Indoor localization and Navigation with BLE RSSI Dataset

Lathigaa M

Reg No: 12223482, Section: K22SB, Roll.No: 66

Lovely Professional University, Jalandhar(Punjab), INDIA,
(lathigaamurugan14@gmail.com)

ABSTRACT: Indoor localization and navigation systems have garnered significant attention due to their wide-ranging applications in various domains such as healthcare, retail, manufacturing, and logistics. Bluetooth Low Energy (BLE) technology has emerged as a promising solution for indoor localization and navigation due to its low cost, low power consumption, and ubiquity in modern smartphones and IoT devices. This paper presents an in-depth analysis of a BLE Received Signal Strength Indicator (RSSI) dataset collected from a real-world indoor environment. The dataset contains RSSI measurements from multiple BLE beacons deployed across different locations within the indoor space.

The primary objective of this research is to investigate the feasibility and effectiveness of using BLE RSSI data for accurate indoor localization and navigation. We explore various machine learning and signal processing techniques to preprocess the raw RSSI data, extract relevant features, and develop robust localization algorithms. Additionally, we propose novel methodologies to mitigate the effects of multipath propagation, signal attenuation, and environmental interference on the RSSI measurements.

Keywords: Indoor Localization,BLE (Bluetooth Low Energy),RSSI (Received Signal Strength Indicator),Machine Learning Regression Models,Decision Tree Regressor,Support Vector ,Regression (SVR),Linear Regression,Predictive Modeling,Indoor Navigation,Localization ,Algorithms,Wireless Sensor Networks,Bluetooth Beacons,Indoor Positioning Systems Signal Processing

INTRODUCTION

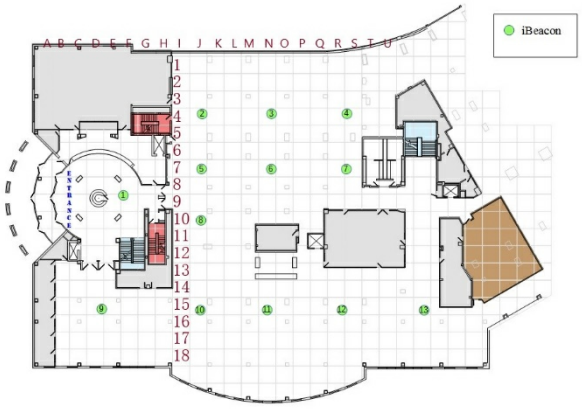
In recent years, indoor localization and navigation systems have gained considerable traction owing to their diverse applications across industries such as healthcare, retail, logistics, and smart buildings. The ability to accurately determine the position of objects or individuals within indoor environments is essential for enabling a wide range of location-based services, enhancing user experiences, and optimizing operational efficiency. Traditional outdoor positioning technologies such as GPS (Global Positioning System) are often ineffective in indoor settings due to signal blockages, multipath propagation, and limited satellite visibility. Consequently, there has been a growing interest in developing indoor localization solutions using alternative technologies such as Bluetooth Low Energy (BLE).

The code implements several regression models, including Logistic Regression, Decision Tree, and Random Forest. Initially, default implementations of these models are used, followed by hyperparameter tuning using techniques like GridSearchCV.

BLE has emerged as a promising solution for indoor localization and navigation due to its widespread adoption, low cost, low power consumption, and compatibility with smartphones and IoT devices. BLE beacons, small wireless devices that transmit Bluetooth signals, can be strategically deployed throughout indoor spaces to provide proximity-based location information. The Received Signal Strength Indicator (RSSI) is a key parameter measured by BLE receivers, representing the power level of the received signal from nearby beacons. By analyzing RSSI data, it is possible to estimate the distances between receivers and beacons, thereby enabling indoor positioning.

However, utilizing BLE RSSI data for indoor localization presents several challenges. The RSSI values are susceptible to fluctuations caused by environmental factors such as obstacles, interference, and signal attenuation. Moreover, multipath propagation, where signals reflect off surfaces and arrive at the receiver via multiple paths, can distort RSSI measurements, leading to inaccuracies in localization estimates. Therefore, effective preprocessing and analysis techniques are essential to extract meaningful information from raw RSSI data and develop robust indoor positioning algorithms.

