****

| **Course :** | **CSC3105 Data Analytics** |
| --- | --- |
| **Topic :** | **Data Analytics in Line-of-Sight (LOS) & Non-Line-of-Sight (NLOS) Wireless Signal Prediction in Indoor Environment for Precise Localisation** |

| **Name** | **Individual contribution** | **GUID** | **SIT ID** |
| --- | --- | --- | --- |
| Nur Khadijah Afreen Binte Mohamed Jahabar Sadik | LOS/NLOS Classification, Range Prediction | **2957846B** | **2301124** |
| Toh Hao Yi | LOS/NLOS Classification, Range Prediction | **2957911T** | **2303340** |
| Ignatius Lim Zheng Liang | Data Preprocessing | **2957928L** | **2303403** |
| Tan Ting Xuan, Daryl | Data Preprocessing | **2957974T** | **2301119** |
| Choo Jin Quan | Unsupervised Machine Learning Experimentation, Range Prediction | **2957855C** | **2300892** |

## Abstract

This project explores the classification of Line-of-Sight (LOS) and Non-Line-of-Sight (NLOS) Ultra-Wideband (UWB) wireless signals and the prediction of measured range for LOS paths in indoor environments—an essential step toward improving localisation accuracy where GPS is unreliable. The solution adopts a supervised machine learning approach using statistical features extracted from Channel Impulse Response (CIR) data. For LOS/NLOS classification, both Random Forest and Multi-Layer Perceptron (MLP) classifiers were trained and evaluated using metrics like accuracy, precision, recall, and F1-score. For range prediction, a Random Forest Regressor was used and assessed with MAE, RMSE, and R² metrics. Key preprocessing steps included feature engineering, dimensionality reduction via mutual information and correlation analysis, SMOTE for class balancing, and Z-score normalization. Visualisation tools such as correlation heatmaps and prediction scatter plots further supported model analysis. Results demonstrated high classification accuracy (~92%) and effective range prediction, with important insights into feature importance, especially the role of RXPACC. This dual-model pipeline contributes to robust indoor localisation by addressing both signal classification and distance estimation.

**TABLE OF CONTENTS**

[**Abstract 1**](#_5b1bynb8le3q)

[**Problem Definition 4**](#_siggvetxldqj)

[**Problem Analysis and Objectives 4**](#_prfe6f41j5yf)

[**Algorithm 4**](#_hrizysu2vl9d)

[Data Preprocessing 4](#_kf2m3il8sumb)

[Supervised Vs Unsupervised Machine Learning 5](#_cp5rn8eqve9w)

[Data Mining: Random Forest Classifier (Objective 1) 5](#_e4dp8n4dqsh2)

[Data Mining: Neural Network Model (Objective 1) 6](#_rpye5p634a5o)

[Data Visualisation: Correlation Heatmap (Objective 1) 6](#_ldviuel9bgzf)

[Data Mining: Regression (Objective 2) 6](#_70uri1frpc28)

[Data Visualisation: Random Forest Regressor (Objective 2) 6](#_d4eh337uhvzk)

[**Plots and Results 7**](#_30a5tmwziauk)

[Objective 1: Classification of LOS/NLOS 7](#_whk6jw3ragad)

[Class Imbalance & SMOTE 7](#_syomuobecx57)

[Performance Metrics 7](#_xkxfwac973z1)

[Logical Analysis 8](#_xovbvppaaatt)

[Feature Selection and Impact on Performance 9](#_tisqc42lz3gs)

[Objective 2: Predicting the measured range for LOS path 10](#_wz3dwqvxznz)

[Performance Metrics 10](#_zeagqmy0cf23)

[Non-Line-of-Sight 11](#_d5ffdtghciz7)

[Logical Analysis 11](#_webo70u5kv9)

[**Overall Observations and Patterns 12**](#_vsg6g13vihdz)

[**Data Visualisation and Results Analysis 13**](#_ews461tm50zs)

[Objective 1: Classification of LOS/NLOS 13](#_ym1g8bnhg2es)

[Correlation Heatmap 13](#_fvup27daahbr)

[Objective 2: Predicting the measured range for LOS path 14](#_nfst4r6ynett)

[Actual Vs Predicted Range 14](#_pxp1xaq8xsvd)

[Actual Vs Predicted Range (colour-coded by RXPACC) 15](#_b2oh95d2pdut)

[**Interesting Aspects of Solution (Assumptions/Optimisations) 15**](#_mvg03ktlg4zd)

[An Alternative Approach: Unsupervised Machine Learning (Clustering) 15](#_su2qdvoy6bmd)

[Feature Importance (Decision Tree Classifier) 20](#_461nuarvgplp)

[Assumptions 20](#_z0xiq22p6ohy)

[**Optimisations 21**](#_ixb6vu53xqbw)

[**Conclusion 21**](#_4nkrqokpobiv)

[**Appendix 22**](#_51cr7ua9z00k)

[**A: Pseudocode 22**](#_846qlwrg5xpy)

[Import libraries 22](#_50g6ymraltby)

[Load and combine CSV files 22](#_y0eovyhk0b14)

[Feature engineering on CIR columns 22](#_8hm4268u7i1q)

[Data cleaning 23](#_lc0hofm33ogm)

[Feature selection 23](#_pxp4ry7n7aqv)

[Feature scaling 24](#_9pnpt16ra244)

[Class imbalance handling (SMOTE) 24](#_ikgxmrgc7m8b)

[Train-Test split (Classification task) 24](#_twss3d2d8g7p)

[Train Random Forest Classifier 24](#_y83q3igd7h6k)

[Train Neural Network Classifier (MLP) 25](#_emhy7j6zkmle)

[Correlation Heatmap Visualisation 25](#_pdo220p7rhq8)

[Train-Test split (Regression task) 25](#_szq1cmarmgkn)

[Train Random Forest Regressor 26](#_ebudtrc46aes)

[Actual vs Predicted Visualisation 26](#_eg5ecd9wo3lj)

[Actual vs Predicted Visualisation (coloured RXPACC) 26](#_1lyl277t1hkf)

[B: Source Code 27](#_9at7low9ws2p)

## Problem Definition

This project focuses on Line-of-Sight (LOS) and Non-Line-of-Sight (NLOS) classification in Ultra-Wideband (UWB) wireless signals for precise indoor localisation. This is important because GPS does not work well indoors due to signal interference and alternative technologies like Wi-Fi and Bluetooth (which are used for indoor positioning) suffer from signal degradation when obstacles like walls obstruct the path. NLOS conditions introduce errors in distance

estimation, which leads to inaccurate location tracking. However, precise localisation is important for a myriad of applications such as autonomous vehicles, robotics and smart tracking systems.

## Problem Analysis and Objectives

The proposed solution is a supervised learning solution that uses CIR measurements to fulfil two objectives.

1. Recognise whether a pair of CIR measurements is LOS/NLOS
2. Predict the measured range for the LOS path for accurate localisation.

The novelty lies in combining statistical features from the CIR with traditional signal attributes, applying both classical ML (Random Forest) and deep learning (MLP) models for robustness.

The approach taken in the solution involves data cleaning, model training with supervised machine learning, as well as evaluation using key performance metrics and analysis of insights.

## Algorithm

The pipeline was designed to address both classification (LOS vs NLOS detection) and regression (range prediction for LOS samples). It incorporates all the key stages of the data analytics lifecycle: data preparation, data mining, data visualisation and data analysis.

### Data Preprocessing

The pipeline begins with loading the datasets and storing the data into a dataframe for data preparation. Several steps are taken to reduce the size of the dataset, the first of which is to calculate statistics like mean, median, standard deviation and skewness of the CIR columns. Once the statistics are derived, all of the CIR columns are dropped.

The dataset is further reduced by performing the following:

1. Duplicate rows are dropped
2. Rows with missing values are dropped as the median cannot be computed
3. Through domain expertise, “BITRATE”, “PRFR” and “CH” features were identified to be constant columns with no variation in data. These constant columns are dropped.

After dropping this information, some formatting is done to make all the remaining features numeric.

The next step would be feature selection, and it involves the computation of Mutual Information (MI) scores with “NLOS” as the target feature. The MI scores are sorted by importance and the top 75% of the most important features are kept, with the rest being dropped. A correlation analysis is also performed to remove features with high correlation values (greater than 0.8). Low variance filtering is also performed to remove near-constant features. For logging and debugging purposes, the removed features are printed in the terminal.

The features are then standardised using Z-score normalisation, transforming them to have zero mean and unit variance. This is important for neural networks, which comes later in the implementation. Following normalization, class imbalance was assessed by examining the distribution of LOS and NLOS samples, and logic was included to handle imbalance using the Synthetic Minority Oversampling Technique (SMOTE).

Before classification, the dataset is split into training and testing sets using stratified sampling with an 80/20 ratio. Given that the dataset contains 42,000 samples, the 8,400 test samples are more than sufficient to reliably evaluate model performance. Furthermore, allocating more data to training (33,600 samples) allows the model to learn better and generalize more effectively to unseen data. Stratified sampling also ensures that the class balance is maintained after splitting. Before predicting the range, the data needs to be split into LOS and NLOS samples. This allows us to train separate models for LOS and NLOS scenarios.

