Optimization of classifiers for person detection

Course M.Eng Information Technology

Module Individual Project

By Dr. Andreas Pech

Shiva Kumar Biru
Mat No: 1436929
shiva.biru@stud.fra-uas.de

***Abstract* - Accurate detection of individuals in office environments is crucial for optimizing workspace utilization .Current methods often struggle with accuracy, particularly in dynamic office settings where seating occupancy can change frequently. This project aims to address these challenges by optimizing machine-learning classifiers for person detection. The proposed solution focuses on improving the accuracy of classifiers that differentiate between occupied and unoccupied office seats. The project will employ several advanced machine-learning algorithms, including Random Forest Classifier (RFC), XG Boost, Gradient Boosting, Logistic Regression, and Support Vector Machines (SVM). These classifiers will be evaluated and optimized to ensure the highest possible accuracy. The optimized classifier will be integrated into an executable application featuring a user-interactive graphical user interface (GUI).**

Keywords—Red Pitaya, Ultrasonic Sensor SRF02, Fast Fourier Transform, Machine Learning, Supervised Learning, Random Forest Classifier, XGBoost, Gradient Boosting, Logistic Regression, and Support Vector Machines (SVM), GUI, Confusion Matrix, panel

I. INTRODUCTION

Accurate detection of individuals within office environments is essential for optimizing workspace utilization and enhancing energy efficiency. Traditional detection methods often lack the precision needed to handle dynamic office settings, where the occupancy status of seats can frequently change. This project aims to address these challenges by optimizing machine-learning classifiers for person detection using ultrasonic reflections.

The primary objective is to develop and refine a machine-learning model capable of accurately classifying ultrasonic reflections from both occupied and unoccupied office seats. This model will be integrated into an executable application with a graphical user interface (GUI) to facilitate user interaction and practical deployment in real-world office environments.

To achieve these goals, the study will employ several advanced machine-learning algorithms, including Random Forest Classifier (RFC), Support Vector Machines (SVM), XGBoost, Gradient Boosting, and Logistic Regression. These algorithms will be evaluated and optimized to ensure the highest possible accuracy in person detection. After adjusting the output readings and gathering the necessary datasets during the project, utilized these measurements to construct numerous confusion matrices for our study

By enhancing the accuracy and reliability of person detection systems in office settings, this research aims to contribute to better resource management and operational efficiency, ultimately leading to significant cost savings and environmental benefits

II. METHODOLOGY

The theoretical foundation of the experiment is outlined in the preceding section, encompassing explanations of the Ultrasonic Red Pitaya sensor, FFT and ADC data analysis methodologies, background information on Machine Learning algorithms, and the concept of Confusion matrices.

A. Ultrasonic sensor and Red Pitaya.

The Red Pitaya STEM Lab board incorporates a system-on-a-chip (SoC) [2] from Xilinx[1]. In this setup, the Ultrasonic Sensor SRF02 [Fig.2] is utilized, which operates at an average frequency of 40 kHz and has a power output of 150 mW according to the manufacturer's specifications. Unlike dual transducer systems, the SRF02 uses a single transducer for both transmitting and receiving signals, resulting in a minimum detection range of approximately 15 cm. [3]

It operates with a 5V grounded power supply. The Red Pitaya embedded system [Fig.3], featuring a sampling frequency of 1.95 MS/s and a resolution of 14 bits, manages the SRF02 and converts the received signal from analog to digital form. Additionally, the Red Pitaya device supports wireless data transmission to a laptop for extra processing.

