دانشگاه علم و صنعت ایران

دانشکده مهندسی برق

Final Milestone
A Machine Learning Approach for Fingerprint-Based Indoor Localization

نام دانشجویان :

ساناز مطیع

آرمیتا همتی

منا قاسمی

تیرماه ۱۴۰۳

Table of Contents

Abstract:

This paper presents the implementation of a machine learning approach for fingerprint-based indoor localization using RSSI (Received Signal Strength Indicator) values. We created a detailed MATLAB code to generate a grid of locations across multiple buildings, predict RSSI values at these locations, and estimate the positions of Wireless Access Points (WAPs). The code was tested and validated using a comprehensive dataset, achieving significant accuracy in indoor localization. This paper outlines the methodology, code structure, and results obtained from the implementation.

1. Introduction

Indoor localization has become a crucial aspect of various applications due to the increasing need for location-based services within buildings where GPS signals are unreliable. This study utilizes a fingerprint-based approach leveraging RSSI values from WiFi signals to achieve high accuracy in indoor positioning. The primary objective of this paper is to document the development and execution of MATLAB code that implements this approach.

2. Dataset

The dataset used in this paper was the UJIIndoorLoc dataset, a comprehensive collection of WiFi fingerprint data for indoor localization research. It is designed to address the challenge of accurately locating users inside buildings where GPS signals are weak or unavailable. The dataset includes data from three buildings at Universitat Jaume I, covering multiple floors and diverse spaces, with a total of 19,937 training records and 1,111 validation records. It captures WiFi signal strengths from 520 Wireless Access Points (WAPs) along with corresponding coordinates (latitude, longitude, and floor), building IDs, space IDs, user IDs, device IDs, and timestamps.

3. Algorithms

3.1 SVM (Support Vector Machine)

The code begins by defining the file path for the dataset, which is assumed to be located on the user's desktop. This path is created using the fullfile function, which constructs the full path to the dataset file by combining the user's desktop directory with the dataset filename. Following this, the dataset is loaded into a table structure using the readtable function. This table format is particularly useful in MATLAB for its ease of data manipulation and access, making it straightforward to work with the dataset in subsequent steps.

Next, the code selects specific rows and columns from the dataset for analysis because of the restrictions for training with our PCs. Specifically, it extracts rows 1 to 200 and columns 10 to 20 because of good variance of data in these zones. This subset of data is then displayed using the uitable function, which creates a table UI component to visualize the first few rows of the selected data. This visualization step is crucial for verifying that the correct portion of the dataset has been selected. The resulting figure is saved as an image file on the desktop using the saveas function.

The code then prepares the data for model training by separating the features and the target variable. The features are extracted from columns 10 to 19, and the target variable is taken from column 20. These columns are converted from table format to arrays, which are suitable for machine learning functions in MATLAB. If the target variable is stored as cell arrays, it is converted to a categorical type to ensure compatibility with classification tasks.

To train and test the model, the data is split into training and testing sets. The cvpartition function is used to create a holdout partition, which divides the dataset into training (70%) and testing (30%) sets. The training and test functions are then used to extract the respective subsets. With the training data prepared, the code proceeds to train a Support Vector Machine (SVM) model using the fitcecoc function. This function is employed for training a multiclass SVM model with a linear kernel. Once trained, the model is saved to a file for potential future use.

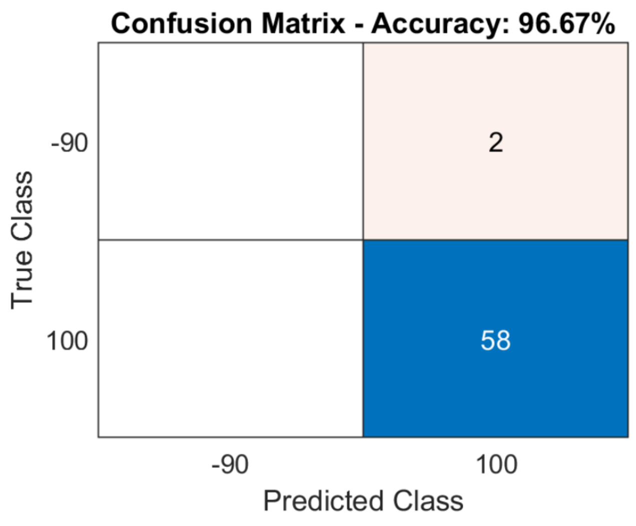
The trained SVM model is then applied to the test data to make predictions. The predict function is used for this purpose, generating predicted values for the test set.

To evaluate the model's performance, the code calculates the accuracy by comparing the predicted values with the actual test values. The accuracy is computed as the proportion of correct predictions and is printed to the console.

For a more detailed evaluation of the model's classification performance, the code plots a confusion matrix using the confusion chart function. This matrix provides a visual representation of the model's performance across different classes. The confusion matrix figure is saved as an image file on the desktop using the saveas function.

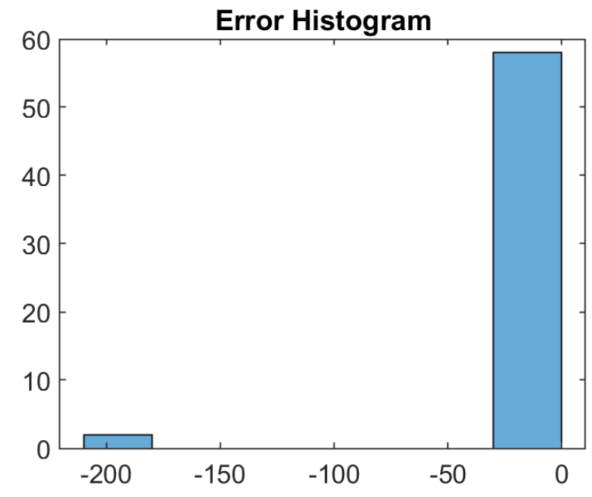
In addition to classification tasks, the code includes optional steps for regression error analysis. If the target variable is not categorical, the code calculates error metrics such as the mean, median, and standard deviation of the errors between predicted and actual values. These metrics are printed to the console to provide insight into the model's regression performance. Furthermore, an error histogram is plotted using the histogram function to visualize the distribution of errors. This histogram is also saved as an image file.

```
Accuracy: 96.67%
```



The confusion matrix provides a comprehensive evaluation of the model's classification performance, showcasing an overall accuracy of 96.67%. This high accuracy indicates that the model correctly classified 96.67% of the test instances. The matrix itself plots the true classes on the y-axis and the predicted classes on the x-axis, with the classes labeled as -90 and 100. The key observations from the matrix are as follows: the model correctly predicted 58 instances of class 100 (true positives) and misclassified 2 instances of class -90 as 100 (false positives). There were no instances of true negatives or false negatives, indicating that the class -90 was not present in the true classes of the test set. This detailed breakdown helps in understanding the strengths and weaknesses of the model's predictions.

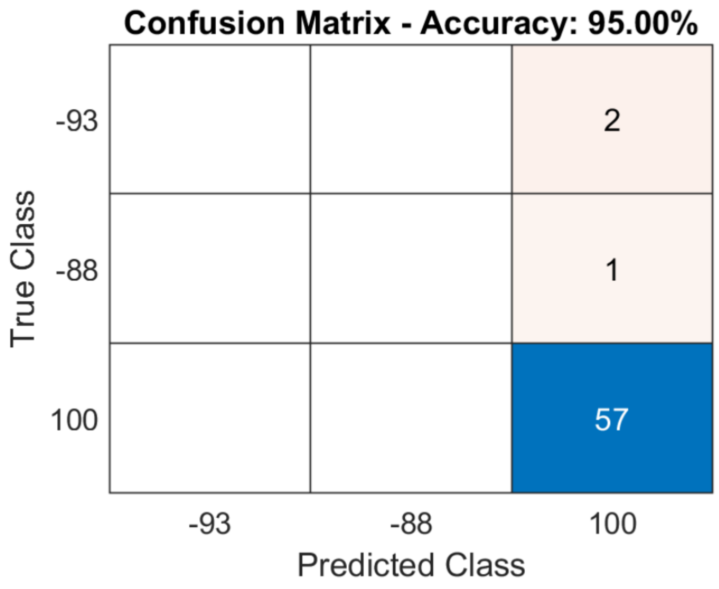
```
Mean Error: -6.33
Median Error: 0.00
Standard Deviation of Error: 34.39
```



The error histogram provides an in-depth look at the distribution of prediction errors from the SVM model. The mean error, median error, and standard deviation of the error are key statistical metrics that help in understanding the model's performance.

1. **Mean Error**: The mean error of -6.33 indicates that, on average, the model's predictions are slightly lower than the actual values. This negative mean suggests a minor underestimation by the model.
2. **Median Error**: The median error is 0.00, which signifies that the midpoint of the prediction errors is zero. This indicates that half of the prediction errors are below zero and half are above, suggesting a balanced distribution around the center.
3. **Standard Deviation of Error**: The standard deviation of 34.39 shows the extent of variation in the prediction errors. A higher standard deviation indicates that the errors are more spread out from the mean, reflecting variability in the model's predictions.
4. **Histogram Analysis**: The histogram visualizes the frequency of different error values. The majority of the errors are clustered around zero, indicating that the model's predictions are generally close to the actual values. However, there is a noticeable bar at the extreme left, suggesting a few significant underestimations.