### Supervised Vs Unsupervised Machine Learning

Since the task involves predicting a known target variable—specifically, classifying whether a signal is LOS or NLOS (classification) and predicting the range for LOS samples (regression)—a supervised machine learning approach was chosen. Supervised learning is appropriate because the dataset includes labeled outcomes for both tasks, allowing models to learn a direct mapping from features to targets. Unsupervised techniques such as clustering were not suitable in this context, as they do not leverage the existing labels and would not directly support the prediction goals of the project.

### Data Mining: Random Forest Classifier (Objective 1)

For the first objective of recognising whether a pair of CIR measurements is LOS or NLOS, a Random Forest classifier was trained on the training set and evaluated on the test set. The model’s performance was assessed using several metrics, including accuracy, a confusion matrix and a classification report (which provides precision, recall and F1 score for each class). Additionally, a feature importance analysis was done to identify which features contributed the most to the model’s decision-making. After analysis of a bar chart that plots the feature importance, features with low importance scores are dropped to further reduce the dataset. After the removal of features with low importance scores, the Random Forest classifier is trained again under the same conditions as before to get an updated set of performance metrics.

### Data Mining: Neural Network Model (Objective 1)

As an alternative to the Random Forest classifier, a Neural Network model was also implemented using a Multi-Layer Perceptron (MLP) classifier. The architecture consisted of two hidden layers with 100 and 50 neurons respectively, and the model was trained for a maximum of 1000 iterations to ensure convergence. The neural network was trained on the same preprocessed training data and evaluated using the same test set for consistency. Its performance was assessed using the same evaluation metrics: accuracy, a confusion matrix, and a classification report. The findings from the comparison of the two methods will be discussed in the “Plots and Results” section.

### Data Visualisation: Correlation Heatmap (Objective 1)

To further understand the relationships between the features, a correlation heatmap was generated using the scaled dataset. The heatmap provides a visual representation of the pairwise correlations between all selected features, with values ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation). From the heatmap, features with high positive or negative correlations can be identified. Highly correlated features may indicate redundancy, which could potentially lead to multicollinearity issues in certain models or unnecessary complexity. Identifying such relationships is valuable when considering feature selection, dimensionality reduction, or when choosing algorithms that are sensitive to correlated inputs.

### Data Mining: Regression (Objective 2)

For the second objective of predicting the measured range for the LOS path for accurate localisation, the dataset was first split into NLOS and LOS data. Both datasets were trained separately using the same data preparation steps and train-test split (80/20), ensuring consistency in training.

Four different regression models were used:

1. K-nearest Neighbours Regression
2. XGBoost Regression
3. LightGBM Regression
4. Random Forest Regression

The target variable of this regression task was the RANGE feature, and features related to the NLOS classification were excluded to prevent data leakage.

### Data Visualisation: Random Forest Regressor (Objective 2)

A scatter plot of actual vs predicted range values was also generated to visually access the regression model’s performance. Points close to the diagonal red line indicate accurate predictions. The findings will be further discussed in the “Plots and Results” section.

## Plots and Results

### Objective 1: Classification of LOS/NLOS

#### Class Imbalance & SMOTE

Although the provided dataset was already balanced, the pipeline included logic to handle imbalance using the Synthetic Minority Oversampling Technique (SMOTE). When applied, SMOTE generates synthetic examples of the minority class (either LOS or NLOS), thereby ensuring that both classes have a comparable number of samples. This helps prevent model bias toward the majority class and improves generalisation in classification tasks, in case an imbalance was present in the dataset.

#### Performance Metrics

The metrics used for the classification are as follows:

1. Accuracy  
   Proportion of total predictions the model made correctly.
2. Precision  
   Out of all the samples predicted as positive, how many are actually positive? High precision means few false positives.
3. Recall  
   Out of all the actual positive samples, how many were correctly predicted? High recall means few false negatives.
4. F1-score  
   Harmonic mean of precision and recall. High F1 score means good balance between precision and recall.

The findings are documented in the following table.

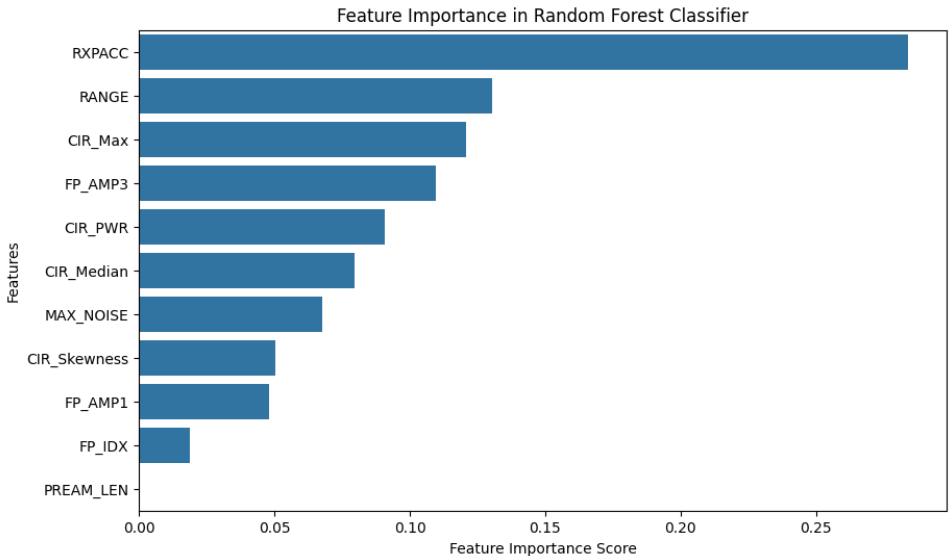
| **Model** | **LOS/NLOS** | **Accuracy** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- | --- | --- |
| Random Forest Classifier | LOS | 0.92 | 0.91 | 0.93 | 0.92 |
| NLOS | 0.93 | 0.91 | 0.92 |
| Neural Network Classifier | LOS | 0.92 | 0.90 | 0.94 | 0.92 |
| NLOS | 0.93 | 0.90 | 0.92 |

| **Confusion Matrix  (Random Forest Classifier)** | **Confusion Matrix  (Neural Network Classifier)** |
| --- | --- |
|  |  |

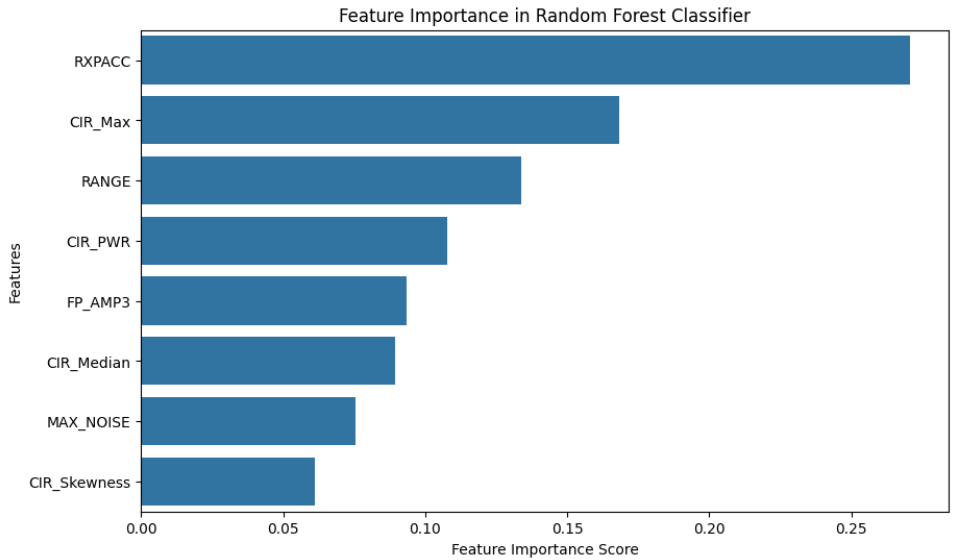
#### Logical Analysis

Both Random Forest Classifier and Neural Network Classifier achieved identical accuracy (92%), indicating strong overall performance. However, the Random Forest model recorded 394 false positives (NLOS misclassified as LOS) and 258 false negatives (LOS misclassified as NLOS), while the Neural Network had 378 false positives and 288 false negatives. This means the Neural Network made fewer false positives, suggesting better precision for NLOS classification, whereas the Random Forest made fewer false negatives, indicating better recall for LOS classification. Overall, while both models are comparable, the Random Forest slightly outperforms in total error count (652 vs 666), but the Neural Network provides a more balanced error distribution. The moderate count of false positives and negatives could be attributed to an overlap in signal features between LOS and NLOS, as real-world UWB signals often have similar characteristics due to reflections or multi-path effects. Overall, the Random Forest Classifier generalises slightly better as it has a lower total error count.