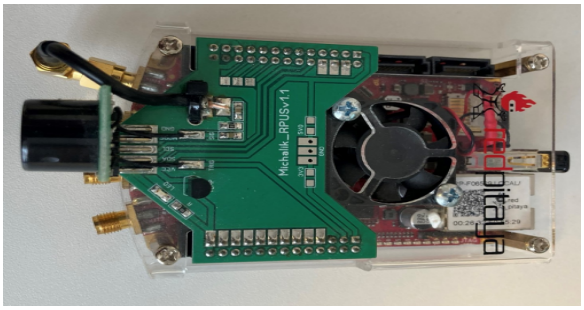

Fig.1. Ultrasonic sensor and red pitaya device

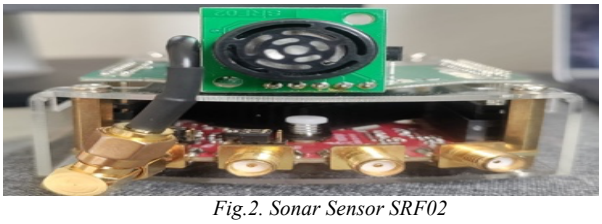

Fig.2. Sonar Sensor SRF02

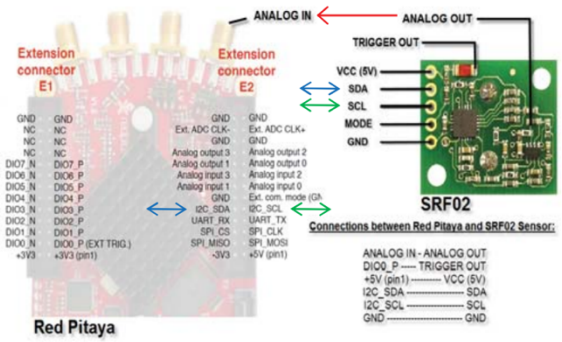

Fig.3.Interface between Embedded System and Sensor

It runs on the GNU/Linux operating system, allowing for the management and logging of data on a computer. The main Red Pitaya unit comes with 16 standard I/O ports, as well as two analog RF input and output ports. It also has a micro-SD card slot, an RJ45 Ethernet port, a USB port, and a micro-USB connector for console access. The Red Pitaya is capable of receiving and transmitting radio frequency signals within a 50MHz frequency range.

B. Analog to Digital Converter

The Analog-to-Digital Converter (ADC) acts as a critical link between the continuous analog signals and the discrete digital world. By converting continuous analog signals into digital forms, the ADC facilitates the processing and analysis of these signals through computational methods.

In our study, the data from the ADC in the Ultrasonic Red Pitaya sensor offers valuable insights into the nature of the captured signals. Through detailed analysis of this digitized data, we aim to gain important information about the observed environment, with a particular emphasis on distance measurements. By utilizing ADC data analysis techniques like signal filtering and sampling, our primary goal was to improve the precision and dependability of our sensor system.

C. Fast Fourier Transform

To generate output in the desired numerical discrete format, the Red Pitaya server utilizes the Fast Fourier Transform (FFT). To fully grasp the project's requirements, it is essential to have a basic understanding of the FFT.

The Fast Fourier Transform (FFT) is a highly efficient algorithm for calculating the discrete Fourier transform (DFT) of a sequence, allowing for the conversion of a signal from the time domain to the frequency domain. By using a divide-and-conquer strategy, the FFT algorithm breaks down a DFT of size N into two smaller DFTs of size N/2 and then combines the results in O(N) time, offering a significant speed advantage over the basic O(N^2) approach.

The FFT is widely used in various fields such as signal processing, data compression, and image processing, where analyzing or manipulating signals in the frequency domain is essential. It has become a fundamental tool in many scientific and engineering applications.

D. Confusion Matrix:

A confusion matrix is a common classification output. It is a table used to evaluate the performance of a classification method by comparing the expected labels of a dataset with the actual labels. It serves as a standard tool in statistics and machine learning for assessing the accuracy of a classification model.