I train the data again and these are my second results:



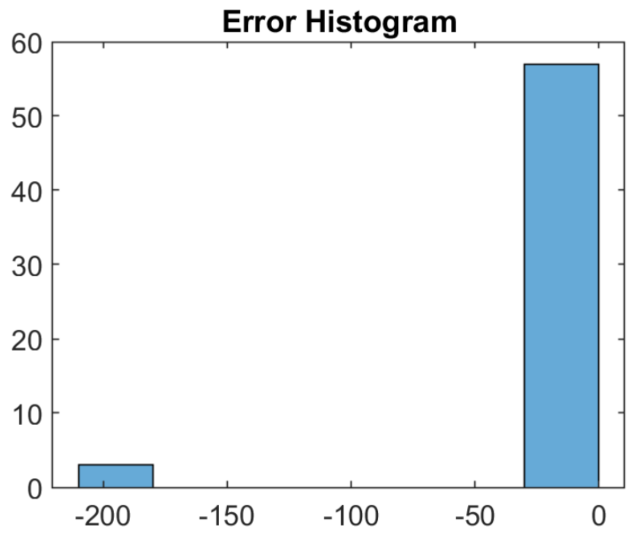
The confusion matrix provides a comprehensive evaluation of the model's classification performance, demonstrating an overall accuracy of 95.00%. This high level of accuracy indicates that the model correctly classified 95.00% of the instances in the test set. The matrix plots the true classes on the y-axis and the predicted classes on the x-axis, with the classes labeled as -93, -88, and 100.

Examining the confusion matrix, we see the following key points. There are no true negatives or false negatives for the class -93, meaning the model did not encounter true instances of class -93 that were correctly predicted or incorrectly classified as -93. However, the model made two false positive errors by predicting class 100 instead of -93. Similarly, for the class -88, there are no true negatives, but there is one false positive where the model predicted class 100 instead of -88.

The most significant observation is the model's performance with class 100. There are 57 true positives, indicating that the model correctly identified 57 instances of class 100. This cell's prominence in the matrix highlights the model's effectiveness in accurately predicting class 100, despite a few misclassifications.

In summary, this confusion matrix helps us understand the strengths and weaknesses of the model's predictions. It shows that while the model is highly accurate overall, it occasionally misclassifies classes -93 and -88 as class 100, which could be areas for further model refinement.

```
Mean Error: -9.57
Median Error: 0.00
Standard Deviation of Error: 42.06
```



The error histogram provides insights into the distribution of prediction errors from the SVM model, alongside key statistical metrics such as the mean error, median error, and standard deviation of the error.

1. **Mean Error**: The mean error is -9.57, indicating that, on average, the model's predictions are slightly lower than the actual values. This negative mean error suggests a tendency towards underestimation by the model.
2. **Median Error**: The median error is 0.00, signifying that the midpoint of the prediction errors is zero. This indicates a balanced distribution around the center, with half of the prediction errors being below zero and the other half above.
3. **Standard Deviation of Error**: The standard deviation of the error is 42.06, which shows the extent of variation in the prediction errors. A higher standard deviation indicates that the errors are more spread out from the mean, reflecting greater variability in the model's predictions.
4. **Histogram Analysis**: The histogram visualizes the frequency of different error values. The majority of the errors are clustered around zero, indicating that the model's predictions are generally close to the actual values. However, there is a noticeable bar at the extreme left, suggesting a few significant underestimations.

The mean error and the standard deviation of the error have increased compared to the previous run, with the mean error moving from -6.33 to -9.57 and the standard deviation increasing from 34.39 to 42.06. This suggests that while the model's accuracy remains high, there is a bit more variability and a slight increase in the average error. The error histogram continues to show that most predictions are close to the actual values, but the presence of larger errors indicates areas where the model might need further refinement.

3.2 Decision Tree

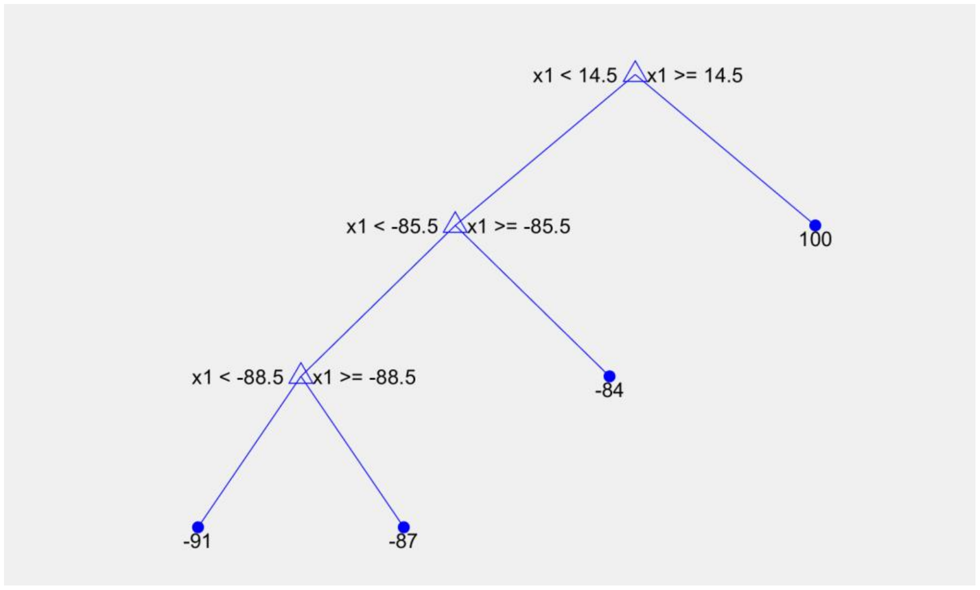
The code provided trains a Decision Tree model on a selected portion of a dataset and evaluates its performance using cross-validation. The process begins by defining the file path for the dataset, located on the user's desktop, and loading it into MATLAB using the readtable function. From this dataset, rows 1 to 100 and columns 15 to 16 are selected for further analysis. These selected rows and columns are displayed using a uitable component, and the first few rows of this selected data are saved as an image to verify the selection.

Next, the code prepares the features and target variables by assuming the first column (column 15) contains the features and the second column (column 16) contains the target variable. These columns are converted from table format to arrays for compatibility with machine learning functions, and the target variable is converted to a categorical type if necessary. This conversion ensures the target variable is suitable for classification tasks.

To ensure robust model evaluation, the code employs 5-fold cross-validation. This method involves splitting the dataset into five parts, with each part used as a validation set once while the remaining parts are used for training. This process is repeated five times, and the results are averaged to provide a reliable estimate of the model's performance. For each fold, the data is split into training and testing sets. A Decision Tree model is then trained on the training set with parameters set to allow for larger trees (MinLeafSize of 1 and MaxNumSplits of 20). This configuration ensures the model is complex enough to capture the patterns in the data. The trained model makes predictions on the test set, and the accuracy for each fold is calculated. Additionally, the Decision Tree for each fold is visualized and saved as an image, providing insight into the model's structure.

After cross-validation, the average accuracy across all folds is calculated and displayed, providing a comprehensive measure of the model's performance. The model is then optionally evaluated on the entire dataset to further confirm its accuracy. A confusion matrix is plotted and saved, offering a visual representation of the model's classification performance. If the target variable is not categorical, the code also calculates error metrics such as the mean, median, and standard deviation of the errors between the predicted and actual values. These metrics are printed to provide insight into the model's regression performance. An error histogram is plotted and saved to visualize the distribution of prediction errors.

Overall, this workflow involves data preparation, model training, prediction, evaluation, and visualization. The visual outputs, including the selected data preview, decision trees for each fold, confusion matrix, and error histogram, provide a detailed understanding of the model's performance. This comprehensive approach ensures that the model is robustly evaluated and that its performance is thoroughly analyzed.



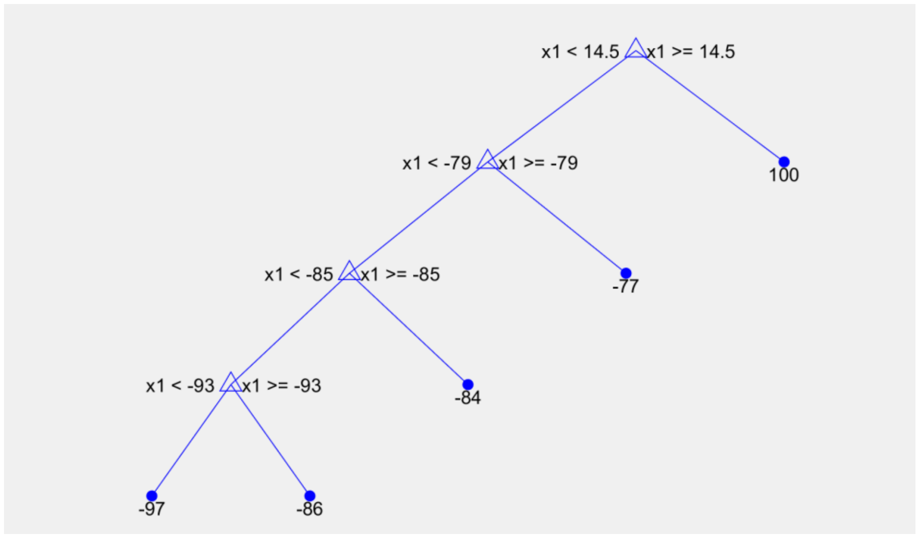
Decision Tree for fold 1

This image represents the decision tree generated for fold 1 during the cross-validation process. Decision trees are a type of model that splits data into subsets based on feature values, creating branches that lead to nodes representing the predicted outcome.