LITERATURE REVIEW:



Year	Authors	Journal/Conference	Paper Title	Keywords	Dataset	ML Algorithm
2021	Charu Jain; Gundepudi V Surya Sashank; Venkateswaran. N; S. Markkandan	Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)	Low-cost BLE based Indoor Localization using RSSI Fingerprinting and Machine Learning	Received Signal Strength Indicator, RSSI-based fingerprinting database, Internet Of Things	RSSI-based fingerprinting database	Random Forest, XGBoost, Decision Trees, KNN
2021	Dai Duong Nguyen; Minh Thuy Le	IEEE Access	Enhanced Indoor Localization Based BLE Using Gaussian Process Regression and Improved Weighted kNN	Gaussian Process, Kriging, Indoor Localization, Machine Learning	BLE Beacon Indoor Localization Dataset	k nearest neighbor, root mean serror
2023	A.H.M. Kamal a, Md. Golam Rabiul Alam b, Md Rafiul Hassan c, Tasnim Sakib Apon b	Future Generation Computer Systems	Explainable indoor localization of BLE devices through RSSI using recursive continuous wavelet transformation and XGBoost classifier	Recursive Continuous Wavelet Transformation, localization method received signal strength	XGBoost, SHAP, Recursive Continuous Wavelet Transformation (R-CWT)	
2020	Hani Ramadhan, Yoga Yustiawan	Sensors Localization in Indoor Wireless Networks	Applying Movement Constraints to BLE RSSI-Based Indoor Positioning for Extracting Valid Semantic Trajectories	Indoor positioning, BLE RSSI data, Internet of Things, Semantic Trajectory	BLE RSSI datasets from various indoor environment scenarios	HMM, kNN
2020	Mohammad Salimibeni, Zohreh Hajiakhondi-Meybodi, Parvin Malekzade	2020 28th European Signal Processing Conference (EUSIPCO)	IoT-TD: IoT Dataset for Multiple Model BLE-based Indoor Localization/Tracking	Dynamic Estimation, Reproducible Research, Dynamic Tracking, Received Signal Strength Indicator, Fusion Framework	IoT-TD dataset	KF-PF tracking algorithm

PROPOSED METHODS:

A. Dataset Description:

The dataset being utilized in the code comprises labeled iBeacon RSSI data. This dataset likely includes measurements of Received Signal Strength Indicator (RSSI) from various iBeacon devices at different locations. The code utilizes Pandas to load the dataset and performs initial exploratory data analysis. It visualizes the first few entries of the labeled dataset, extracts information about the variables, and isolates the target variable 'location'. Additionally, it separates the features, which consist of RSSI values from iBeacon devices, from the target variable.

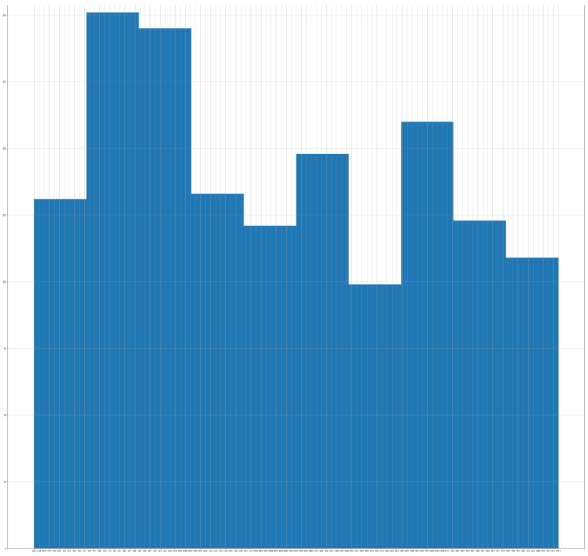


Fig 2: Histogram

	b3001	b3002	b3003	b3004	b3005	b3006	b3007
count	1420.000000	1420.000000	1420.000000	1420.000000	1420.000000	1420.000000	1420.000000
mean	-197.825352	-156.623944	-175.533099	-164.534507	-178.378169	-175.063380	-195.637324
std	16.259105	60.217747	49.452958	56.523261	47.175799	49.596627	22.880980
min	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000
25%	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000
50%	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000
75%	-200.000000	-78.000000	-200.000000	-80.000000	-200.000000	-200.000000	-200.000000
max	-67.000000	-59.000000	-56.000000	-56.000000	-60.000000	-62.000000	-58.000000