#### Feature Selection and Impact on Performance



The RXPACC feature was ranked to be the most important feature in the classification of LOS/NLOS signals. On the other hand, features like FP\_AMP1, FP\_IDX and PREAM\_LEN were determined to be of little importance in the classification. Through this ranking of feature importance, the aforementioned 3 features were dropped, resulting in the final ranking of:



| Metric | Before Removal | After Removal |
| --- | --- | --- |
| Accuracy | 91.73% | 91.56% |
| LOS Precision | 0.90 | 0.90 |
| LOS Recall | 0.93 | 0.94 |
| LOS F1-score | 0.92 | 0.92 |
| NLOS Precision | 0.93 | 0.93 |
| NLOS Recall | 0.90 | 0.90 |
| NLOS F1-score | 0.92 | 0.91 |

After the removal of low-importance features, the accuracy remained almost identical, with a minor drop of 0.17%. There was a slight performance improvement in LOS recall (0.94 vs 0.93) and a slight drop in NLOS F1-score (0.91 vs 0.92). This shows that performance is preserved despite the removal of low-importance features, and remains equally effective while being slightly more efficient.

### Objective 2: Predicting the measured range for LOS path

#### Performance Metrics

The various regression models were trained on the training set and evaluated on the test set using the following performance metrics:

1. Mean Absolute Error  
   Measures the average magnitude of prediction errors
2. Mean Squared Error  
   Penalises larger errors more heavily
3. Root Mean Squared Error  
   Square root of MSE, more interpretable in the same units as the target
4. R-squared  
   Proportion of variance in the target variable explained by the model

These metrics provide insight into how well the model is able to predict the actual range values from the input features. The findings are documented in the table below:

Line-of-Sight

| Regression Model | MAE | MSE | RMSE | R2 |
| --- | --- | --- | --- | --- |
| KNN | 0.87 | 1.26 | 1.12 | 0.54 |
| XGBoost | 0.74 | 0.92 | 0.96 | 0.66 |
| LightGBM | 0.74 | 0.92 | 0.96 | 0.66 |
| Random Forest | 0.76 | 0.95 | 0.97 | 0.65 |

#### Non-Line-of-Sight

| Regression Model | MAE | MSE | RMSE | R2 |
| --- | --- | --- | --- | --- |
| KNN | 1.59 | 4.19 | 2.05 | 0.29 |
| XGBoost | 1.22 | 2.54 | 1.59 | 0.57 |
| LightGBM | 1.22 | 2.53 | 1.59 | 0.57 |
| Random Forest | 1.20 | 2.51 | 1.59 | 0.58 |

#### Logical Analysis

Four regression models were evaluated for their ability to predict the UWB-based range under LOS conditions: K-Nearest Neighbors (KNN), XGBoost, LightGBM, and Random Forest. Among these, XGBoost and LightGBM achieved the best overall performance, both with a mean absolute error (MAE) of 0.74 meters, root mean squared error (RMSE) of 0.96 meters, and an R² score of 0.66. These metrics indicate that the models explain approximately 66% of the variance in the actual range values, which is reasonable given the inherent noise and complexity of indoor UWB signal behavior. Random Forest also performed comparably, with slightly higher MAE and RMSE but a slightly lower R² value.

In contrast, the KNN model produced the lowest performance, with a higher MAE of 0.87 meters, RMSE of 1.12 meters, and a reduced R² score of 0.54. This suggests that KNN was less capable of capturing the underlying structure in the data, likely due to its sensitivity to feature scale and its limitations in high-dimensional, noisy spaces. Overall, ensemble tree-based models demonstrated superior robustness and predictive accuracy, making them more suitable for UWB range estimation tasks in indoor environments.

## Overall Observations and Patterns

One of the most consistent findings is the strong predictive power of the RXPACC feature. It ranked the highest in feature importance during the classification phase and also showed a strong correlation with range in the regression task. In visualisation, RXPACC values had a tendency to increase with actual distance, suggesting that this feature reflects accumulated signal strength or power over distance, making it crucial for both LOS/NLOS classification and range estimation.

There were no significant signs of highly skewed or bimodal distributions in the selected features, likely due to the standardization and filtering steps performed during preprocessing. However, the distribution of RANGE values was concentrated in the lower range (mean ≈ 2.x meters), which led to the regression models performing better at short distances and struggling slightly at higher ranges.

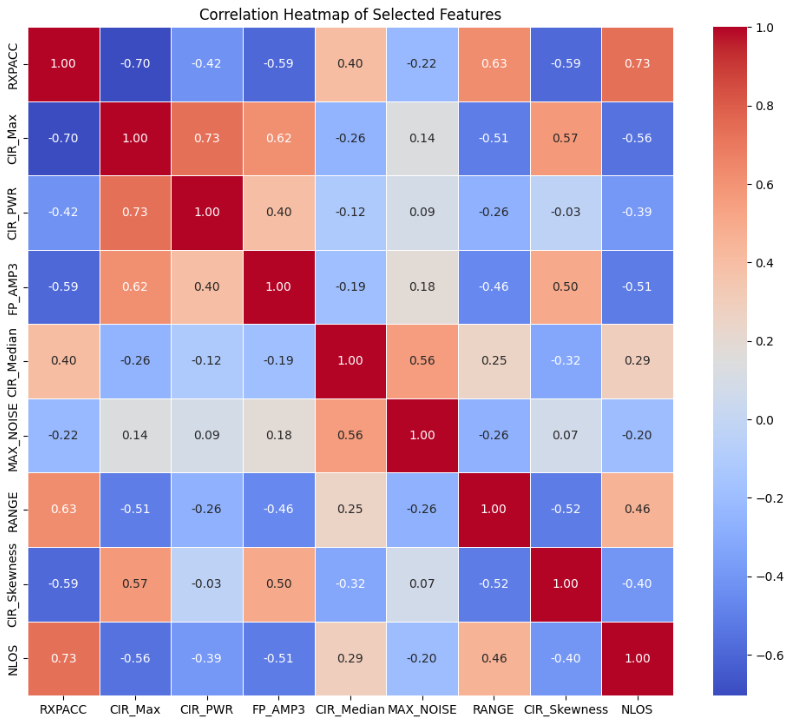
In the regression task, ensemble tree-based models (Random Forest, XGBoost, LightGBM) consistently outperformed KNN in terms of both accuracy and error metrics. These models handled the non-linear relationships and high-dimensional feature space more effectively. However, all models plateaued at an R² of around 0.66, suggesting that while the models captured the general trends in the data, some inherent noise (likely from multipath effects or hardware inaccuracies) limited further gains in predictive power.

These observations highlight the robustness of the feature engineering and modeling approach adopted in this project. While RXPACC emerged as a key indicator across both tasks, model performance was inherently bounded by the complexities of real-world UWB signal behavior. Nonetheless, the consistency of results across multiple models and evaluation metrics confirms that the system effectively captures the underlying structure of the data, particularly in more common short-range scenarios.

## Data Visualisation and Results Analysis

### Objective 1: Classification of LOS/NLOS

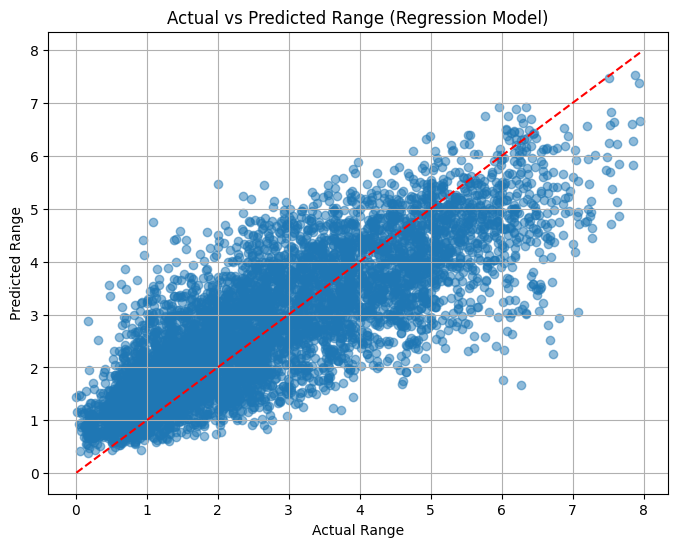
#### Correlation Heatmap



Some features exhibit moderate to strong correlations, however there are no features considered highly redundant ( > 0.80 or < -0.80 ), suggesting that the features are mostly independent and informative. This is because redundant features have already been successfully removed, by manually dropping them after doing a feature importance analysis earlier in the solution.

### Objective 2: Predicting the measured range for LOS path

#### Actual Vs Predicted Range



The scatter plot comparing actual and predicted range values demonstrates that the regression model performs reasonably well, with most points clustering around the ideal prediction line. Predictions are most accurate in the 0–4 meter range, where data density is highest. However, at medium to long ranges (above 6 meters), the spread increases and a noticeable underestimation trend emerges, particularly beyond 8 meters. This may reflect data imbalance or reduced model sensitivity to long-range CIR features. Despite a few outliers, the overall prediction pattern aligns with the reported R² of 0.66, indicating moderate predictive power with room for improvement at higher ranges.