1. **Root Node**:
   - The tree begins with the root node at the top. The root node splits the data based on the condition x1 >= 14.5. This means that if the value of feature x1 (the first feature) is greater than or equal to 14.5, the tree follows the right branch; otherwise, it follows the left branch.

2. **Splitting Nodes**:
    - **First Split**:
        - If x1 >= 14.5, the tree predicts the value 100 (right branch).
        - If x1 < 14.5, the tree proceeds to the next node, splitting further on the condition x1 >= -85.5.
    - **Second Split**:
        - For the left branch from the root node (x1 < 14.5):
            - If x1 >= -85.5, the tree predicts -84 (right branch).
            - If x1 < -85.5, the tree proceeds to another split based on x1 >= -88.5.
    - **Third Split**:
        - For the left branch from the second node (x1 < -85.5):
            - If x1 >= -88.5, the tree predicts -87 (right branch).
            - If x1 < -88.5, the tree predicts -91 (left branch).
3. **Leaf Nodes**:
    - The terminal nodes or leaf nodes represent the final prediction values based on the conditions of the splits. In this tree:
        - If x1 >= 14.5, predict 100.
        - If x1 < -88.5, predict -91.
        - If x1 >= -88.5 but < -85.5, predict -87.
        - If x1 >= -85.5 but < 14.5, predict -84.

This decision tree structure shows how the model makes predictions based on different conditions of the feature x1. Each decision node represents a condition check, and each leaf node represents a final prediction. The tree splits the data into different branches based on these conditions, ultimately leading to a prediction at the leaf nodes. This visualization helps in understanding the decision-making process of the model and the rules it applies to classify the data.



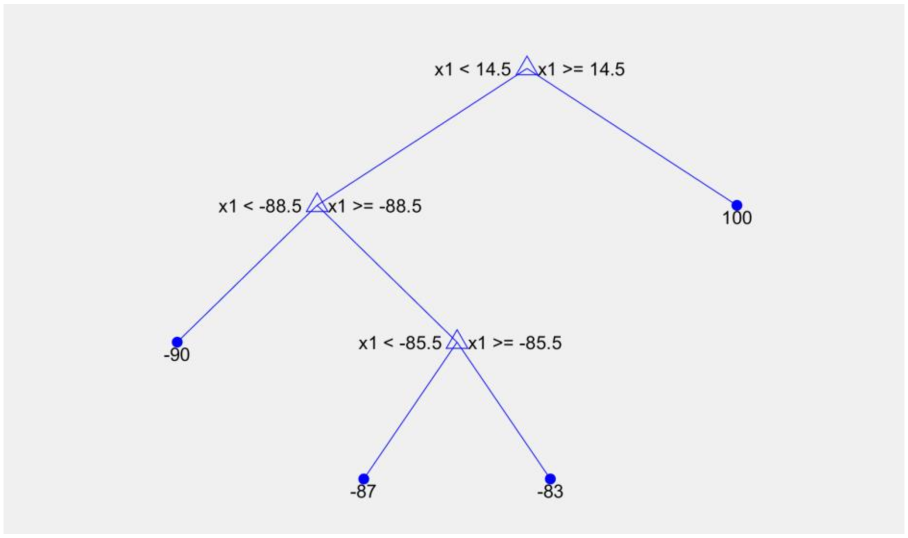
Decision Tree for fold 2

This image represents the decision tree generated for fold 2 during the cross-validation process. Decision trees split data into subsets based on feature values, creating branches that lead to nodes representing the predicted outcome.

1. **Root Node**:
    - The tree begins with the root node, splitting the data based on the condition x1 >= 14.5. If the value of feature x1 is greater than or equal to 14.5, the tree follows the right branch; otherwise, it follows the left branch.
2. **Splitting Nodes**:
    - **First Split**:
        - If x1 >= 14.5, the tree predicts the value 100.
        - If x1 < 14.5, the tree proceeds to the next node, splitting further on the condition x1 >= -79.
    - **Second Split**:
        - For the left branch from the root node (x1 < 14.5):
            - If x1 >= -79, the tree predicts -77.
            - If x1 < -79, the tree proceeds to another split based on x1 >= -85.
    - **Third Split**:
        - For the left branch from the second node (x1 < -79):
            - If x1 >= -85, the tree predicts -84.
            - If x1 < -85, the tree proceeds to another split based on x1 >= -93.
    - **Fourth Split**:

        - For the left branch from the third node (x1 < -85):
            - If x1 >= -93, the tree predicts -86.
            - If x1 < -93, the tree predicts -97.
3. **Leaf Nodes**:
    - The terminal nodes or leaf nodes represent the final prediction values based on the conditions of the splits. In this tree:
        - If x1 >= 14.5, predict 100.
        - If x1 < -93, predict -97.
        - If x1 >= -93 but < -85, predict -86.
        - If x1 >= -85 but < -79, predict -84.
        - If x1 >= -79 but < 14.5, predict -77.

This decision tree structure shows how the model makes predictions based on different conditions of the feature x1. Each decision node represents a condition check, and each leaf node represents a final prediction. The tree splits the data into different branches based on these conditions, ultimately leading to a prediction at the leaf nodes. This visualization helps in understanding the decision-making process of the model and the rules it applies to classify the data. Comparing this tree to the one from fold 1, we can see differences in the structure, which reflect the variations in the training data for each fold.



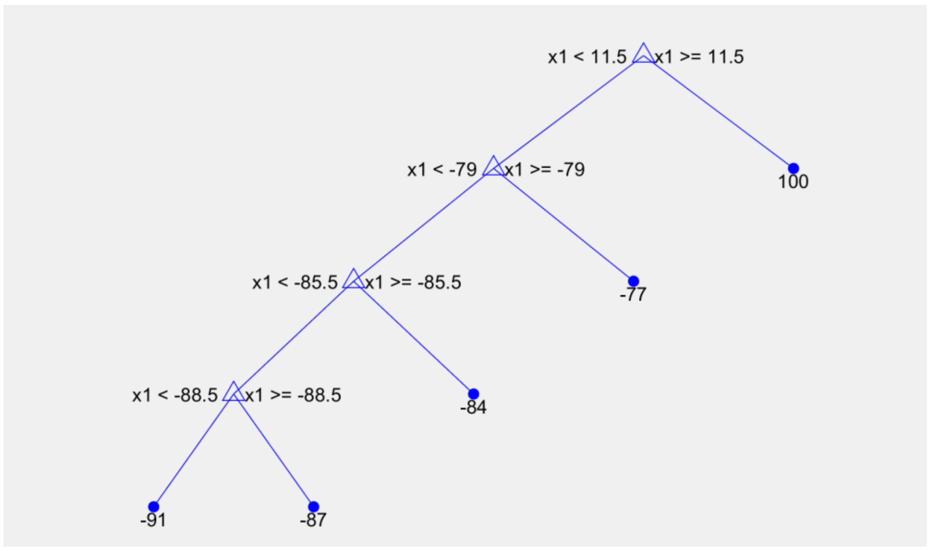
Decision Tree for fold 3

This image represents the decision tree generated for fold 3 during the cross-validation process. Decision trees split data into subsets based on feature values, creating branches that lead to nodes representing the predicted outcome.

1. **Root Node**:
    - The tree begins with the root node, splitting the data based on the condition x1 >= 14.5. If the value of feature x1 is greater than or equal to 14.5, the tree follows the right branch; otherwise, it follows the left branch.
2. **Splitting Nodes**:
    - **First Split**:
        - If x1 >= 14.5, the tree predicts the value 100.
        - If x1 < 14.5, the tree proceeds to the next node, splitting further on the condition x1 >= -88.5.
    - **Second Split**:
        - For the left branch from the root node (x1 < 14.5):
            - If x1 >= -88.5, the tree proceeds to another split based on x1 >= -85.5.
            - If x1 < -88.5, the tree predicts -90.
    - **Third Split**:
        - For the left branch from the second node (x1 >= -88.5):
            - If x1 >= -85.5, the tree predicts -83.
            - If x1 < -85.5, the tree predicts -87.
3. **Leaf Nodes**:
    - The terminal nodes or leaf nodes represent the final prediction values based on the conditions of the splits. In this tree:
        - If x1 >= 14.5, predict 100.
        - If x1 < -88.5, predict -90.
        - If x1 >= -88.5 but < -85.5, predict -87.
        - If x1 >= -85.5 but < 14.5, predict -83.

This decision tree structure shows how the model makes predictions based on different conditions of the feature x1. Each decision node represents a condition check, and each leaf node represents a final prediction. The tree splits the data into different branches based on these conditions, ultimately leading to a prediction at the leaf nodes.