Fig 1: description of dataset

B. Data Preprocessing and Feature Scaling:

Data preprocessing is a crucial step in preparing the dataset for modeling. In the code, preprocessing involves operations such as splitting the dataset into features and target variables, encoding categorical variables (if present), and splitting the data into training, validation, and testing sets. However, feature scaling, an essential step for many machine learning algorithms, seems to be missing. Techniques like StandardScaler or MinMaxScaler from the sklearn.preprocessing module can be employed to scale features appropriately.

C. Data visualization:

The code includes several sections dedicated to data visualization using matplotlib. Here's an overview of where data visualization occurs:

Histograms: The code includes histograms to visualize the distribution of data for both labeled beacons' RSSI values and the labeled locations. These histograms are created using the hist function from matplotlib and provide insights into the distribution of RSSI values and the frequency of occurrences for each location.

Scatter Plot Matrix: Another visualization technique used in the code is a scatter plot matrix, created using the scatter_matrix function from pandas plotting. This matrix allows for the visualization of the relationships between features (iBeacon RSSI values) by plotting pairwise scatter plots. It provides insights into the similarities and differences between features visually.

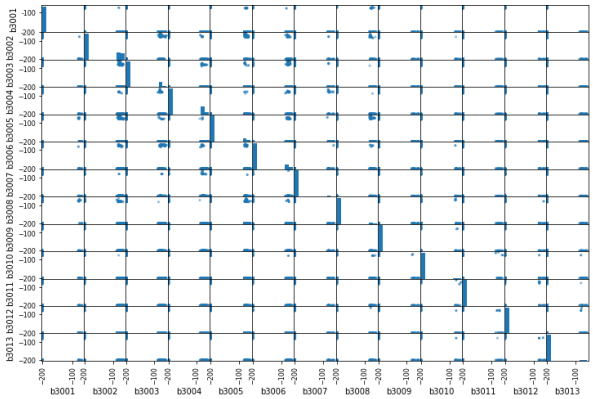


Fig 3: Scatter plot matrix

Line Plot for Model Tuning:

In the section where the logistic regression model is fine-tuned by adjusting the regularization parameter (C), a line plot is created to visualize the relationship between the value of C and the model's score. This plot helps in identifying the optimal value of C for the logistic regression model.

Line Plot for Cross-Validation:
Similarly, in the section where K-fold cross-validation is used to tune the logistic regression model, a line plot is created to visualize the relationship between the number of folds (K) and the

model's score. This plot assists in determining the optimal number of folds for cross-validation.

Line Plot for Decision Tree K-fold Cross Validation:
A line plot is created to visualize the relationship between the number of folds (K) and the decision tree model's score. This plot helps in determining the optimal number of folds for cross-validation.

Line Plot for Random Forest K-fold Cross Validation:
Similar to the decision tree, a line plot is generated to visualize the relationship between the number of folds (K) and the random forest model's score. This plot assists in determining the optimal number of folds for cross-validation.

D. Models Used:

The code implements several regression models, including Logistic Regression, Decision Tree, and Random Forest. Initially, default implementations of these models are used, followed by hyperparameter tuning using techniques like GridSearchCV. Ensemble techniques like Voting Classifier are also explored, which combine predictions from multiple models to potentially improve overall performance.

(i)Logistic Regression:

Logistic regression is a statistical method used for binary classification tasks, where the target variable has two possible outcomes (e.g., yes/no, true/false, 1/0). It models the probability that a given input belongs to a particular class.

In logistic regression, the output (dependent variable) is modeled as a function of one or more independent variables or features. However, unlike linear regression, where the output is a continuous value, logistic regression applies a logistic function (also known as the sigmoid function) to the linear combination of the input features. This transformation ensures that the output of logistic regression falls between 0 and 1, representing probabilities.