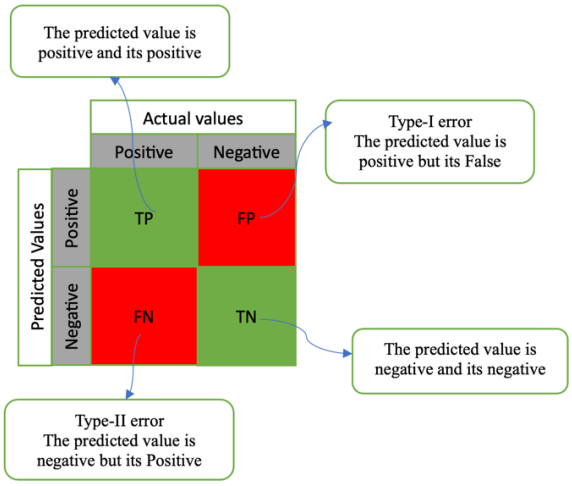

Fig.4. confusion matrix

The above figure 4 consists of parameters which are calculated based on four core parameters:

• **True Positive (TP)**: This occurs when the model correctly predicts a positive outcome that matches the actual positive value.

• **True Negative (TN)**: This represents situations where the model accurately predicts a negative outcome, consistent with the actual negative value.

• **False Positive (FP)**: This happens when the model incorrectly predicts a positive outcome while the actual value is negative, representing the first type of prediction error. This is also referred to as a Type-I error.

• **False Negative (FN)**: This occurs when the model mistakenly predicts a negative outcome while the actual value is positive, representing the second type of prediction error. This is also known as a Type-II error. These metrics are crucial for calculating additional parameters that are important for assessing the performance of the classification model.

From the confusion matrix, various crucial metrics can be computed to assess the classification algorithm's performance, including:

Accuracy:

Accuracy measures the ratio of correctly predicted samples to the total number of samples in the dataset, reflecting the overall effectiveness of the classifier. The formula for accuracy is:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Recall (Sensitivity):

Recall evaluates the proportion of correctly identified positive samples out of the total number of actual positive samples in the dataset, highlighting the model's ability to detect true positives. The formula for recall is:

$$\text{Recall} = \frac{TP}{(TP + FN)}$$

Precision:

Precision measures the proportion of correctly predicted positive samples out of all the samples the model predicted as positive, showcasing the model's accuracy in identifying true positives. The formula for precision is:

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

Specificity:

Specificity assesses the proportion of accurately predicted negative samples out of the total number of actual negative samples in the dataset, reflecting the model's ability to correctly identify true negatives. The formula for specificity is:

$$\text{Specificity} = \frac{TN}{(TN + FP)}$$

F1 Score:

The F1 score, representing the harmonic mean of precision and recall, offers a balanced metric when both precision and recall are essential. The F1 score formula is:

$$\text{F1 Score} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

The confusion matrix is a key tool for assessing a classification model's performance. It records the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) based on the model's predictions. These counts are used to calculate various performance metrics.

True Positive Rate (TPR) or Sensitivity:

TPR (True Positive Rate) measures the percentage of actual positive cases that the model correctly classifies as positive. It reflects the model's effectiveness in detecting true positives from all actual positive instances. A high TPR value signifies that the model is proficient at identifying positive cases accurately. It is calculated as TP / (TP + FN).

False Positive Rate (FPR):

FPR (False Positive Rate) quantifies the percentage of actual negative cases that are incorrectly identified as positive by the model. It reflects the model's tendency to misclassify negative instances as positive. A low FPR value indicates that the model is effective at minimizing false positives and accurately distinguishing negative cases.
It is calculated as FP / (TN + FP).

True Negative Rate (TNR) or Specificity:

TNR (True Negative Rate) assesses the percentage of actual negative cases that the model correctly classifies as negative. It shows how effectively the model identifies true negatives out of all actual negative instances. A high TNR value indicates that the model is effective at accurately recognizing negative cases. It is calculated as TN / (TN + FP).

False Negative Rate (FNR):

FNR (False Negative Rate) gauges the percentage of actual positive cases that the model wrongly classifies as negative. It reflects the model's tendency to overlook or misclassify positive instances as negative. A low FNR value indicates that the model is effective at detecting positive cases and minimizing missed detections. It is calculated as FN / (TP + FN).

