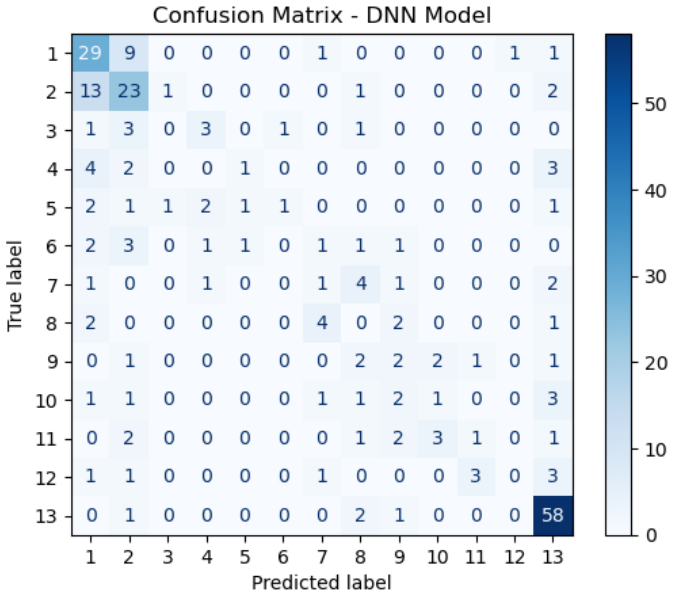
Confusion Matrix of Predicting Area using Random Forest



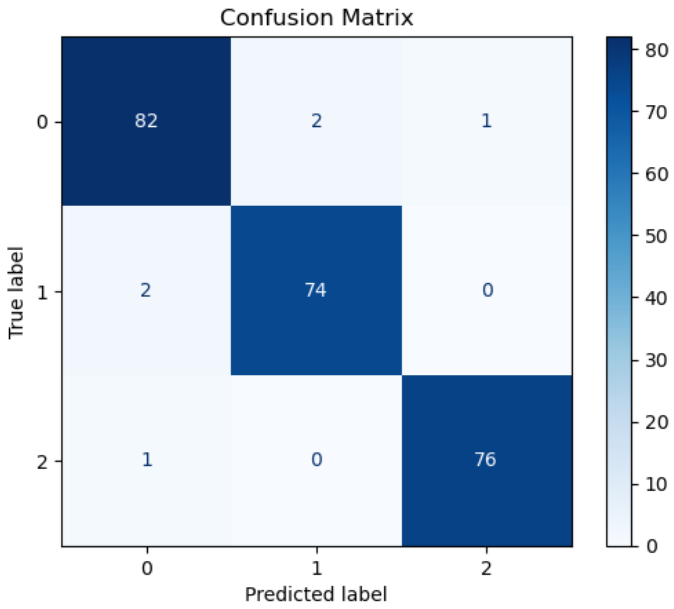
Confusion Matrix of Predicting Area using DNN



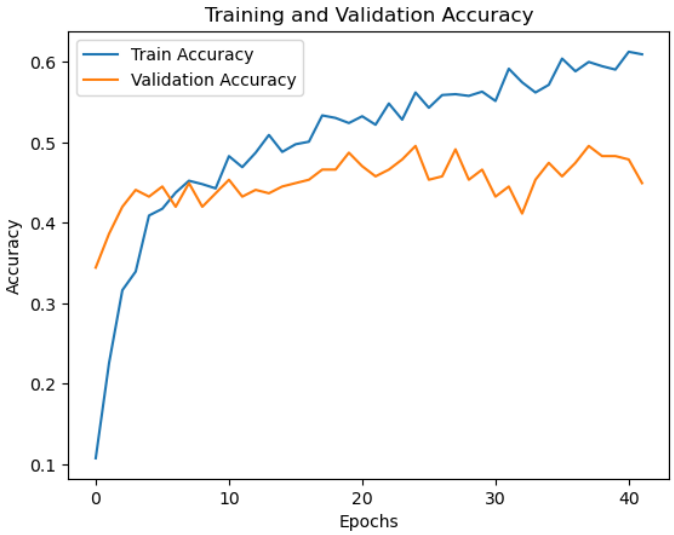
Confusion Matrix of Predicting Block using DNN



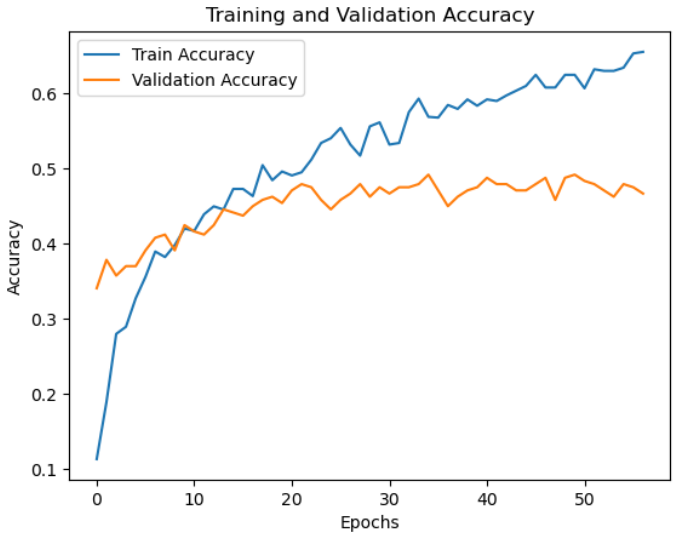
Confusion Matrix of Predicting Area using DNN



Confusion Matrix of Predicting Floor using Random Forest



Accuracy of Block using DNN



Accuracy of Area using DNN

1) WiFiScanner application code: https://github.com/harshmistry3172/WiFiScanner/tree/master.
2) Model code: https://github.com/PrincyChauhan/indoor$_{l}ocalization$.
3) Dataset: https://tinyurl.com/4b89ve47.
4) WiFiScanner APK: https://tinyurl.com/ym5jvyk7.