#### Actual Vs Predicted Range (colour-coded by RXPACC)

RXPACC was added as a third visual dimension to evaluate whether signal quality affects prediction performance. The scatter plot shows a clear correlation between RXPACC and actual range, with higher RXPACC values typically associated with longer distances. The regression model performs better for lower RXPACC values, where points align closely along the ideal prediction line. However, at higher RXPACC values, the model tends to underestimate the range, suggesting that prediction confidence may vary with signal quality. These insights reinforce the potential value of RXPACC as an important feature and highlight its impact on model behavior across the range spectrum.

## Interesting Aspects of Solution (Assumptions/Optimisations)

### An Alternative Approach: Unsupervised Machine Learning (Clustering)

The solution implemented by the team explores the use of unsupervised machine learning for clustering. Specifically, the K-means algorithm was applied to the pre-processed dataset that was also used for supervised learning. The original dataset contained nine features, which made data visualization challenging. However, the team was still able to achieve an accuracy of 0.7555. Accuracy is done by comparing the predicted cluster against the dataset’s class.

The table below shows the result for the K-means clustering.

Classification Report:

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0.0 | 0.68 | 0.98 | 0.80 | 21000 |
| 1.0 | 0.96 | 0.53 | 0.69 | 21000 |
| **Accuracy** |  |  | **0.76** | **42000** |
| **Macro avg** | 0.82 | 0.76 | 0.74 | 42000 |
| **Weighted avg** | 0.82 | 0.76 | 0.74 | 42000 |

Accuracy: 0.7555

A similar experiment was conducted using the DBSCAN algorithm, but the results were not satisfactory. The poor performance of DBSCAN in this scenario can be attributed to its sensitivity to noise and varying densities, which led to suboptimal clustering. Moreover, DBSCAN's inability to handle clusters of different shapes and sizes further contributed to its unsuitability for this specific dataset.

In order to enhance visualization and gain a clearer understanding of the clustering, the team did a second round of pre-processing by selecting three significant features based on their correlation and importance, which had been evaluated earlier. These features RXPACC, RANGE, and AMP3 were chosen based off their relevance to the data. To ensure that these features were appropriate, the team ran the dataset through a Random Forest Classifier to check for accuracy.

| Confusion Matrix | **Predicted 0.0** | **Predicted 1.0** |
| --- | --- | --- |
| **Actual 0.0** | 3763 | 437 |
| **Actual 1.0** | 694 | 3506 |

The classifier confirmed that these three features were accurate enough, yielding an accuracy 0.8653571428571428 in the supervised machine learning model, which was deemed satisfactory.

Classification Report:

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0.0 | 0.84 | 0.90 | 0.87 | 4200 |
| 1.0 | 0.89 | 0.83 | 0.86 | 4200 |
| **Accuracy** |  |  | **0.87** | 8400 |
| **Macro avg** | 0.87 | 0.87 | 0.87 | 8400 |
| **Weighted avg** | 0.87 | 0.87 | 0.87 | 8400 |

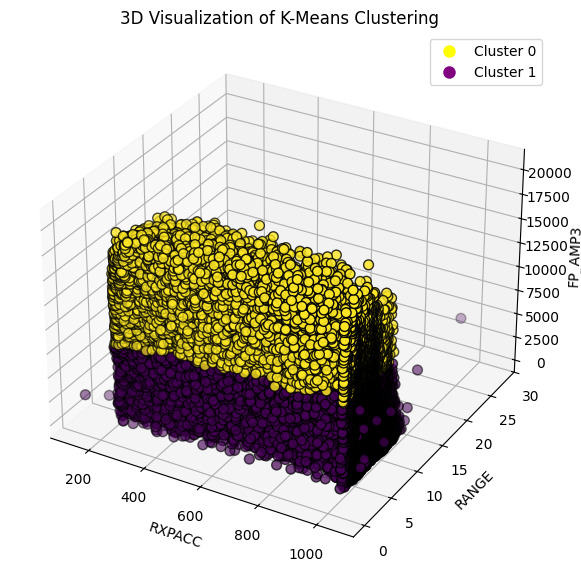
Accuracy: 0.8653571428571428

Following this validation, the dataset was then transferred into an unsupervised learning setup. The K-means algorithm, with K-means++ used to initialize the centroids, was applied to the reduced dataset. With K-means, the team achieved a clustering accuracy of 0.7435, which was considered satisfactory. The visualization of the clusters confirmed that K-means produced reasonable and well-defined clusters.

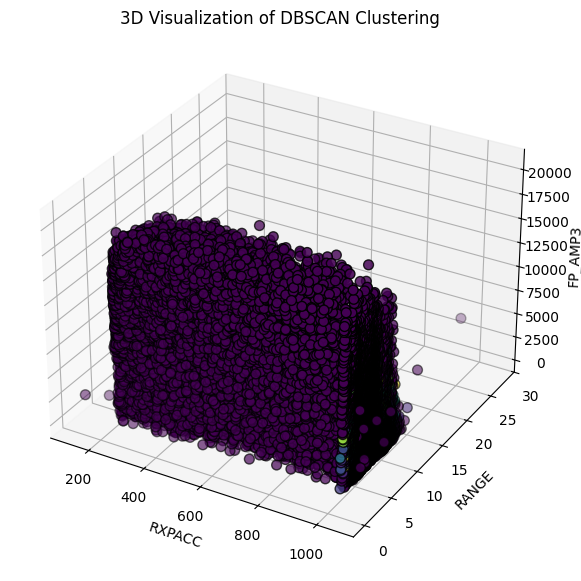
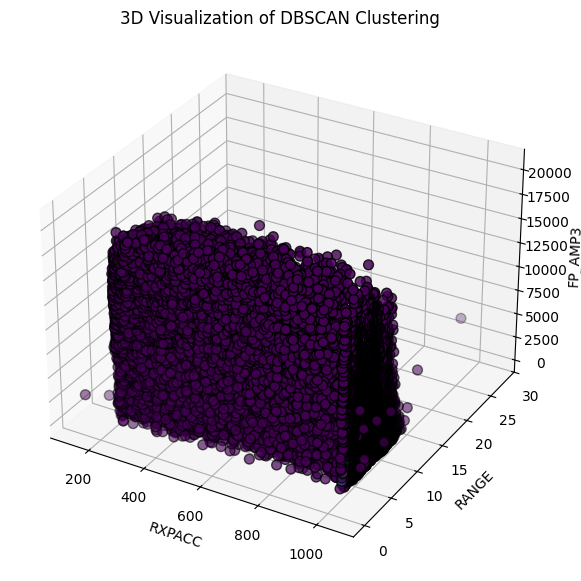
Classification Report:

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0.0 | 0.72 | 0.79 | 0.76 | 21000 |
| 1.0 | 0.77 | 0.69 | 0.73 | 21000 |
| **Accuracy** |  |  | **0.74** | **42000** |
| **Macro avg** | 0.75 | 0.74 | 0.74 | 42000 |
| **Weighted avg** | 0.75 | 0.74 | 0.74 | 42000 |

Accuracy: 0.7435



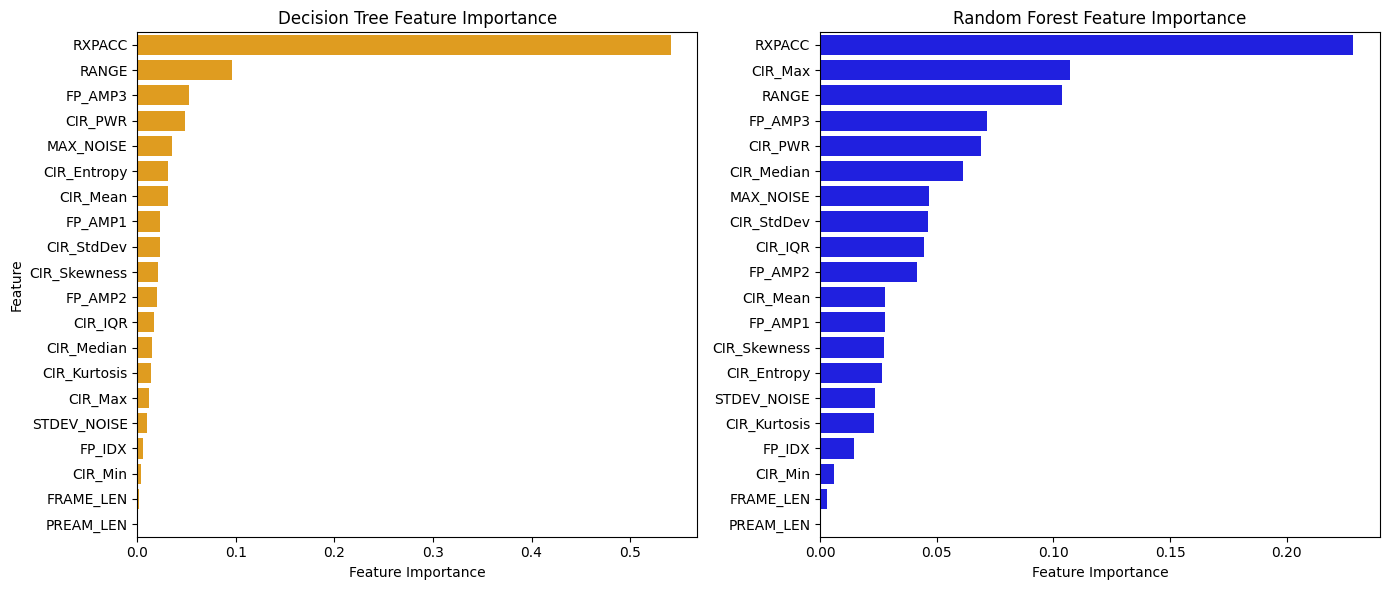
This 3D visualization showcases K-Means clustering using three features: RXPACC, RANGE, and FP\_AMP3. The plot highlights two distinct clusters, suggesting that the K-Means algorithm has effectively partitioned the data into two groups with clearly differing characteristics.