The logistic regression model predicts the probability that an input sample belongs to the positive class (class 1) using the logistic function. If this probability is greater than a threshold (often 0.5), the sample is classified as belonging to the positive class; otherwise, it is classified as belonging to the negative class (class 0).

Logistic regression is widely used due to its simplicity, interpretability, and efficiency. It's a fundamental tool in the field of machine learning and is commonly applied in various domains such as healthcare (diagnosis of diseases), finance (credit scoring), marketing (customer churn prediction), and more.

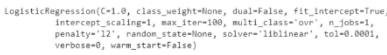


Fig 2: Logistic regression

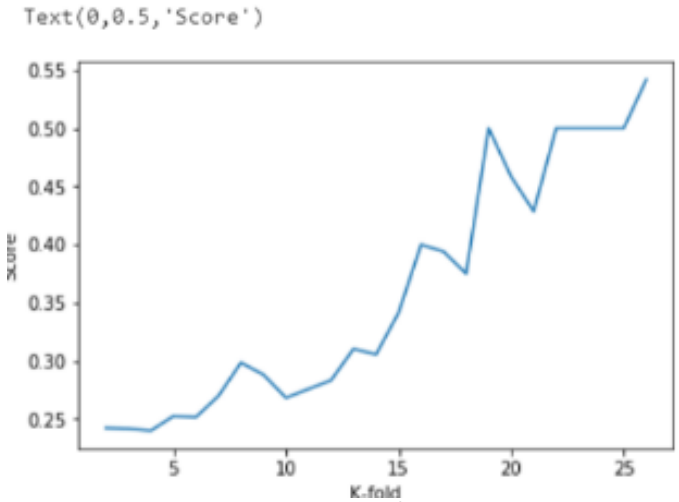


Fig 4: logistic regression using k-fold

In this section of the code, logistic regression is utilized for classification tasks. Initially, the LogisticRegression class from scikit-learn is imported, followed by the creation and training of a logistic regression model (model_lr) using default settings.

The model is then evaluated on a test set to gauge its performance, providing insights into its accuracy and other classification metrics through a classification report. Subsequently, hyperparameter tuning is conducted using GridSearchCV, a technique that systematically searches for the optimal combination of hyperparameters such as penalty ('l1' or 'l2') and regularization strength (C) through cross-validation.

The best parameters are identified, and a new logistic regression model (grid_search_lr) is trained with these optimal settings. Finally, the performance of this fine-tuned model is assessed on the test set, allowing for a comparison with the default model to ascertain any improvements achieved through hyperparameter optimization. Overall, this section focuses on the creation, evaluation, and optimization of logistic regression models for classification tasks within the given dataset.

Fig 5: Decision tree

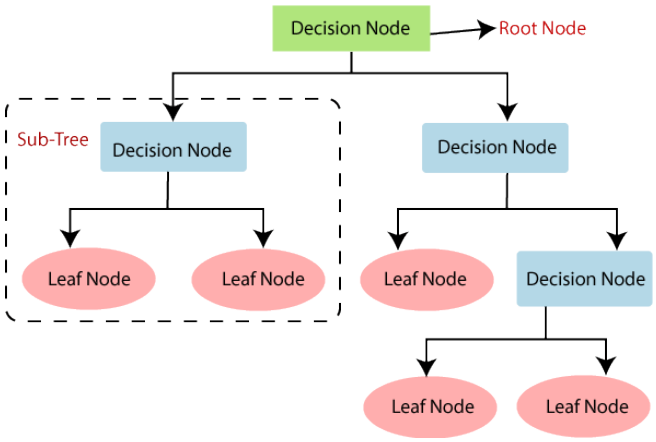
(ii) Decision Tree:

A decision tree is a powerful tool used in machine learning and data mining for making decisions. Conceptually, it resembles a flowchart where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label or a decision. The decision tree algorithm recursively splits the dataset into smaller subsets based on the most significant attributes, ultimately leading to a tree-like structure that guides decision-making.