These metrics, including TPR, FPR, TNR, and FNR, are essential for evaluating the performance of classification models. They offer a detailed view of how effectively a model differentiates between positive and negative cases in a dataset. By analyzing these rates, researchers and practitioners can gauge a model's performance, compare it with other models, and make necessary adjustments to enhance its accuracy. Understanding these metrics is vital for improving model performance and guiding optimization efforts, thereby supporting better decision-making in various applications.

E. Model Training

Machine learning, a branch of artificial intelligence, focuses on developing algorithms and models that enable

computers to learn from data and make predictions without explicit programming. Instead of being manually programmed, these systems learn from examples and experiences. They are trained on datasets with input data and corresponding output values, aiming to forecast outputs for new inputs. The training process involves iteratively refining model parameters to minimize the gap between predicted and actual results.

Machine learning encompasses various methods, including supervised, unsupervised, and reinforcement learning. In supervised learning, models are trained on labelled datasets that include input-output pairs, while unsupervised learning involves analysing unlabelled data to identify patterns and structures. For our study, we utilized supervised learning techniques with several classification models, including Support Vector Machines (SVM)[11], Random Forest Classifier (RFC)[9], Logistic Regression, XGBoost[10], and Gradient Boosting[12], to build and assess our models.

F. Random Forest classifiers (RFC)

Random forests are a popular supervised machine learning method used for tasks with labelled target variables. They can handle both regression problems, which involve predicting numeric values, and classification problems, which involve predicting categorical outcomes. This technique is highly versatile and applicable across a range of problem areas. Random forests work by combining the predictions of multiple individual models through an ensemble approach. Each model in this ensemble is a decision tree, and the collective output of these trees enhances the overall predictive performance. Additionally, random forests offer robust performance by reducing overfitting compared to single decision trees and providing measures of feature importance, which help in understanding the influence of different features on the predictions.[8][4]

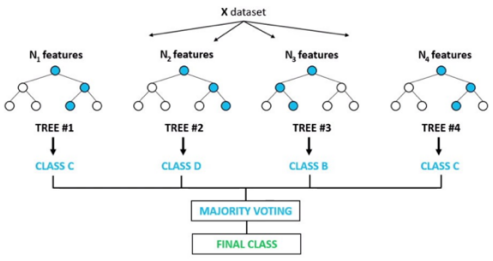

Fig.5. Random Forest classifiers Algorithm

The following steps are described to create a RFC model to detect a person in an office environment.

Step 1: Select random samples from a given data or training set.
Step 2: This algorithm will construct a decision tree for every training data.
Step 3: Voting will take place by averaging the decision tree.
Step 4: Finally, select the most voted prediction result as the final prediction result

G. Extreme Gradient Boosing (XG Boost)

XGBoost (Extreme Gradient Boosting**)** is a highly effective machine learning algorithm renowned for its superior performance and efficiency in classification and regression tasks. As an advanced implementation of gradient boosting, XGBoost improves predictive accuracy and computational speed through a range of features. It leverages boosting to iteratively refine predictions by combining multiple decision trees, each correcting the errors of its predecessors. The algorithm incorporates L1 and L2 regularization to prevent overfitting, supports parallel processing to expedite training, and handles missing values adeptly by learning the best imputation direction. XGBoost's scalability makes it suitable for large datasets, and its tree pruning techniques help manage model complexity[9]. Additionally, its built-in cross-validation aids in hyperparameter tuning and ensures robust model performance