(eps=5, min\_samples= 2000) (eps=5, min\_samples= 10)

When testing DBSCAN with different values for the eps and min\_samples parameters the algorithm consistently failed to form meaningful clusters. This issue arises primarily due to DBSCAN's sensitivity to the density and distribution of the data.

In contrast, when the same approach was applied to DBSCAN, the clustering results were less effective. The visualization revealed that DBSCAN struggled to produce meaningful clusters. However, this is expected because DBSCAN relies on density-based separation, and in our case, the dataset has a uniform density distribution with 21,000 data points each for LOS and NLOS, making it difficult for DBSCAN to differentiate between them. Since both categories have a similar density, DBSCAN fails to identify distinct clusters and instead either merges them into a single cluster or labels a significant number of points as noise. This limitation highlights why DBSCAN is not suitable for our dataset, as it assumes clusters are separated by regions of lower density, which is not the case here.

### Feature Importance (Decision Tree Classifier)



To reaffirm the findings from the Random Forest Classifier, Decision Tree Classifier was also used to evaluate feature importance. The results revealed that four features—FP\_IDX, CIR\_MIN, FRAME\_LEN, and PREAM\_LEN—consistently ranked at the bottom in both classifiers. This indicates that these features contribute minimally to the model’s predictive power, allowing for removal without impacting performance.

For the top-ranked features, we observed that RXPACC consistently held the highest importance across both classifiers, making it the most influential feature in our dataset. Additionally, the top four features identified were RXPACC, RANGE, FP\_AMP3, and CIR\_PWR in the Decision Tree, while the Random Forest ranked RXPACC, CIR\_MAX, RANGE, and FP\_AMP3 as the most important. Despite minor variations in ranking, these results confirm that these four features are among the most critical for classification.

Furthermore, in the process of calculating correlation, FP\_AMP3 and FP\_AMP2 were found to have a high correlation (0.89), suggesting redundancy. To avoid multicollinearity and improve model efficiency, FP\_AMP2 was removed, retaining FP\_AMP3 as it ranked higher in feature importance. These insights set guidelines for the feature selection process, ensuring that only the most relevant features were retained for further analysis.

### Assumptions

The solution assumes that the statistical features derived from the numerous CIR samples are assumed to adequately represent the characteristics of the signal, and that the transformation of the features capture sufficient information for learning. Signal noise is also assumed to be relatively consistent across samples. No denoising methods were applied and constant or non-informative features like “BITRATE”, “PRFR” and “CH” were removed based on domain knowledge.

Although class balance was verified, the implementation included SMOTE as a precaution. It was assumed that both LOS and NLOS classes had similar levels of data quality and noise, allowing for fair training and evaluation.

### Optimisations

One optimisation made in the implementation was feature reduction using mutual information and correlation filtering. Highly correlated features and low variance features were removed and this was done to reduce overfitting and improve computational efficiency. Z-score normalisation was applied on all numerical features to scale them to ensure zero mean and unit variance, which was important for models sensitive to feature magnitudelike neural networks and KNN. Stratified train-test split in the 80/20 split was observed to preserve class proportions in the training and test datasets. This ensures fair model evaluation and prevents skewed predictions.

Through a combination of targeted assumptions and carefully designed optimisations, the implementation effectively addresses the challenges of LOS/NLOS classification and LOS range prediction in complex indoor UWB signal environments. The use of statistical feature engineering and balanced data preprocessing steps contributed to strong and reliable model performance. These choices, supported by logical analysis, demonstrate the robustness and generalisability of the overall data analytics pipeline.

## Conclusion

This solution demonstrates a comprehensive data analytics pipeline for classifying LOS and NLOS UWB signals and predicting LOS/NLOS path range in indoor environments. Through rigorous preprocessing, statistical feature engineering and machine learning models, the solution achieved high classification accuracy (92%) and competitive regression performance (R² ≈ 0.66). Feature selection techniques effectively reduced dimensionality without sacrificing predictive power, and effective visualisation brought about valuable insights into signal behaviour.

Key features like RXPACC consistently proved important across both tasks, reinforcing their value in UWB signal analysis. Models such as Random Forest, XGBoost, LightGBM, and KNN were individually evaluated and showed strong performance in handling the non-linear and noisy characteristics of indoor signals. The exploration of unsupervised clustering further highlighted the potential and limitations of alternative approaches.

Overall, this project demonstrates how combining domain expertise with data-driven techniques can improve indoor localisation. The implemented pipeline is effective, interpretable, and provides a strong foundation for future enhancements such as real-time deployment or hardware integration.

## Appendix

### A: Pseudocode

#### Import libraries

Import required libraries for:

- Data manipulation (Pandas, NumPy)

- Statistical computation (Scipy)

- Machine Learning (sklearn, imblearn)

#### Load and combine CSV files

Define function `import\_from\_files()`:

- Traverse all CSV files in the dataset directory

- Read each CSV into a DataFrame

- Convert to NumPy arrays and stack them vertically

- Return combined array as a single dataset

Call `import\_from\_files()` and convert array into a Pandas DataFrame

Dynamically define column names:

- First few are fixed features (e.g., NLOS, RANGE, FP\_AMPx, etc.)

- Remaining are CIR samples (named CIR0, CIR1, ..., CIRn)

Assign the column names to the DataFrame

#### Feature engineering on CIR columns

Define helper function `calculate\_entropy()`:

- Compute histogram of CIR signal

- Return entropy value

From the full dataset, create a copy for processing (`df\_selected`)

Identify all CIR columns (starting with "CIR", excluding "CIR\_PWR")

For each row, compute statistical features over the CIR columns:

- Mean

- Standard deviation

- Median

- Max

- Min

- Skewness

- Kurtosis

- Interquartile range (IQR)

Add these as new features to the DataFrame

Drop the original CIR columns (to reduce dimensionality)

#### Data cleaning

Drop:

- Duplicate rows

- Rows with missing values (NaNs)

Drop constant features identified through domain expertise:

- "BITRATE", "PRFR", "CH"

Convert all feature values to numeric type (if needed)

#### Feature selection

Compute Mutual Information (MI) scores between features and target ("NLOS")

Keep top 75% most informative features based on MI ranking

Compute correlation matrix

Drop one feature from any pair of features with correlation > 0.8

Remove features with very low variance (threshold: 1e-3)

Print names of removed features for logging/debugging

#### Feature scaling

Apply Z-score normalization to the features using StandardScaler

#### Class imbalance handling (SMOTE)

Check if class distribution (LOS vs NLOS) is imbalanced

If imbalanced:

Apply SMOTE to synthesize new samples for the minority class

#### Train-Test split (Classification task)

Split dataset into 80% training and 20% testing sets

Use stratified sampling to maintain class balance across sets

#### Train Random Forest Classifier

Initialize Random Forest classifier

Train on training set

Predict on test set

Evaluate using:

- Accuracy

- Confusion matrix

- Classification report (Precision, Recall, F1-score)

Compute and plot feature importances using a bar chart

- Create a scatter plot with:

-X-axis: Actual range values

-Y-axis: Predicted range values

-Add a red dashed line representing perfect predictions (y=x line)

-Optionally color points by feature values (like RXPACC)

-Add labels for both axes and a title

-Display grid lines

-Show the plot

Drop low-importance features and retrain model (repeat same steps)

#### Train Neural Network Classifier (MLP)

Initialize MLPClassifier with two hidden layers: (100, 50)

Train on training set

Predict on test set

Evaluate using:

- Accuracy

- Confusion matrix

- Classification report

#### Correlation Heatmap Visualisation

Compute correlation matrix of the scaled dataset

Use seaborn heatmap to visualize pairwise correlations

#### Train-Test split (Regression task)

For LOS samples:

- Drop "NLOS" and use "RANGE" as the regression target

- Keep previously selected features

Split data into 80% training, 20% testing

#### Train Random Forest Regressor

Initialize RandomForestRegressor

Train on training set

Predict on test set

Evaluate using:

- MAE (Mean Absolute Error)

- MSE (Mean Squared Error)

- RMSE (Root Mean Squared Error)

- R² (Coefficient of Determination)

#### Actual vs Predicted Visualisation

Plot scatter of predicted vs actual range values

Add red diagonal line to show ideal prediction line

#### Actual vs Predicted Visualisation (coloured RXPACC)

If X\_test is not already a DataFrame, convert it using training set column names.