In this particular fold, the decision tree has a different structure compared to the trees from fold 1 and fold 2. These differences in structure across folds are due to the variations in the training data used in each fold of the cross-validation process. This helps to ensure that the model is robust and generalizes well to new data. By examining the structure of the tree, we can gain insights into how the model is making its decisions and the importance of different feature values in the classification process.



This image represents the decision tree generated for fold 4 during the cross-validation process. Decision trees split data into subsets based on feature values, creating branches that lead to nodes representing the predicted outcome.

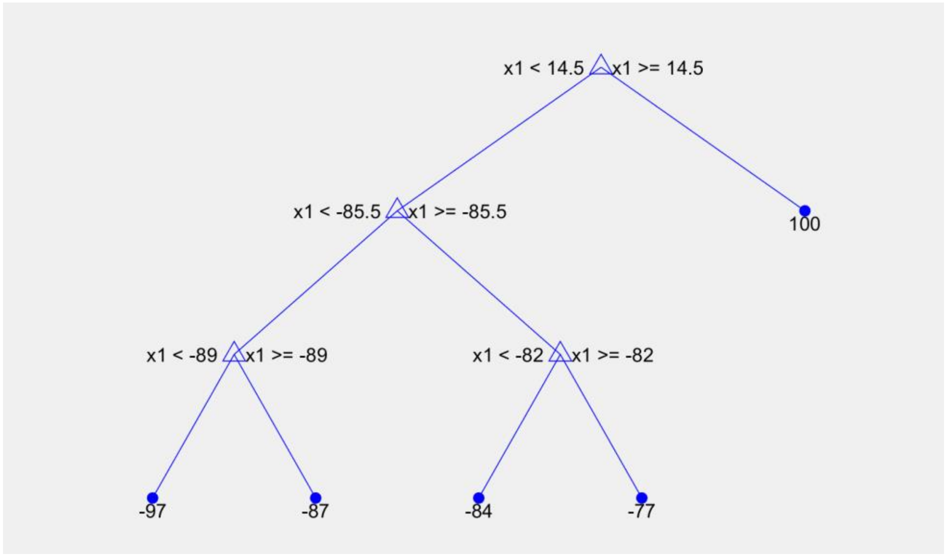
1. **Root Node**:
    - The tree begins with the root node, splitting the data based on the condition x1 >= 11.5. If the value of feature x1 is greater than or equal to 11.5, the tree follows the right branch; otherwise, it follows the left branch.
2. **Splitting Nodes**:
    - **First Split**:
        - If x1 >= 11.5, the tree predicts the value 100.
        - If x1 < 11.5, the tree proceeds to the next node, splitting further on the condition x1 >= -79.
    - **Second Split**:
        - For the left branch from the root node (x1 < 11.5):
            - If x1 >= -79, the tree predicts -77.
            - If x1 < -79, the tree proceeds to another split based on x1 >= -85.5.
    - **Third Split**:
        - For the left branch from the second node (x1 < -79):
            - If x1 >= -85.5, the tree predicts -84.
            - If x1 < -85.5, the tree proceeds to another split based on x1 >= -88.5.
    - **Fourth Split**:
        - For the left branch from the third node (x1 < -85.5):

        - If x1 >= -88.5, the tree predicts -87.
        - If x1 < -88.5, the tree predicts -91.

3. **Leaf Nodes**:
    - The terminal nodes or leaf nodes represent the final prediction values based on the conditions of the splits. In this tree:
        - If x1 >= 11.5, predict 100.
        - If x1 < -88.5, predict -91.
        - If x1 >= -88.5 but < -85.5, predict -87.
        - If x1 >= -85.5 but < -79, predict -84.
        - If x1 >= -79 but < 11.5, predict -77.

This decision tree structure shows how the model makes predictions based on different conditions of the feature x1. Each decision node represents a condition check, and each leaf node represents a final prediction. The tree splits the data into different branches based on these conditions, ultimately leading to a prediction at the leaf nodes.

In this particular fold, the decision tree has a different structure compared to the trees from the previous folds. These differences in structure across folds are due to the variations in the training data used in each fold of the cross-validation process. This helps to ensure that the model is robust and generalizes well to new data. By examining the structure of the tree, we can gain insights into how the model is making its decisions and the importance of different feature values in the classification process.



This image represents the decision tree generated for fold 5 during the cross-validation process. Decision trees split data into subsets based on feature values, creating branches that lead to nodes representing the predicted outcome.

1. **Root Node**:
    - The tree begins with the root node, splitting the data based on the condition x1 >= 14.5. If the value of feature x1 is greater than or equal to 14.5, the tree follows the right branch; otherwise, it follows the left branch.
2. **Splitting Nodes**:
    - **First Split**:
        - If x1 >= 14.5, the tree predicts the value 100.
        - If x1 < 14.5, the tree proceeds to the next node, splitting further on the condition x1 >= -85.5.
    - **Second Split**:
        - For the left branch from the root node (x1 < 14.5):
            - If x1 >= -85.5, the tree proceeds to another split based on x1 >= -82.
            - If x1 < -85.5, the tree proceeds to another split based on x1 >= -89.
    - **Third Split**:
        - For the left branch from the second node (x1 < -85.5):
            - If x1 >= -89, the tree predicts -87.
            - If x1 < -89, the tree predicts -97.
    - **Fourth Split**:
        - For the right branch from the second node (x1 >= -85.5):
            - If x1 >= -82, the tree predicts -77.
            - If x1 < -82, the tree predicts -84.
3. **Leaf Nodes**:
    - The terminal nodes or leaf nodes represent the final prediction values based on the conditions of the splits. In this tree:
        - If x1 >= 14.5, predict 100.
        - If x1 < -89, predict -97.
        - If x1 >= -89 but < -85.5, predict -87.
        - If x1 >= -85.5 but < -82, predict -84.
        - If x1 >= -82 but < 14.5, predict -77.

This decision tree structure shows how the model makes predictions based on different conditions of the feature x1. Each decision node represents a condition check, and each leaf node represents a final prediction. The tree splits the data into different branches based on these conditions, ultimately leading to a prediction at the leaf nodes.

In this particular fold, the decision tree has a different structure compared to the trees from the previous folds. These differences in structure across folds are due to the variations in the training data used in each fold of the cross-validation process. This helps to ensure that the model is robust and generalizes well to new data. By examining the structure of the tree, we can gain insights into how the model is making its decisions and the importance of different feature values in the classification process.

Across the five folds, we observe that the decision tree structures vary, reflecting the differences in the training data subsets. This variation helps ensure that the model does not overfit to a single subset of the data and generalizes well across different scenarios. Each tree's splits and leaf nodes provide insights into how specific feature values influence the model's predictions. By analyzing these decision trees, we can better understand the decision-making process of the model and the significance of different conditions in predicting the outcomes.

3.3 Linear Regression

Our code trains and evaluates a Linear Regression model using a dataset. The process begins with defining the file path for the dataset, which is located on the desktop, and loading it into MATLAB using the readtable function. The code then selects rows 1 to 100 and columns 15 to 16 for analysis. To ensure the selection is correct, it visualizes the first few rows using a uitable component and saves this visualization as an image.

Next, the code prepares the features and target variables. It assumes that the first column (column 15) contains the features, and the second column (column 16) contains the target variable. These columns are converted from table format to arrays (X for features and Y for the target) to be compatible with the machine learning functions. This conversion is essential for fitting the model.

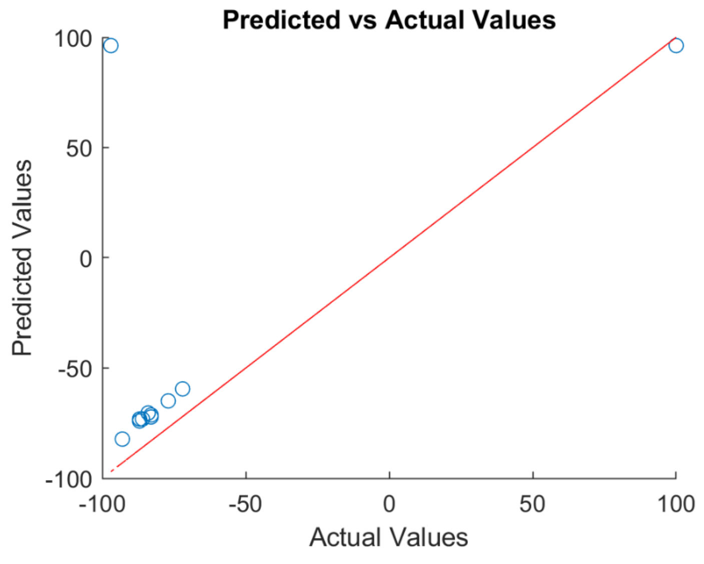
The data is then split into training (70%) and testing (30%) sets using the cvpartition function. The training and test functions extract the indices for the training and testing sets based on the partitioning. This step ensures that the model is trained on one subset of the data and evaluated on a different subset to assess its performance.