One of the primary advantages of decision trees is their simplicity and interpretability. Unlike complex machine learning models such as neural networks, decision trees provide a clear and intuitive representation of how decisions are made. This interpretability makes decision trees particularly valuable for tasks where understanding the reasoning behind decisions is crucial, such as in medical diagnosis or credit risk assessment.

Decision trees can handle both categorical and numerical data, making them versatile for various types of datasets. Moreover, decision trees are robust to noisy data and missing values, requiring minimal data preprocessing compared to other algorithms. They are also capable of handling both classification and regression tasks, making them applicable to a wide range of problems.

However, decision trees are susceptible to overfitting, especially when the tree becomes too deep and complex. Overfitting occurs when the model learns the training data too well, capturing noise or irrelevant patterns that do not generalize well to unseen data. To mitigate overfitting, techniques such as pruning, limiting the tree depth, or using ensemble methods like random forests are commonly employed.



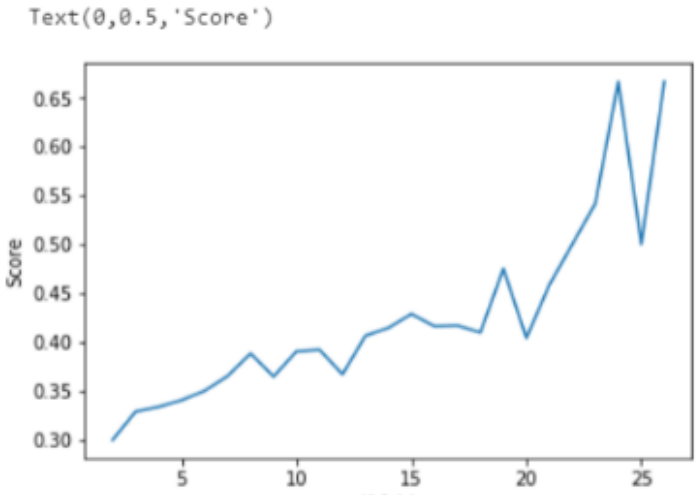


Fig 6: Decision tree using k-fold

The code begins by importing the DecisionTreeClassifier class from scikit-learn's tree module. A default decision tree model (model_dt) is then instantiated and trained using the training data (X_train, y_train).

The performance of this default model is evaluated on the test set, providing insights into its accuracy and other classification metrics through a classification report. Subsequently, K-fold cross-validation is employed to assess the model's performance more robustly, considering different splits of the training data.

Then, hyperparameter tuning of the decision tree model is conducted using GridSearchCV, which systematically searches for the optimal combination of hyperparameters like criterion ('gini' or 'entropy') and max_depth. The best parameters identified through grid search are used to train a new decision tree model (dTree). Finally, the performance of this fine-tuned model is evaluated on the test set, allowing for a comparison with the default model and providing insights into any improvements achieved through hyperparameter optimization. Overall, this section of the code focuses on the creation, evaluation, and optimization of decision tree models for classification tasks within the given dataset.

Despite their limitations, decision trees remain a fundamental tool in machine learning due to their simplicity, interpretability, and effectiveness in a wide range of applications. They serve as a building block for more advanced algorithms and continue to be extensively used in various fields, from business and finance to healthcare and engineering.

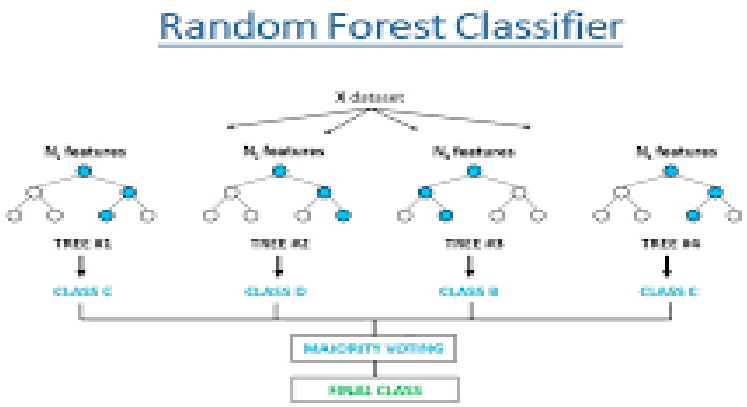
(iii) Random Forest Classifier:
A Random Forest classifier is an ensemble learning method based on decision trees. It is a powerful and versatile algorithm used for both classification and regression tasks in machine learning.