Reset index for X\_test.

Reset index for y\_test.

Convert y\_pred into a Series using the same index as y\_test.

Trim X\_test rows if necessary.

This will be used to color the scatter plot.

X-axis: actual range values (y\_test)

Y-axis: predicted range values (y\_pred)

Color: RXPACC values using a color map (e.g., Viridis)

Add a color bar labeled RXPACC Value

Draw a diagonal reference line (ideal prediction line)

Add title, X and Y labels, and show the plot.

Output the shapes of X\_test, y\_test, and y\_pred.

Confirm that indices of X\_test and y\_test match.

### B: Source Code

import os

import pandas as pd

import numpy as np

from numpy import vstack

from scipy.stats import skew, kurtosis, entropy

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature\_selection import mutual\_info\_classif

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

def import\_from\_files():

"""

Read .csv files and store data into an array

format: |NLOS|data...|

"""

rootdir = '../dataset/'

output\_arr = []

first = 1

for dirpath, dirnames, filenames in os.walk(rootdir):

for file in filenames:

filename = os.path.join(dirpath, file)

print(filename) # Print the file being processed

df = pd.read\_csv(filename, sep=',', header=0) # Load the CSV file

input\_data = df.to\_numpy() # Convert to numpy array

if first > 0:

first = 0

output\_arr = input\_data # First data set

else:

output\_arr = vstack((output\_arr, input\_data)) # Stack subsequent data vertically

return output\_arr

# Import and process the dataset

data = import\_from\_files()

df\_data = pd.DataFrame(data)

# Validate the number of columns in the dataset

print(f"Number of columns in df\_data: {df\_data.shape[1]}")

# Dynamically define column names to match the actual data dimensions

base\_columns = ["NLOS", "RANGE", "FP\_IDX", "FP\_AMP1", "FP\_AMP2", "FP\_AMP3", "STDEV\_NOISE", "CIR\_PWR", "MAX\_NOISE", "RXPACC", "CH", "FRAME\_LEN", "PREAM\_LEN", "BITRATE", "PRFR"]

num\_cir\_columns = df\_data.shape[1] - len(base\_columns)

cir\_columns = [f"CIR{i}" for i in range(num\_cir\_columns)]

df\_data.columns = base\_columns + cir\_columns # Update column names dynamically

df\_selected = df\_data.copy() # Feature extraction and transformation

# Identify CIR columns (not including "CIR\_PWR" for now)

cir\_columns = [col for col in df\_selected.columns if col.startswith("CIR") and col != "CIR\_PWR"]

# Extract statistical features from CIR columns

df\_selected["CIR\_Mean"] = df\_selected[cir\_columns].mean(axis=1)

df\_selected["CIR\_StdDev"] = df\_selected[cir\_columns].std(axis=1)

df\_selected["CIR\_Median"] = df\_selected[cir\_columns].median(axis=1)

df\_selected["CIR\_Max"] = df\_selected[cir\_columns].max(axis=1)

df\_selected["CIR\_Min"] = df\_selected[cir\_columns].min(axis=1)

df\_selected["CIR\_Skewness"] = df\_selected[cir\_columns].apply(skew, axis=1)

df\_selected["CIR\_Kurtosis"] = df\_selected[cir\_columns].apply(kurtosis, axis=1)

df\_selected["CIR\_IQR"] = df\_selected[cir\_columns].apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25), axis=1)

# Drop the original CIR columns after feature extraction

df\_selected = df\_selected.drop(columns=cir\_columns)

• Removing Duplicates: We eliminate any duplicate rows to avoid redundancy.

• Handling Missing Values: We drop rows with missing values to maintain data integrity.

• Eliminating Constant Columns: Columns with only one value (like BITRATE, PRFR, and CH) don’t help in predictions, so we remove them.

• Encoding Categorical Features: We convert any categorical features into numeric values using Label Encoding so that the model can work with them.

# Remove duplicate rows to avoid redundancy

df\_selected.drop\_duplicates(inplace=True)

# 2. Handle missing values: drop rows with missing values (cannot impute median)

df\_selected.dropna(inplace=True)

# Check for and remove constant columns (columns with only one unique value, which don't add any predictive value)

constant\_columns = []

for col in ["BITRATE", "PRFR", "CH"]:

if df\_selected[col].nunique() == 1:

constant\_columns.append(col)

# Drop constant columns if any

if constant\_columns:

print("Removing constant columns:", constant\_columns)

df\_selected = df\_selected.drop(columns=constant\_columns)

# Ensure that all features are numeric by encoding categorical features

for col in df\_selected.columns:

if df\_selected[col].dtype == 'object': # Identify non-numeric features

encoder = LabelEncoder()

df\_selected[col] = encoder.fit\_transform(df\_selected[col]) # Encode categorical columns to numeric

#Feature Selection and Reduction

# Mutual Information (MI)

# Split the dataset into features (X) and target (y)

X = df\_selected.drop(columns=["NLOS"]) # Features (exclude NLOS column)

y = df\_selected["NLOS"] # Target variable

# Feature Selection: Compute Mutual Information (MI) Scores

mi\_scores = mutual\_info\_classif(X, y, discrete\_features=False, random\_state=42)

mi\_scores\_df = pd.DataFrame({"Feature": X.columns, "MI\_Score": mi\_scores})

# Sort by MI score to determine the most relevant features

mi\_scores\_df = mi\_scores\_df.sort\_values(by="MI\_Score", ascending=False)

# Select features with top 75% MI scores for better performance and less overfitting

threshold\_mi = mi\_scores\_df["MI\_Score"].quantile(0.25)

selected\_features = mi\_scores\_df[mi\_scores\_df["MI\_Score"] > threshold\_mi]["Feature"].tolist()

# Update the dataset to include only the selected features

df\_selected = df\_selected[["NLOS"] + selected\_features].copy()

# Correlation Analysis: Remove highly correlated features (correlation > 0.8) to reduce redundancy

corr\_matrix = df\_selected.corr().abs()

upper\_triangle = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

to\_drop = [column for column in upper\_triangle.columns if any(upper\_triangle[column] > 0.8)]

df\_selected = df\_selected.drop(columns=to\_drop).copy()

# Remove features with low variance (near-constant features)

variance\_threshold = 1e-3

low\_variance\_features = [col for col in df\_selected.columns if df\_selected[col].var() < variance\_threshold]

df\_selected = df\_selected.drop(columns=low\_variance\_features).copy()

# Print the features that were removed at each step

print("Features removed due to low Mutual Information:", set(X.columns) - set(selected\_features))

print("Features removed due to high correlation:", to\_drop)

print("Features removed due to low variance:", low\_variance\_features)

print("Columns after removal:", df\_selected.columns.tolist())

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df\_selected.drop(columns=["NLOS"])),

columns=df\_selected.columns[1:])

df\_scaled["NLOS"] = df\_selected["NLOS"].values # Add the target variable back

# Handle class imbalance (if the class distribution is skewed) using SMOTE

class\_counts = df\_scaled["NLOS"].value\_counts(normalize=True)

print("Class Distribution Before SMOTE:")

print(class\_counts)

# Apply SMOTE if the minority class is less than 20% of the dataset

if class\_counts.min() < 0.2:

X\_resampled, y\_resampled = SMOTE().fit\_resample(df\_scaled.drop(columns=["NLOS"]), df\_scaled["NLOS"])

df\_scaled = pd.DataFrame(X\_resampled, columns=df\_scaled.columns[:-1])

df\_scaled["NLOS"] = y\_resampled

print("Class Distribution After SMOTE:")

print(df\_scaled["NLOS"].value\_counts(normalize=True))

#Data Mining - Model Training and Evaluation

#Splitting the Dataset

# Split data into training and test sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_scaled.drop(columns=["NLOS"]), df\_scaled["NLOS"],

test\_size=0.2, stratify=df\_scaled["NLOS"], random\_state=42)

print(f"Train Set: {X\_train.shape}, Test Set: {X\_test.shape}")

print("Columns after removal:", df\_selected.columns.tolist())

df\_scaled.hist(bins=30, figsize=(15, 12))

plt.suptitle("Histogram of Normalized Features", fontsize=16)

plt.show()