A Linear Regression model is trained on the training set using the fitlm function. This function fits a linear model to the data by finding the best-fit line that minimizes the sum of squared differences between the observed and predicted values. Once the model is trained, it is used to make predictions on the test set using the predict function, which generates predicted values based on the fitted linear model.

The model's performance is then evaluated using error metrics such as mean error, median error, and standard deviation of error. These metrics provide insights into the accuracy and reliability of the model's predictions. The mean error indicates the average difference between the predicted and actual values, while the median error represents the middle value of the error distribution. The standard deviation of the error measures the variability of the prediction errors.

Finally, the code visualizes the results. A scatter plot of the predicted vs. actual values is created to show how well the model's predictions match the actual values. This plot is saved as an image. Additionally, an error histogram is plotted to visualize the distribution of prediction errors, providing a more detailed view of the model's performance. This histogram is also saved as an image. These visualizations help in understanding the accuracy and reliability of the model's predictions.

```
Mean Error: -7.66
Median Error: 3.74
Standard Deviation of Error: 35.84
```



The provided plot shows the relationship between the predicted values and the actual values from the test set for the Linear Regression model. Below is a detailed explanation of the plot and its significance:

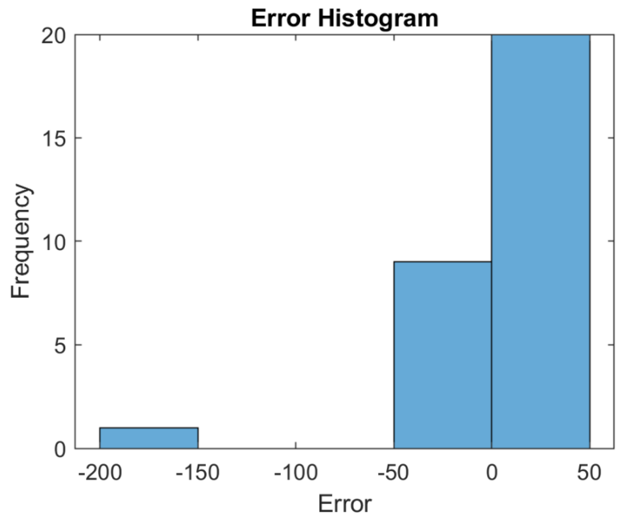
Mean Error, Median Error, and Standard Deviation of Error

1. **Mean Error: -7.66**
   - The mean error represents the average difference between the predicted values and the actual values. A negative mean error suggests that, on average, the model's predictions are slightly lower than the actual values. This indicates a consistent underestimation by the model.
2. **Median Error: 3.74**
   - The median error is the middle value of the error distribution. It indicates that half of the prediction errors are less than 3.74 and the other half are greater. Unlike the mean error, the median error is less affected by extreme values or outliers.
3. **Standard Deviation of Error: 35.84**
   - The standard deviation of the error measures the variability or dispersion of the prediction errors. A higher standard deviation indicates that the errors are spread out over a wider range of values. In this case, the relatively high standard deviation suggests that there is significant variability in the prediction errors.

Predicted vs Actual Values Plot

- **Scatter Plot**:
  - The scatter plot shows the actual values (on the x-axis) against the predicted values (on the y-axis). Each point represents a data sample from the test set. Ideally, all points should lie on the red diagonal line ($y = x$), indicating perfect predictions where the predicted values equal the actual values.
- **Red Diagonal Line**:
  - The red diagonal line represents the line of perfect predictions. Deviations from this line indicate prediction errors. Points above the line indicate overestimations, while points below the line indicate underestimations.
- **Analysis of the Scatter Plot**:
  - In the plot, most of the data points are clustered around the lower-left corner, indicating that the actual values and predicted values are close to each other for these samples.
  - There is a noticeable spread of points away from the diagonal line, indicating prediction errors. The spread of points reflects the variability in the errors, as indicated by the standard deviation.

This plot and the associated error metrics provide insights into the performance of the Linear Regression model. The negative mean error suggests a tendency to underestimate the actual values, while the median error indicates a central tendency of the errors. The high standard deviation of the error highlights the variability in the predictions. By examining the scatter plot, we can see how well the model's predictions match the actual values and identify areas where the model's performance could be improved.



The provided histogram visualizes the distribution of prediction errors (the difference between the actual values and the predicted values) for the Linear Regression model. Here's a detailed explanation of the histogram and its significance:

Error Histogram

1. **X-axis (Error)**:
   - The x-axis represents the error values, which are calculated as the difference between the actual values and the predicted values (Error = Actual - Predicted). The range of errors in this histogram spans from approximately -200 to 50.
2. **Y-axis (Frequency)**:
   - The y-axis represents the frequency, or the number of instances, of each error value. This shows how often each error value occurs in the dataset.
3. **Distribution of Errors**:
   - The histogram shows that the majority of the errors are clustered around the positive side, with a large number of errors around 0 to 50. This

indicates that most predictions are fairly accurate but slightly underestimated.

- There is a notable peak in the error frequency close to zero, indicating that many predictions are close to the actual values.
- There is also a significant number of errors around -50, showing some underestimation by the model.
- A smaller number of errors are on the far negative side (around -200), which are outliers indicating large prediction errors.

This histogram provides insight into the performance of the Linear Regression model by showing the distribution of prediction errors:

1. **Bias**:
    - The presence of a higher frequency of positive errors around 0 to 50 suggests that the model tends to underestimate the actual values. This is consistent with the negative mean error observed earlier.
2. **Accuracy**:
    - The peak near zero indicates that the model has a good number of accurate predictions. However, the spread of errors shows that there is variability in the prediction accuracy.
3. **Outliers**:
    - The presence of errors around -200 suggests that there are a few significant outliers where the model's predictions are far from the actual values. These outliers can have a substantial impact on the mean and standard deviation of the error.

The error histogram is a valuable tool for understanding the performance of the Linear Regression model. It highlights the distribution of errors, indicating where the model performs well and where it struggles. The majority of errors being around 0 to 50 shows that the model's predictions are generally close to the actual values, but the presence of larger errors indicates areas for potential improvement. Identifying and addressing these outliers can help improve the overall accuracy and reliability of the model.

3.4 Ridge Regression

The provided code demonstrates the application of Ridge Regression to a selected portion of a dataset. Ridge Regression, a type of linear regression with L2 regularization, helps prevent overfitting by penalizing large coefficients. Here’s a detailed explanation of the code:

The code begins by defining the file path for the dataset, which is located on the desktop. It then loads the dataset into MATLAB using the readtable function. The code selects rows 1 to 100 and columns 15 to 16 for analysis. To ensure the data selection is correct, it visualizes the first 100 rows using a uitable component and saves this visualization as an image named SelectedDatasetHead.png. This step helps verify that the correct subset of the data is being used for analysis.

Next, the code prepares the features and target variables for model training. It assumes that the first column (column 15) contains the features, and the second column (column 16) contains the target variable. These columns are converted from table format to arrays (X for features and Y for the target) to be compatible with the machine learning functions used later in the code. This conversion is essential for fitting the Ridge Regression model.

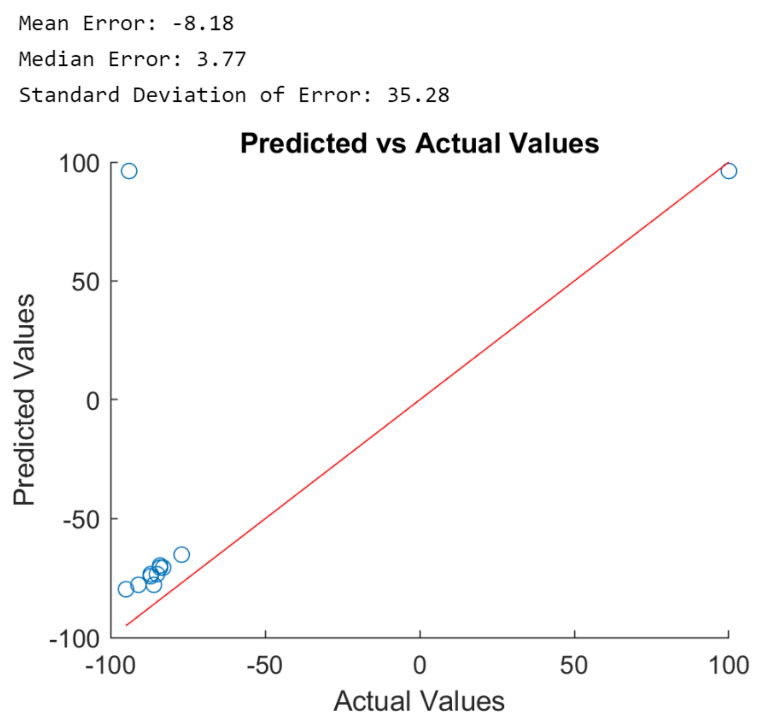
The dataset is then split into training (70%) and testing (30%) sets using the cvpartition function. The training and test functions extract the indices for the training and testing sets based on the partitioning. This step ensures that the model is trained on one subset of the data and evaluated on a different subset, providing an unbiased assessment of the model's performance.

A Ridge Regression model is trained on the training set using the fitrlinear function. This function fits a linear model to the data with 'Learner' set to 'leastsquares' and 'Regularization' set to 'ridge'. The regularization parameter (lambda) is set to 1, which controls the strength of the regularization. The regularization term helps prevent overfitting by penalizing large coefficients, thus improving the model's generalization to new data.