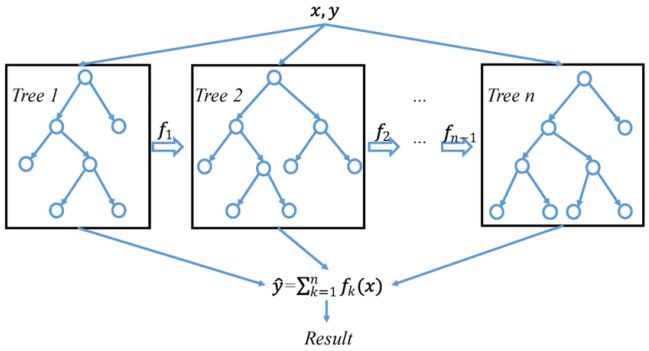

Fig.6. XG Boost

H. Support Vector Machine (SVM)

Support Vector Machine (SVM) stands out as a robust supervised learning algorithm utilized for classification and regression tasks. Its core principle involves identifying the optimal hyperplane within the feature space, effectively separating distinct classes while maximizing the margin between them.[13] SVM exhibits efficacy in high-dimensional spaces and finds applications across diverse domains, such as image recognition, text classification, and bioinformatics.[5]

SVM encompasses several variants:

1. **Linear SVM**: This variant employs a linear hyperplane to segregate classes, ideally suited for datasets exhibiting linear separability.

2. **Non-linear SVM (Kernel SVM):** Kernel SVM utilizes kernel functions to project input data into a higher-dimensional space, facilitating class separation through hyperplanes. Common kernel functions include Polynomial Kernel, Gaussian Radial Basis Function (RBF) Kernel, and Sigmoid Kernel.

3. **Multiclass SVM:** Despite being inherently binary, SVM can be extended to handle multi-class classification through techniques like One-vs-One and One-vs-All methods.

4. **Weighted SVM:** This variant assign varying weights to different classes, addressing issues related to class imbalance within the dataset

I. Gradient Boosting Classifier

Gradient Boosting Classifier is a robust ensemble learning method designed for classification tasks, leveraging the power of multiple weak learners, usually decision trees, to create a highly accurate predictive model. This technique operates by building models sequentially, where each new model is trained to correct the errors of its predecessor. By minimizing a specified loss function through gradient descent, the Gradient Boosting Classifier iteratively improves the model's performance and reduces bias[11]. The approach allows the model to capture complex relationships within the data, adapting to various patterns and interactions, which enhances its predictive accuracy and robustness.

The Gradient Boosting Classifier offers several notable advantages, including its ability to handle a wide range of data types and its resilience to overfitting through built-in regularization techniques[11]. Its flexibility and iterative nature make it particularly effective in tackling intricate classification problems across diverse fields The combination of these features makes the Gradient Boosting Classifier a valuable tool for practitioners seeking to develop sophisticated and effective predictive models.

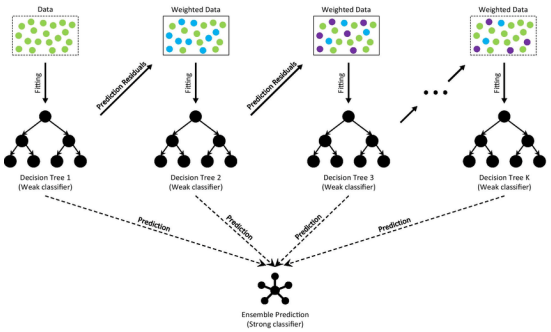

Fig.7. Gradient Boosting classifier

J. Logistic Regression

Logistic Regression is a key classification method used to predict binary outcomes by modelling the likelihood of a specific class based on one or more input features. Despite its name, logistic regression is a classification technique rather than a regression method. It employs the logistic function to convert predicted values into probabilities ranging from 0 to 1, enabling the algorithm to classify inputs by estimating parameters that optimize the likelihood of the observed outcomes. The simplicity and interpretability of logistic regression are major advantages, as it provides straightforward coefficients that illustrate how each predictor affects the probability of the target class.[12]

Beyond its straightforward application and clarity, logistic regression excels in various practical scenarios, such as in medical diagnosis, credit scoring, and binary classification tasks like spam detection. Its ability to efficiently handle linearly separable data makes it a strong choice for many applications. Furthermore, logistic regression can be adapted for multiclass classification through methods like One-vs-Rest or multinomial logistic regression. This adaptability, along with its capability to produce probabilistic predictions and its performance with large datasets, underscores its importance in both applied machine learning and theoretical statistical analysis.