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import matplotlib.pyplot as plt

import seaborn as sns

# Train the Random Forest model

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

# Feature importance visualization for Random Forest model

feature\_importances = pd.DataFrame({"Feature": X\_train.columns, "Importance": model.feature\_importances\_})

feature\_importances = feature\_importances.sort\_values(by="Importance", ascending=False)

plt.figure(figsize=(10, 6))

sns.barplot(x=feature\_importances['Importance'], y=feature\_importances['Feature'])

plt.xlabel("Feature Importance Score")

plt.ylabel("Features")

plt.title("Feature Importance in Random Forest Classifier")

plt.show()

# Performance evaluation using accuracy, confusion matrix, and classification report

# After visual inspection of the feature importance diagram, we noticed that some features

# like "PREAM\_LEN", "FP\_IDX", and "FP\_AMP1" may not provide significant predictive power.

# These features might be constant or highly redundant, which could negatively affect the model's performance.

# To address this, we decided to remove these constant features before retraining the model.

constant\_columns2 = []

for col in [ "FP\_IDX", "FP\_AMP1"]:

constant\_columns2.append(col)

# Drop constant columns

if constant\_columns2:

print("Removing constant columns:", constant\_columns2)

df\_scaled = df\_scaled.drop(columns=constant\_columns2)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_scaled.drop(columns=["NLOS"]), df\_scaled["NLOS"],

test\_size=0.2, stratify=df\_scaled["NLOS"], random\_state=42)

print(f"Train Set: {X\_train.shape}, Test Set: {X\_test.shape}")

print("Columns after removal:", df\_scaled.columns.tolist())

Train Set: (33600, 8), Test Set: (8400, 8)

Columns after removal: ['RXPACC', 'CIR\_Max', 'CIR\_PWR', 'FP\_AMP3', 'CIR\_Median', 'MAX\_NOISE', 'RANGE', 'CIR\_Skewness', 'NLOS']

# Train the Random Forest model

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

feature\_importances = pd.DataFrame({"Feature": X\_train.columns, "Importance": model.feature\_importances\_})

feature\_importances = feature\_importances.sort\_values(by="Importance", ascending=False)

plt.figure(figsize=(10, 6))

sns.barplot(x=feature\_importances['Importance'], y=feature\_importances['Feature'])

plt.xlabel("Feature Importance Score")

plt.ylabel("Features")

plt.title("Feature Importance in Random Forest Classifier")

plt.show()

# Performance metrics

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print(f"Confusion Matrix:\n{conf\_matrix}")

print(f"Classification Report:\n{class\_report}")

# Confusion matrix plot

plt.figure(figsize=(6, 5))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=['Predicted LOS', 'Predicted NLOS'],

yticklabels=['Actual LOS', 'Actual NLOS'])

plt.title("Confusion Matrix - Random Forest Classifier")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

from sklearn.neural\_network import MLPClassifier

# Train Neural Network model

nn = MLPClassifier(hidden\_layer\_sizes=(100, 50), max\_iter=1000, random\_state=42)

nn.fit(X\_train, y\_train)

# Evaluate the model

y\_pred\_nn = nn.predict(X\_test)

accuracy\_nn = accuracy\_score(y\_test, y\_pred\_nn)

conf\_matrix\_nn = confusion\_matrix(y\_test, y\_pred\_nn)

class\_report\_nn = classification\_report(y\_test, y\_pred\_nn)

print(f"Accuracy: {accuracy\_nn}")

print(f"Confusion Matrix:\n{conf\_matrix\_nn}")

print(f"Classification Report:\n{class\_report\_nn}")

import seaborn as sns

import matplotlib.pyplot as plt

sns.heatmap(conf\_matrix\_nn, annot=True, fmt='d', cmap='Blues',

xticklabels=['Predicted LOS', 'Predicted NLOS'],

yticklabels=['Actual LOS', 'Actual NLOS'])

plt.title("Confusion Matrix - Neural Network")

plt.xlabel("Prediction")

plt.ylabel("Actual")

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Compute correlation matrix for the selected and processed data

corr = df\_scaled.corr()

# Plot the heatmap

plt.figure(figsize=(12, 10))

sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title("Correlation Heatmap of Selected Features")

plt.show()

**#Objective 2:**

# Import required libraries for regression

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# 1. Split both df\_scaled and df\_selected into LOS and NLOS

df\_scaled\_LOS = df\_scaled[df\_scaled["NLOS"] == 0].copy()

df\_scaled\_NLOS = df\_scaled[df\_scaled["NLOS"] == 1].copy()

df\_selected\_LOS = df\_selected[df\_selected["NLOS"] == 0].copy()

df\_selected\_NLOS = df\_selected[df\_selected["NLOS"] == 1].copy()

# 2. Define features (X) from scaled dataframe and target (y) from original

X\_LOS = df\_scaled\_LOS.drop(columns=["NLOS", "RANGE" ]) # scaled features

y\_LOS = df\_selected\_LOS["RANGE"] # unscaled target.

#'RXPACC', 'CIR\_Max', 'CIR\_PWR', 'FP\_AMP3', 'CIR\_Median', 'MAX\_NOISE', 'RANGE', 'CIR\_Skewness', 'NLOS'

X\_NLOS = df\_scaled\_NLOS.drop(columns=["NLOS", "RANGE" ])

y\_NLOS = df\_selected\_NLOS["RANGE"]

# 3. Optional: check mean RANGE for LOS samples

average\_value = y\_LOS.median()

print("Average RANGE for LOS samples:", average\_value)

#Data Mining

#Split the data into training and testing sets

X\_train\_reg\_LOS, X\_test\_reg\_LOS, y\_train\_reg\_LOS, y\_test\_reg\_LOS = train\_test\_split(X\_LOS, y\_LOS, test\_size=0.2, random\_state=42)

regressor\_LOS = RandomForestRegressor(random\_state=42)

regressor\_LOS.fit(X\_train\_reg\_LOS, y\_train\_reg\_LOS)

RandomForestRegressor(random\_state=42)

Evaluate the Regression Model

Random Forest Regression

# : Evaluate the Regression Model

# LOS Predict on the test set

y\_pred\_reg\_LOS = regressor\_LOS.predict(X\_test\_reg\_LOS)

# Evaluate the regression model

mae = mean\_absolute\_error(y\_test\_reg\_LOS, y\_pred\_reg\_LOS)

mse = mean\_squared\_error(y\_test\_reg\_LOS, y\_pred\_reg\_LOS)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test\_reg\_LOS, y\_pred\_reg\_LOS)

print(f"Random Forest Regressor: MAE: {mae:.2f}")

print(f"Random Forest Regressor: MSE: {mse:.2f}")

print(f"Random Forest Regressor: RMSE: {rmse:.2f}")

print(f"Random Forest Regressor: R²: {r2:.2f}")

Random Forest Regressor: MAE: 0.76

Random Forest Regressor: MSE: 0.95

Random Forest Regressor: RMSE: 0.97

Random Forest Regressor: R²: 0.65

#XGBoost

# Import required libraries

from xgboost import XGBRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

# Initialize and train XGBoost Regressor

xgb\_regressor = XGBRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=6, random\_state=42)

xgb\_regressor.fit(X\_train\_reg\_LOS, y\_train\_reg\_LOS)

# : Predict and evaluate

y\_pred\_xgb = xgb\_regressor.predict(X\_test\_reg\_LOS)

mae\_xgb = mean\_absolute\_error(y\_test\_reg\_LOS, y\_pred\_xgb)

mse\_xgb = mean\_squared\_error(y\_test\_reg\_LOS, y\_pred\_xgb)

rmse\_xgb = np.sqrt(mse\_xgb)

r2\_xgb = r2\_score(y\_test\_reg\_LOS, y\_pred\_xgb)

# Output results

print(f"XGBoost Regression - MAE: {mae\_xgb:.2f}")

print(f"XGBoost Regression - MSE: {mse\_xgb:.2f}")

print(f"XGBoost Regression - RMSE: {rmse\_xgb:.2f}")

print(f"XGBoost Regression - R²: {r2\_xgb:.2f}")

#LightGBM

# Import required libraries

from lightgbm import LGBMRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

# Initialize and train LightGBM Regressor

lgbm\_regressor = LGBMRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=6, random\_state=42)

lgbm\_regressor.fit(X\_train\_reg\_LOS, y\_train\_reg\_LOS)

# Predict and evaluate

y\_pred\_lgbm = lgbm\_regressor.predict(X\_test\_reg\_LOS)

mae\_lgbm = mean\_absolute\_error(y\_test\_reg\_LOS, y\_pred\_lgbm)

mse\_lgbm = mean\_squared\_error(y\_test\_reg\_LOS, y\_pred\_lgbm)

rmse\_lgbm = np.sqrt(mse\_lgbm)

r2\_lgbm = r2\_score(y\_test\_reg\_LOS, y\_pred\_lgbm)

# Output results

print(f"LightGBM Regression - MAE: {mae\_lgbm:.2f}")

print(f"LightGBM Regression - MSE: {mse\_lgbm:.2f}")

print(f"LightGBM Regression - RMSE: {rmse\_lgbm:.2f}")

print(f"LightGBM Regression - R²: {r2\_lgbm:.2f}")