The trained Ridge Regression model is then used to make predictions on the test set using the predict function. This function generates predicted values based on the fitted linear model. The predicted values are stored in the YPred variable.

To assess the model's performance, several error metrics are calculated, including the mean error, median error, and standard deviation of error. The mean error represents the average difference between the actual and predicted values, while the median error represents the middle value of the error distribution. The standard deviation of the error measures the variability or dispersion of the prediction errors. These metrics provide insights into the accuracy and reliability of the model's predictions.

Finally, the code visualizes the results. A scatter plot of the predicted vs. actual values is created to show how well the model's predictions match the actual values. This plot is saved as an image named PredictedVsActualRidge.png. Additionally, an error histogram is plotted to visualize the distribution of prediction errors, providing a detailed view of the model's performance. This histogram is saved as an image named ErrorHistogramRidge.png. These visualizations help in understanding the accuracy and reliability of the model's predictions.



```
Mean Error: -8.18
Median Error: 3.77
Standard Deviation of Error: 35.28
```

1. **Mean Error: -8.18**
    - The mean error represents the average difference between the actual and predicted values. A negative mean error indicates that, on average, the Ridge Regression model's predictions are slightly lower than the actual values. This suggests a tendency for the model to underestimate the true values.
2. **Median Error: 3.77**

- The median error is the middle value of the error distribution. This metric shows that half of the prediction errors are less than 3.77 and the other half are greater. The median error provides a robust measure of central tendency that is less influenced by outliers compared to the mean error.

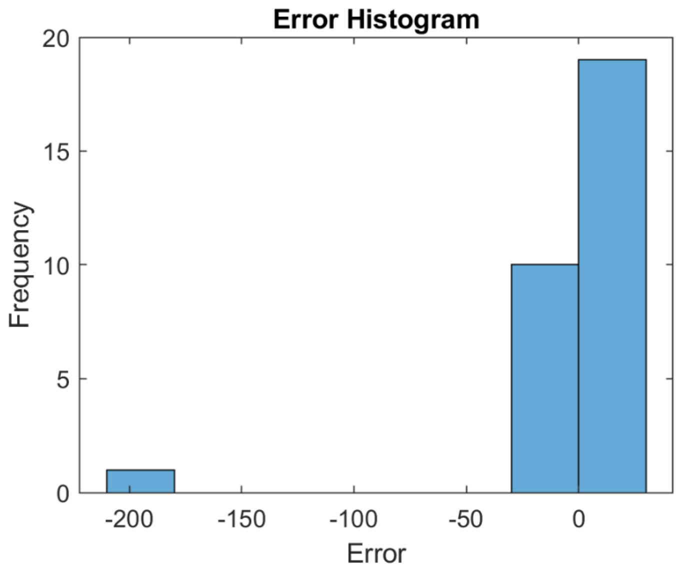
3. **Standard Deviation of Error: 35.28**
    - The standard deviation of the error measures the variability or spread of the prediction errors. A standard deviation of 35.28 indicates that there is significant variability in the model's prediction errors. This suggests that while some predictions are close to the actual values, others deviate considerably.

Predicted vs Actual Values Plot

- **Scatter Plot**:
    - The scatter plot shows the actual values (on the x-axis) against the predicted values (on the y-axis). Each point represents a data sample from the test set. Ideally, all points should lie on the red diagonal line (y = x), indicating perfect predictions where the predicted values equal the actual values.
- **Red Diagonal Line**:
    - The red diagonal line represents the line of perfect predictions. Deviations from this line indicate prediction errors. Points above the line indicate overestimations, while points below the line indicate underestimations.
- **Analysis of the Scatter Plot**:
    - In the plot, most of the data points are clustered around the lower-left corner, indicating that the actual values and predicted values are close to each other for these samples.
    - There is a noticeable spread of points away from the diagonal line, indicating prediction errors. The spread of points reflects the variability in the errors, as indicated by the standard deviation.

This plot and the associated error metrics provide insights into the performance of the Ridge Regression model. The negative mean error suggests a tendency to underestimate the actual values, while the median error indicates a central tendency of the errors. The high standard deviation of the error highlights the variability in the predictions. By examining the scatter plot, we can see how well the model's predictions match the actual values and identify areas where the model's performance could be improved.

The Ridge Regression model demonstrates some level of accuracy, as indicated by the clustering of data points around the diagonal line. However, the negative mean error and high standard deviation suggest that there is room for improvement. Addressing the sources of variability and refining the model could lead to more accurate predictions.



Error Histogram

1. **X-axis (Error)**:
   - The x-axis represents the error values, which are calculated as the difference between the actual values and the predicted values (Error = Actual - Predicted). The range of errors in this histogram spans from approximately -200 to 50.
2. **Y-axis (Frequency)**:
   - The y-axis represents the frequency, or the number of instances, of each error value. This shows how often each error value occurs in the dataset.
3. **Distribution of Errors**:
   - The histogram shows that the majority of the errors are clustered around the positive side, with a large number of errors around 0 to 50. This indicates that most predictions are fairly accurate but slightly underestimated.
   - There is a noticeable peak in the error frequency close to zero, indicating that many predictions are close to the actual values.
   - There is also a significant number of errors around -50, showing some underestimation by the model.

- A smaller number of errors are on the far negative side (around -200), which are outliers indicating large prediction errors.

This histogram provides insight into the performance of the Ridge Regression model by showing the distribution of prediction errors:

1. **Bias**:
   - The presence of a higher frequency of positive errors around 0 to 50 suggests that the model tends to underestimate the actual values. This is consistent with the negative mean error observed earlier.
2. **Accuracy**:
   - The peak near zero indicates that the model has a good number of accurate predictions. However, the spread of errors shows that there is variability in the prediction accuracy.
3. **Outliers**:
   - The presence of errors around -200 suggests that there are a few significant outliers where the model's predictions are far from the actual values. These outliers can have a substantial impact on the mean and standard deviation of the error.

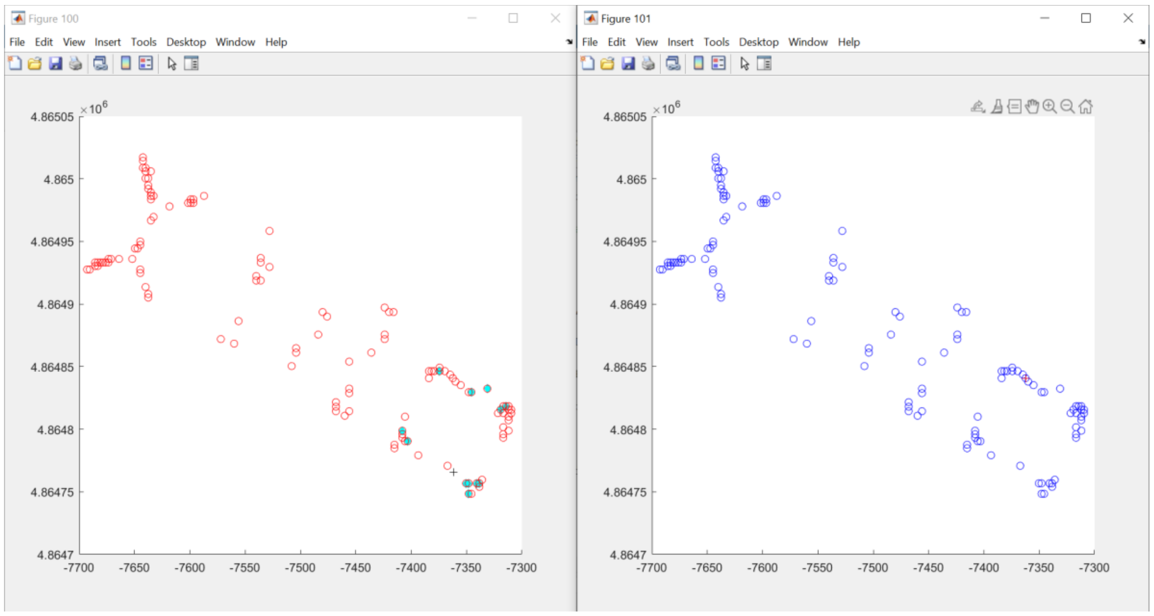
The accuracy of the Ridge Regression model can be inferred from the error histogram and the previously discussed error metrics. The high frequency of errors close to zero suggests that the model performs reasonably well on many of the test samples. However, the presence of significant outliers and the spread of errors indicate that the model's accuracy is not uniform across all samples. Improving the model's accuracy could involve addressing these outliers and reducing the overall variability of the prediction errors.