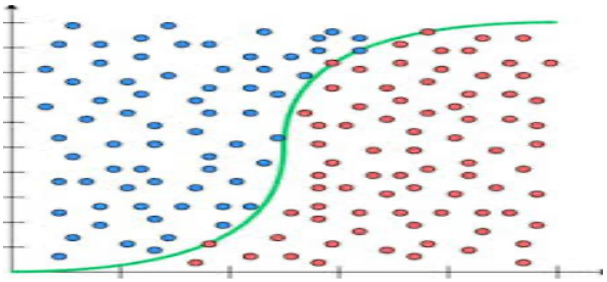

Fig.8. Logistic Regression

A. GUI using panel

The approach involved optimizing a classifier for person detection within an office environment, which required the development of machine learning models and their integration into an executable application featuring a graphical user interface (GUI).

To simplify the development of the GUI and ensure seamless integration with the Python-based data processing and machine learning frameworks, the Panel library was employed. Panel, an open-source Python library, streamlines the creation of interactive tools, dashboards, and complex applications. It provides both high-level reactive APIs and lower-level callback-based APIs, facilitating the rapid development of exploratory applications and supporting the creation of complex, multi-page applications with significant interactivity.

By leveraging Panel, it was possible to create a user-friendly interface that allowed real-time interaction with machine learning models, enhancing workflow efficiency and ensuring the system met the necessary interactivity requirements.

III. IMPLEMENTATION

A. Measurement Setup and Collection:

i. Measurement Environment:

For real-time measurements the experimental setup is conducted in Room 8-102a, which is arranged to mimic typical office settings with both occupied and unoccupied seats at Frankfurt University of Applied Sciences, where all data measurements take place.

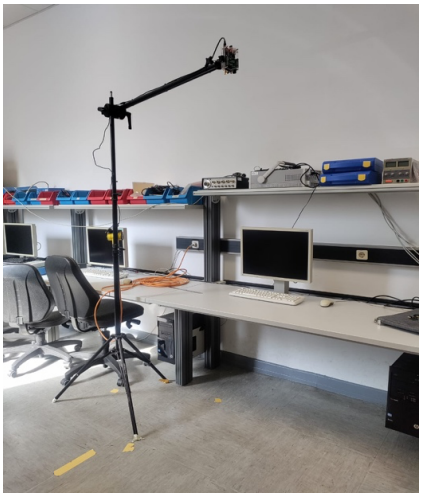

Fig.9. 8-102 a room used for measurement

The Red Pitaya Measurement board, along with the ultrasonic sensor, is installed in the room as shown in the above figure.

ii. Measurement Software:

The data gathered from the Red Pitaya measurement board is processed utilizing measurement software created at Frankfurt University of Applied Sciences. Figure 10 below depicts the Graphical User Interface (GUI) of this measurement software. The software has been set up to analyze either the analog data or raw data from the ultrasonic sensor.

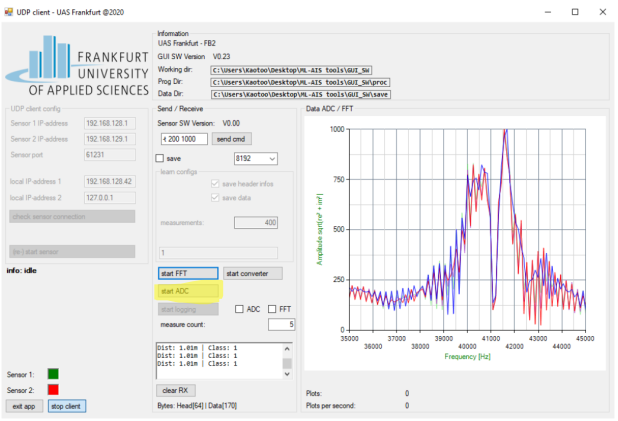

Fig.10. ADC data from the measurement software

Next the raw data was subjected to the ADC and exported as ADC data. The software is typically used to gather data and study how it behaves in various contexts. It can also ADC and time-series graphs.