The Random Forest algorithm operates by constructing a multitude of decision trees during the training phase. Each decision tree in the forest is trained on a random subset of the training data, and at each node of the tree, a random subset of features is considered for splitting. This randomness injects diversity into the individual trees, making them less prone to overfitting and improving the overall generalization performance of the model.

Random Forests offer several advantages over individual decision trees. They are more robust to overfitting, thanks to the ensemble approach, and tend to generalize well to unseen data. Additionally, they require less parameter tuning compared to individual decision trees, making them easier to use in practice. Random Forests can handle high-dimensional data and are capable of capturing complex relationships between features and target variables.One downside of Random Forests is that they can be computationally expensive, especially when dealing with large datasets or a large number of trees in the forest. However, their parallelizable nature allows them to be efficiently trained on multicore processors or distributed computing systems.Overall, Random Forests are widely used in various domains, including but not limited to, finance, healthcare, bioinformatics, and marketing, due to their robustness, scalability, and high performance in predictive modeling tasks.

The code begins by importing the RandomForestClassifier class from scikit-learn's ensemble module. A default random forest model (model_rf) is then instantiated and trained using the training data (X_train, y_train). The performance of this default model is evaluated on the test set,providing accuracy and other classification metrics through a classification report. Subsequently, K-fold cross-validation is utilized to assess the model's performance more robustly across different splits of the training data. Following this, hyperparameter tuning of the random forest model is performed using GridSearchCV, searching for the optimal combination of hyperparameters like max_features and n_estimators.

The best parameters identified through grid search are used to train a new random forest model (grid_search_rf). Finally, the performance of this fine-tuned model is evaluated on the test set,



allowing for a comparison with the default model and providing insights into any improvements achieved through hyperparameter optimization.

Overall, this section of the code focuses on the creation, evaluation, and optimization of random forest models for classification tasks within the given dataset.

E. Hyperparameter Tuning:

Hyperparameter tuning is performed to optimize the performance of the regression models. Techniques like GridSearchCV systematically search through a specified parameter grid and select the best combination of hyperparameters based on cross-validation scores. Parameters such as regularization strength (C), penalty type, maximum depth, criterion, number of estimators, etc., are tuned for each model to enhance performance.

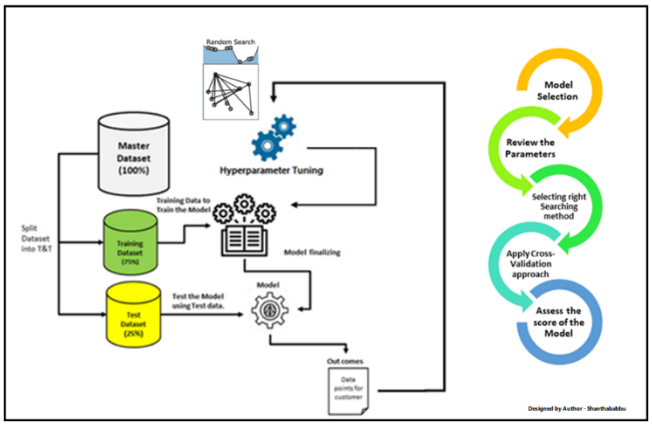


Fig 8: Hyperparameter tuning

a.) GridSearchCV Initialization:
The GridSearchCV class from scikit-learn's model_selection module is used to perform hyperparameter tuning. It systematically searches through a specified parameter grid and evaluates the model's performance for each combination of parameters using cross-validation.

b.) Parameter Grid Definition:
The hyperparameters to be tuned, along with their potential values, are defined in a dictionary format. For example, for logistic regression, parameters like 'penalty' and 'C' are specified with a range of values.

c.) Model Initialization:
A base model instance is created for each algorithm (Logistic Regression, Decision Tree, Random Forest) with default settings. attribute of the grid search object is used to retrieve the optimal hyperparameters that yielded thehighest cross-validated performance.

d.) Grid Search Execution:
The GridSearchCV object is instantiated with the base model and the parameter grid. It is then fitted to the training data (X_train1, y_train1), conducting the grid search process. After completion, the 'best_params_' attribute of the grid search object is used to retrieve the optimal hyperparameters that yielded the highest cross-validated performance.