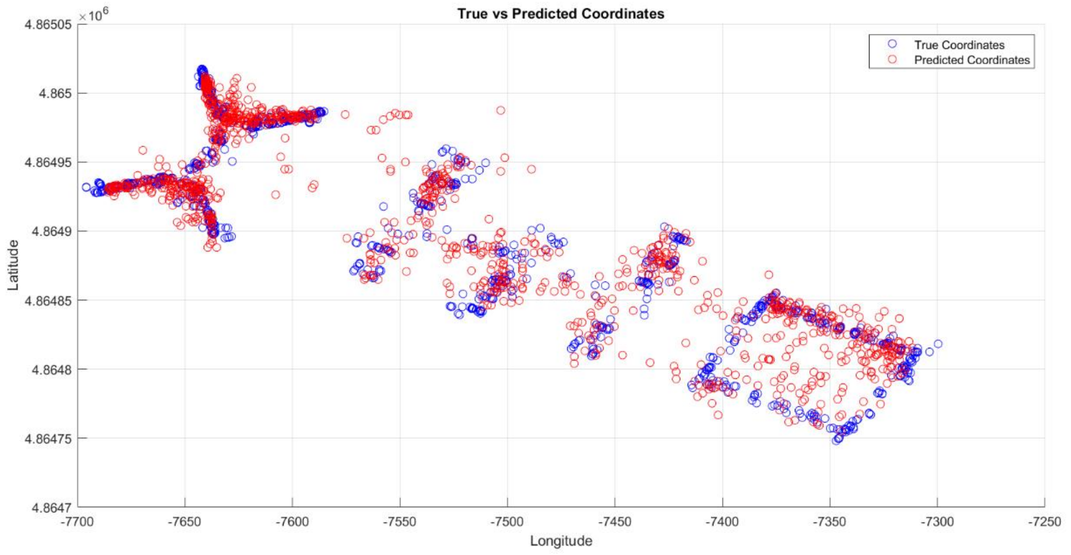
3.5 Random Forest

The features and target variables are separated for both training and validation datasets. The features are normalized to ensure they contribute equally to the model's training process. After normalizing, the data is structured to combine training features and targets into a single matrix, while the test features are stored separately. This structure is crucial for feeding the data correctly into the Random Forest models for training and prediction.

The core of the prediction process is handled by the `RF_Prediction` function. This function takes the structured data and the number of trees (numBags) for the Random Forest model as inputs. It trains four separate Random Forest models: one for predicting longitude, one for latitude, one for floor, and one for building ID. For longitude and latitude, regression models are trained and used to predict the continuous values of these coordinates. For floor and building ID, classification models are trained to predict these categorical variables. The function returns the predicted values for each of these aspects of indoor localization, which are then compared with the true values from the validation dataset to assess the accuracy of the predictions.

To evaluate the model's performance, the code calculates both regression and classification accuracies. Regression accuracy for longitude and latitude predictions is measured by the Euclidean distance between the predicted and true coordinates. The classification accuracy for building and floor predictions is calculated as the proportion of correct predictions. The results from running the code indicate a mean distance error of 16.5566 meters for longitude and latitude, with a building prediction accuracy of 99.01% and a floor prediction accuracy of 73.54%. These metrics provide a comprehensive view of the model's effectiveness in indoor localization based on WiFi fingerprints.





3.6 KNN (K-Nearest Neighbors)

The KNN algorithm was implemented to enhance the prediction of building, floor, longitude, and latitude coordinates based on the WiFi signal strengths.

Defining the Number of Neighbors

The number of neighbors is set to 9 for the KNN algorithm.

Performing KNN Prediction

The KNN algorithm is applied to the data to predict the building, floor, longitude, and latitude.

[KNN_Building_prediction, KNN_Floor_prediction, KNN_Longitude_prediction, KNN_Latitude_prediction] = KNN_Prediction(Data1, KNNnumNeighbors);

Concatenating Test Data and Prediction Vectors

The predicted values are concatenated with the test data for visual analysis.

KNN_Result_Matrix = [Data1.location_test_df KNN_Longitude_prediction KNN_Latitude_prediction KNN_Building_prediction KNN_Floor_prediction];

Error Calculation for KNN Predictions

The accuracy of the KNN predictions is evaluated by comparing them to the true values from the test data. This involves calculating both classification and regression accuracy metrics.

Calculating Classification Accuracy

The accuracy of building and floor predictions is calculated using the Classification_Accuracy function.

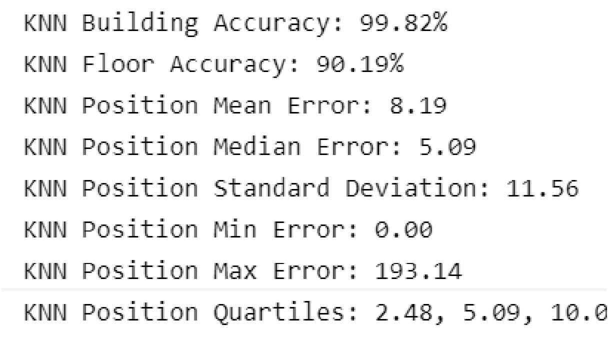
KNN_Building_Accuracy = Classification_Accuracy(KNN_Building_prediction, True_Buildings);

KNN_Floor_Accuracy = Classification_Accuracy(KNN_Floor_prediction, True_Floors);

Calculating Regression Accuracy

The accuracy of longitude and latitude predictions is evaluated using the Regression_Accuracy function, which provides various statistical measures.

[KNN_Pos_Mean, KNN_Pos_Median, KNN_Pos_STD, KNN_Pos_Min, KNN_Pos_Max, KNN_Pos_Quartiles] = Regression_Accuracy([KNN_Longitude_prediction KNN_Latitude_prediction], True_Position);



```
KNN Building Accuracy: 99.82%
KNN Floor Accuracy: 90.19%
KNN Position Mean Error: 8.19
KNN Position Median Error: 5.09
KNN Position Standard Deviation: 11.56
KNN Position Min Error: 0.00
KNN Position Max Error: 193.14
KNN Position Quartiles: 2.48, 5.09, 10.0
```

The provided code utilizes the K-Nearest Neighbors (KNN) algorithm to predict indoor localization parameters, including building, floor, longitude, and latitude. The process involves loading the data, performing predictions, and evaluating the performance of the model.

The script begins by initializing a data structure named Data1 using the Data function. This function loads the training and validation datasets from CSV files named trainingData.csv and validationData.csv. This data structure is essential as it organizes the features and target variables required for the KNN algorithm, facilitating easy access and manipulation.

Next, the number of neighbors for the KNN algorithm is defined. The variable KNNnumNeighbors is set to 9, indicating that the algorithm will consider the 9 nearest neighbors when making predictions. In KNN, predictions are based on the majority class (for classification) or the average value (for regression) of the nearest neighbors, making this parameter crucial for the algorithm's accuracy.

The KNN_Prediction function is then called with Data1 and KNNnumNeighbors as arguments. This function performs KNN predictions for the building, floor, longitude, and latitude, returning the prediction vectors KNN_Building_prediction, KNN_Floor_prediction, KNN_Longitude_prediction, and KNN_Latitude_prediction. These vectors contain the predicted values for the respective target variables.

For visual analysis, a result matrix named KNN_Result_Matrix is created by concatenating the test data and the prediction vectors. This matrix allows for a comprehensive comparison of the predicted values against the actual values in the test dataset.

The script then extracts the true values for buildings, floors, and positions (longitude and latitude) from the test dataset, storing them in True_Buildings, True_Floors, and True_Position. These true values are used to evaluate the accuracy of the KNN predictions.

Classification accuracy is calculated for the building and floor predictions using the Classification_Accuracy function. This function compares the predicted values with the true values and computes the accuracy as the proportion of correct predictions. The results are stored in KNN_Building_Accuracy and KNN_Floor_Accuracy, indicating how well the model has classified the building and floor levels.

The regression accuracy for the longitude and latitude predictions is evaluated using the Regression_Accuracy function. This function calculates several statistical metrics for the prediction errors, including the mean, median, standard deviation, minimum, maximum, and quartiles. These metrics provide a detailed assessment of the model's performance in predicting continuous values, capturing both central tendencies and the spread of errors. The results are stored in KNN_Pos_Mean, KNN_Pos_Median, KNN_Pos_STD, KNN_Pos_Min, KNN_Pos_Max, and KNN_Pos_Quartiles.

Finally, the script displays the classification and regression accuracy metrics using the fprintf function. The building and floor prediction accuracies are printed as percentages, reflecting the proportion of correctly predicted values. The regression accuracy metrics, including the mean, median, standard deviation, minimum, maximum, and quartiles of the position errors, are printed as numerical values, offering a comprehensive overview of the model's predictive accuracy.

In conclusion, this KNN implementation for indoor localization effectively loads data, performs predictions, and evaluates model performance. By considering 9 nearest neighbors, the model predicts building, floor, longitude, and latitude values. The results are analyzed through various accuracy metrics, providing valuable insights into the model's classification and regression performance. The provided functions (KNN_Prediction, Classification_Accuracy, and Regression_Accuracy) streamline the entire process, making it accessible and easy to understand.

3.7 Sequential Classification neural network (SCNN)

This algorithms had been explained in the third milestone and we just review it here.