# K-Nearest Neighbors (KNN)

# Import required libraries

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

# Initialize and train KNN Regressor

knn\_regressor = KNeighborsRegressor(n\_neighbors=5)

knn\_regressor.fit(X\_train\_reg\_LOS, y\_train\_reg\_LOS)

# Predict and evaluate

y\_pred\_knn = knn\_regressor.predict(X\_test\_reg\_LOS)

mae\_knn = mean\_absolute\_error(y\_test\_reg\_LOS, y\_pred\_knn)

mse\_knn = mean\_squared\_error(y\_test\_reg\_LOS, y\_pred\_knn)

rmse\_knn = np.sqrt(mse\_knn)

r2\_knn = r2\_score(y\_test\_reg\_LOS, y\_pred\_knn)

# Output results

print(f"KNN Regression - MAE: {mae\_knn:.2f}")

print(f"KNN Regression - MSE: {mse\_knn:.2f}")

print(f"KNN Regression - RMSE: {rmse\_knn:.2f}")

print(f"KNN Regression - R²: {r2\_knn:.2f}")

#NLOS outputs

# Split the data into training and testing sets

X\_train\_reg\_NLOS, X\_test\_reg\_NLOS, y\_train\_reg\_NLOS, y\_test\_reg\_NLOS = train\_test\_split(X\_NLOS, y\_NLOS, test\_size=0.3, random\_state=42)

# Train a regression model (e.g., Random Forest Regressor)

regressor\_NLOS = RandomForestRegressor(random\_state=42)

regressor\_NLOS.fit(X\_train\_reg\_NLOS, y\_train\_reg\_NLOS)

# NLOS Predict on the test set

y\_pred\_reg\_NLOS = regressor\_NLOS.predict(X\_test\_reg\_NLOS)

# Evaluate the regression model

mae\_reg\_NLOS = mean\_absolute\_error(y\_test\_reg\_NLOS, y\_pred\_reg\_NLOS)

mse\_reg\_NLOS = mean\_squared\_error(y\_test\_reg\_NLOS, y\_pred\_reg\_NLOS)

rmse\_reg\_NLOS = np.sqrt(mse\_reg\_NLOS)

r2\_reg\_NLOS = r2\_score(y\_test\_reg\_NLOS, y\_pred\_reg\_NLOS)

print(f"Random Forest Regressor: MAE : {mae\_reg\_NLOS:.2f}")

print(f"Random Forest Regressor: MSE: {mse\_reg\_NLOS:.2f}")

print(f"Random Forest Regressor: RMSE: {rmse\_reg\_NLOS:.2f}")

print(f"Random Forest Regressor: R²: {r2\_reg\_NLOS:.2f}")

# XGBoost

# Import required libraries

from xgboost import XGBRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

# Initialize and train XGBoost Regressor

xgb\_regressor\_NLOS = XGBRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=6, random\_state=42)

xgb\_regressor\_NLOS.fit(X\_train\_reg\_NLOS, y\_train\_reg\_NLOS)

# Predict and evaluate

y\_pred\_xgb\_NLOS = xgb\_regressor\_NLOS.predict(X\_test\_reg\_NLOS)

mae\_xgb\_NLOS = mean\_absolute\_error(y\_test\_reg\_NLOS, y\_pred\_xgb\_NLOS)

mse\_xgb\_NLOS = mean\_squared\_error(y\_test\_reg\_NLOS, y\_pred\_xgb\_NLOS)

rmse\_xgb\_NLOS = np.sqrt(mse\_xgb\_NLOS)

r2\_xgb\_NLOS = r2\_score(y\_test\_reg\_NLOS, y\_pred\_xgb\_NLOS)

# Import required libraries

from lightgbm import LGBMRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

# Initialize and train LightGBM Regressor

lgbm\_regressor\_NLOS = LGBMRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=6, random\_state=42)

lgbm\_regressor\_NLOS.fit(X\_train\_reg\_NLOS, y\_train\_reg\_NLOS)

# Predict and evaluate

y\_pred\_lgbm\_NLOS = lgbm\_regressor\_NLOS.predict(X\_test\_reg\_NLOS)

mae\_lgbm\_NLOS = mean\_absolute\_error(y\_test\_reg\_NLOS, y\_pred\_lgbm\_NLOS)

mse\_lgbm\_NLOS = mean\_squared\_error(y\_test\_reg\_NLOS, y\_pred\_lgbm\_NLOS)

rmse\_lgbm\_NLOS = np.sqrt(mse\_lgbm\_NLOS)

r2\_lgbm\_NLOS = r2\_score(y\_test\_reg\_NLOS, y\_pred\_lgbm\_NLOS)

# Output results

print(f"LightGBM Regression - MAE: {mae\_lgbm\_NLOS:.2f}")

print(f"LightGBM Regression - MSE: {mse\_lgbm\_NLOS:.2f}")

print(f"LightGBM Regression - RMSE: {rmse\_lgbm\_NLOS:.2f}")

print(f"LightGBM Regression - R²: {r2\_lgbm\_NLOS:.2f}")

#KNN

# Import required libraries

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

# Initialize and train KNN Regressor

knn\_regressor\_NLOS = KNeighborsRegressor(n\_neighbors=5)

knn\_regressor\_NLOS.fit(X\_train\_reg\_NLOS, y\_train\_reg\_NLOS)

# Predict and evaluate

y\_pred\_knn\_NLOS = knn\_regressor\_NLOS.predict(X\_test\_reg\_NLOS)

mae\_knn\_NLOS = mean\_absolute\_error(y\_test\_reg\_NLOS, y\_pred\_knn\_NLOS)

mse\_knn\_NLOS = mean\_squared\_error(y\_test\_reg\_NLOS, y\_pred\_knn\_NLOS)

rmse\_knn\_NLOS = np.sqrt(mse\_knn\_NLOS)

r2\_knn\_NLOS = r2\_score(y\_test\_reg\_NLOS, y\_pred\_knn\_NLOS)

# Output results

print(f"KNN Regression - MAE: {mae\_knn\_NLOS:.2f}")

print(f"KNN Regression - MSE: {mse\_knn\_NLOS:.2f}")

print(f"KNN Regression - RMSE: {rmse\_knn\_NLOS:.2f}")

print(f"KNN Regression - R²: {r2\_knn\_NLOS:.2f}")

# Plotting Actual vs Predicted values

plt.figure(figsize=(8, 6))

plt.scatter(y\_test\_reg\_LOS, y\_pred\_reg\_LOS, alpha=0.5)

plt.plot([y\_test\_reg\_LOS.min(), y\_test\_reg\_LOS.max()], [y\_test\_reg\_LOS.min(), y\_test\_reg\_LOS.max()], 'r--')

plt.xlabel("Actual Range")

plt.ylabel("Predicted Range")

plt.title("Actual vs Predicted Range (Regression Model)")

plt.grid(True)

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

# Ensure X\_test is a DataFrame with correct column names

if not isinstance(X\_test\_reg\_LOS, pd.DataFrame):

X\_test\_reg\_LOS = pd.DataFrame(X\_test\_reg\_LOS, columns=X\_train\_reg\_LOS.columns)

# Reset indices to avoid mismatches

X\_test\_reg\_LOS = X\_test\_reg\_LOS.reset\_index(drop=True)

y\_test\_reg = y\_test\_reg\_LOS.reset\_index(drop=True)

# Ensure y\_pred\_reg is a Pandas Series with matching index

y\_pred\_reg\_LOS = pd.Series(y\_pred\_reg\_LOS[:len(y\_test\_reg\_LOS)], index=y\_test\_reg\_LOS.index)

# Subset X\_test to match y\_test\_reg's length

X\_test\_reg\_LOS = X\_test\_reg\_LOS.iloc[:len(y\_test\_reg\_LOS)]

# Extract RXPACC values after index alignment

rxpacc\_test = X\_test\_reg\_LOS['RXPACC']

# Plot: Predicted vs Actual RANGE, colored by RXPACC

plt.figure(figsize=(8, 6))

scatter = plt.scatter(y\_test\_reg, y\_pred\_reg\_LOS, c=rxpacc\_test, cmap='viridis', alpha=0.7)

plt.colorbar(scatter, label='RXPACC Value')

plt.plot([y\_test\_reg.min(), y\_test\_reg.max()], [y\_test\_reg.min(), y\_test\_reg.max()], color='red', linestyle='--')

plt.title("Predicted vs Actual RANGE (Colored by RXPACC)")

plt.xlabel("Actual RANGE")

plt.ylabel("Predicted RANGE")

plt.show()

# Debugging output to confirm correct index alignment

print("Shapes:", X\_test\_reg\_LOS.shape, y\_test\_reg\_LOS.shape, y\_pred\_reg\_LOS.shape)

print("Indices match:", X\_test\_reg\_LOS.index.equals(y\_test\_reg\_LOS.index))