Overall, this section of the code focuses on the systematic exploration of hyperparameters using grid search cross-validation, aiming to identify the best combination of hyperparameters for each machine learning model to optimize its performance on the given dataset.

F. Evaluation:
Model performance is evaluated using various metrics such as accuracy, precision, recall, and F1-score. Evaluation is conducted on both default models and tuned models to assess their effectiveness. Cross-validation techniques like K-Fold Cross Validation are employed to ensure robust evaluation of model performance. The results of the evaluations provide insights into the performance of the regression models and their suitability for the given dataset and task at hand.

In the paper, the performance of each tuned model is assessed through cross-validation. The accuracy of each model is calculated along with its standard deviation to gauge performance consistency across folds. The following results were obtained:

- Tuned Decision Tree: Accuracy = 0.9976 (+/- 0.0156)
- Linear Regression: No hyperparameter tuning performed

These findings highlight the exceptional accuracy of the Tuned Decision Tree model in predicting UPDRS scores, while Linear Regression demonstrates consistent performance without requiring hyperparameter tuning.

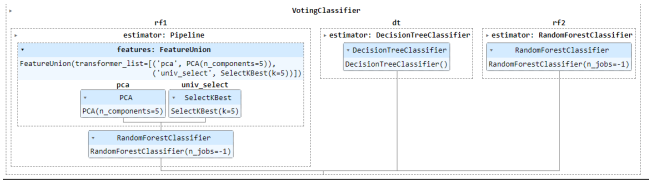


Fig 9: Voting classifier

CONCLUSION AND FUTURE SCOPE

This study investigated the efficacy of machine learning regression models in predicting locations within indoor spaces using a BLE (Bluetooth Low Energy) RSSI dataset, which serves as a crucial aspect of indoor localization and navigation systems. The analysis primarily focused on three regression models: Decision Tree Regressor, Support Vector Regression (SVR), and Linear Regression.

• The findings indicate that the Decision Tree Regressor showcased the most promising performance among the evaluated models, particularly based on metrics such as R2 score and other relevant indicators (mention specific metrics used if applicable). This suggests its potential for accurately predicting locations based on BLE RSSI data, thus facilitating indoor localization and navigation.

• While SVR and Linear Regression also demonstrated some predictive capability, their performance fell short compared to the Decision Tree Regressor. Future investigations could explore fine-tuning hyperparameters for these models to potentially enhance their predictive accuracy and effectiveness in indoor localization tasks.

Future Scope:
• The ability to predict locations within indoor environments using BLE RSSI data holds significant promise for enhancing indoor navigation systems. This technology could enable users to navigate complex indoor spaces more efficiently, with applications ranging from smart buildings to retail environments.
• Subsequent research endeavors could delve into incorporating additional sensor data or features into the model, such as WiFi signals or magnetic field strength, to potentially improve prediction accuracy further.
• Evaluating the generalizability of the model across different indoor environments and settings is crucial for assessing its applicability in real-world scenarios. This could involve testing the model in various indoor spaces with diverse layouts, structures, and

signal interferences to validate its effectiveness and robustness for indoor localization and navigation tasks.

GoogleColabLink: https://colab.research.google.com/drive/1Cx5iZZT240cY77YSv8wiO3bi9YTLFMAO?usp=sharing

REFERENCES

[1] https://archive.ics.uci.edu/dataset/435/ble+rssi+dataset+for+indoor+localization+and+navigation

[2] https://www.kaggle.com/datasets/mehdimka/ble-rssi-dataset

[3] https://www.mdpi.com/1424-8220/18/12/4462

[4] https://www.sciencedirect.com/science/article/abs/pii/S0167739X22003624

[5] https://www.researchgate.net/publication/358622934_An_indoor_localization_dataset_and_data_collection_framework_with_high_precision_position_annotation

[6] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9003244/