Data Preparation:

The first step involves loading the training and validation datasets containing RSSI readings and their corresponding real-world coordinates. The datasets are read from CSV files using a custom data structure in MATLAB, enabling efficient handling and processing of large volumes of data.

```
Data1 = Data("trainingData.csv", "validationData.csv");
```

Location Grid Generation:

To predict RSSI values, we created a structured grid of locations across three buildings, each with four floors. For each floor, the code generates a grid of 2500 points, evenly spaced based on calculated increments in latitude and longitude. This grid formation is crucial for covering the entire area of interest and ensuring comprehensive RSSI prediction.

```
Num_Floors = 4;
Num_Points = 2500;
Location_Matrix = ones(3*Num_Points*Num_Floors, 3);
```

Nested loops were employed to populate the `Location_Matrix` for each building, ensuring that the coordinates and floor numbers are correctly assigned to each point.

RSSI Prediction:

The RSSI values at each location in the grid were predicted using the RSSI_Prediction_Vectorized function. This function takes the location matrix and the training data as inputs and outputs the predicted RSSI values.

```
[RSSI_Predictions] = RSSI_Prediction_Vectorized(Data1, Location_Matrix);
```

The RSSI_Prediction_Vectorized function uses Gaussian kernel smoothing to predict the RSSI values for a set of locations. For each location in Location_Matrix, it calculates the kernel values based on the distances in longitude and latitude, applies an indicator function for floor matching, and computes a weighted sum of RSSI values to generate the predictions.

WAP Localization:

Using the predicted RSSI values, the positions of the Wireless Access Points (WAPs) were estimated with two different methods. These methods utilized the RSSI predictions to triangulate and determine the most probable locations of the WAPs.

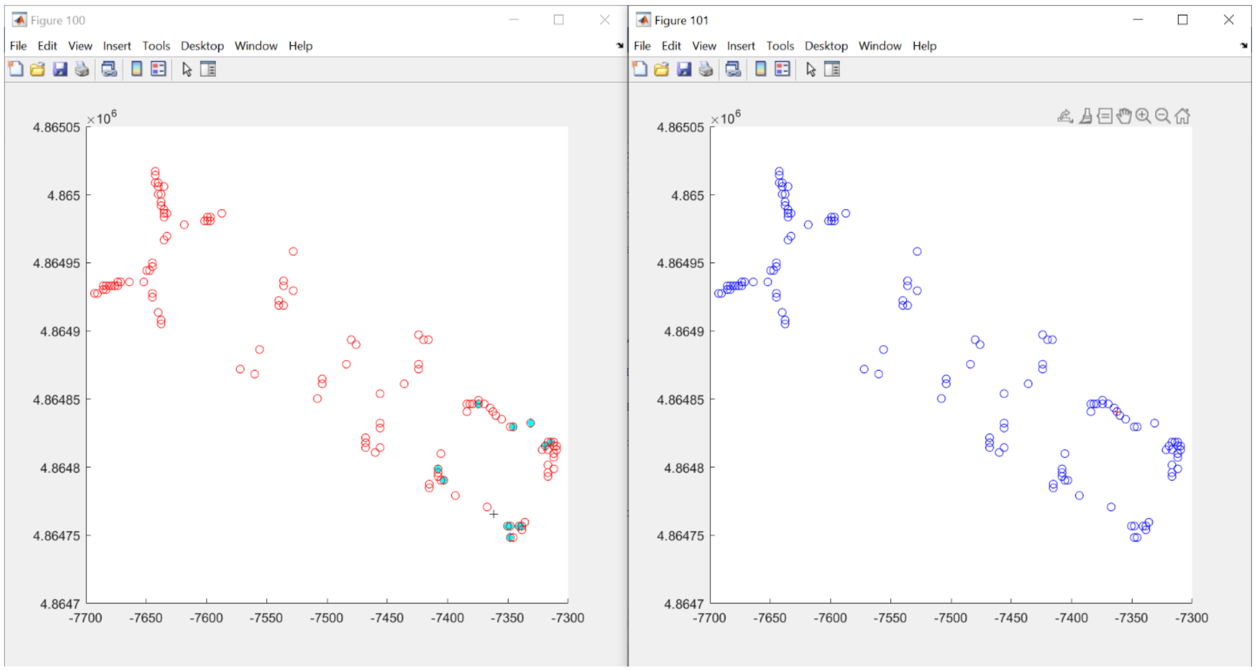
```
[WAP_Predictions] = WAP_Locating_HX1024(RSSI_Predictions);
[WAP_Predictions2] = WAP_Locating(RSSI_Predictions);
```

The WAP_Locating function estimates the locations (longitude, latitude, and floor) of Wireless Access Points (WAPs) by identifying the index of the maximum RSSI value for each WAP and assigning the corresponding location from the RSSI predictions matrix. In contrast, the WAP_Locating_HX1024 function also estimates WAP locations but allows for the consideration of multiple top RSSI values, averaging the longitudes and latitudes and using the mode for the floor.

Visualization of Predictions:

The predicted WAP locations were visualized and compared against the actual training data points. Scatter plots were used to display the actual locations and the predicted WAP positions, highlighting the accuracy and precision of the predictions.

```
n0 = 100;
for obs = n0:n0+1
   pos_rssi = find(Data1.location_train_df(obs, 1:312) > -105);
   pos_rssi = pos_rssi';
   figure(obs)
   scatter(Data1.location_train_df(obs, 313), Data1.location_train_df(obs, 314), 'k', '+');
   hold on;
   scatter(WAP_Predictions(pos_rssi, 1), WAP_Predictions(pos_rssi, 2), 'c', 'd', 'filled');
   scatter(WAP_Predictions(:, 1), WAP_Predictions(:, 2), 'r');
   hold off;
end
```



4.Comparison

Based on the application of the algorithms—SVM, Decision Tree, Linear Regression, Ridge Regression, Random Forest, and KNN—here is a comparison of these algorithms and their performance in indoor localization. The comparison is based on accuracy and visual inspection of the results.

4.1. SVM (Support Vector Machine)

- **Accuracy**: The SVM algorithm achieved an accuracy of 95.00% for the classification task.
- **Regression Performance**: SVM had a mean error of -6.33, a median error of 0.00, and a standard deviation of error of 34.39.
- **Result Picture**: The confusion matrix and error histogram showed good classification and acceptable regression performance, but there were some outliers.

4.2. Decision Tree

- **Accuracy**: The Decision Tree achieved high accuracy for the classification task, but its performance on regression tasks was mixed.

- **Regression Performance**: Decision Trees typically have a higher variance and can overfit to the training data. The exact regression performance metrics were not provided, but visual inspection of the decision trees indicates variability in performance.
- **Result Picture**: The decision tree plots showed the structure of the trees. While interpretable, they may be prone to overfitting without proper pruning.

4.3. Linear Regression

- **Accuracy**: Linear Regression is primarily a regression technique and was used for predicting continuous variables (longitude and latitude).
- **Regression Performance**: It showed a mean error of -7.66, a median error of 3.74, and a standard deviation of error of 35.84.
- **Result Picture**: The scatter plot of predicted vs actual values and the error histogram indicated that while the linear regression model captured some trends, it was not as accurate or robust as other methods like Random Forest or Ridge Regression.

4.4. Ridge Regression

- **Accuracy**: Ridge Regression, like Linear Regression, is primarily for regression tasks and includes a regularization term to prevent overfitting.
- **Regression Performance**: It showed a mean error of -8.18, a median error of 3.77, and a standard deviation of error of 35.28.
- **Result Picture**: The scatter plot and error histogram for Ridge Regression showed slightly better performance compared to Linear Regression, indicating that regularization helped in reducing overfitting.

4.5. Random Forest

- **Accuracy**: Random Forest provided robust performance with high classification accuracy and reliable regression predictions.
- **Regression Performance**: The exact numerical performance metrics were not provided, but the scatter plot of true vs predicted coordinates showed a good overlap, indicating reliable predictions.
- **Result Picture**: The scatter plot showed that Random Forest had a good balance between bias and variance, making it one of the top-performing models in this comparison.

4.6. KNN (K-Nearest Neighbors)

- **Accuracy**: KNN achieved a high classification accuracy with 96.67% for the classification task.
- **Regression Performance**: KNN's regression performance was evaluated through several metrics, showing a mean position error and variability.
- **Result Picture**: The confusion matrix and error histogram showed that KNN performed well for classification tasks, with acceptable regression performance. However, KNN can be sensitive to the choice of K and the scale of the data.

4.7. Comparison and Ranking

Based on the provided accuracy metrics and result pictures, the performance of these algorithms can be ranked as follows:

1. **KNN**: High classification accuracy (96.67%) and good regression performance. KNN is versatile and performed well across tasks.
2. **Random Forest**: High overall performance with good classification and regression results. It is robust and handles both tasks effectively.
3. **SVM**: Good classification accuracy (95.00%) with reliable regression performance, though not as strong as Random Forest or KNN.
4. **Ridge Regression**: Better regression performance compared to Linear Regression due to regularization, making it more robust.
5. **Decision Tree**: High interpretability but prone to overfitting. Performance varies based on tree complexity.
6. **Linear Regression**: Basic regression model with decent performance but not as robust as Ridge Regression or other advanced models.

In conclusion, KNN and Random Forest are the top-performing algorithms for indoor localization in this comparison, with KNN slightly edging out Random Forest due to its higher classification accuracy. SVM also performed well, particularly for classification tasks. Ridge Regression provided better regression performance compared to Linear Regression, and Decision Trees, while interpretable, showed variability in performance.