iii. Data Format:

For this measurement, text-based ADC data that was exported from the software was used. Each ADC data has 13400 columns overall with each row shown in figure 11

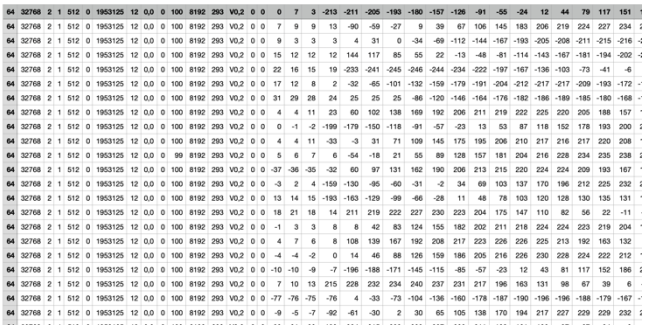

Fig.11. Raw measurement data of ADC.

Data headers are also exported by the program in addition to FFT and ADC data. The length of the data, the classification outcome from the software's current model, or the sampling frequency are just a few examples of the useful information included in data headers. Figure 11 shows a sample data format which was received from the software and

B. Data Collection

With the setup finalized, the ADC measurement data is extracted from the Red Pitaya measurement board with an ultrasonic sensor, detecting occupied seat or unoccupied seat. We establish specific conditions to observe differences across all scenarios. Our approach to implementing the model's response is iterative, based on ongoing analysis and feedback. We delve into detailed techniques and enhancements aimed at optimizing our model and refining the output throughout various phases. Data collection occurred in two phases:

- **Occupied Seat**: The scenarios is tested with the person wearing both long sleeves and half sleeves to understand how different types of clothing affect the ultrasonic sensor readings.

- **Unoccupied Seat**: The seat is completely empty with no person present.

In the baseline scenario for our study, for occupied seats, the scenarios include the person seated normally in an upright position, seated sideways with their body turned, leaning forward, leaning backward, entering the seat, and exiting the seat. Each of these scenarios will be tested with the person wearing both long sleeves and half sleeves to understand how different types of clothing might affect the ultrasonic sensor readings.

Throughout our experimentation, we meticulously documented various scenarios encountered person detection in an office environment.

The below figures show some of the scenarios.


Fig.12a. Person seated normally. *Fig.12b. Person Seated Sidewise*


Fig.12c.Person seated normally in low seat.Fig.12d. Person leaned forward

The experiment is also carried out with the Half Sleeves with different persons.

For unoccupied seats, scenarios include the seat being completely empty, partially placed within the sensor's range, set at a lower height, with an obstacle beside it or in place of the seat, with a laptop or book on it, with a bag, and with two empty seats in the sensor's range.

The below figures show some of the scenarios of unoccupied seats situations

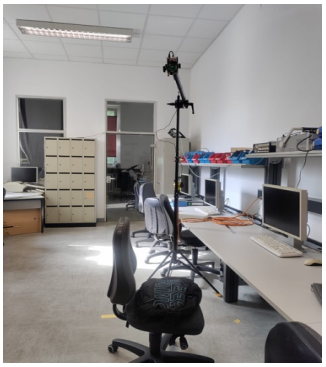


Fig.13a. Seat with a Bag. *Fig.13b. Two empty seats placed together*


Fig.13c. No seat under the sensor. *Fig.13d. Obstacle placed under sensor*

C. Random Forest Classifier implementation

Utilizing the Random Forest Classifier algorithm proves to be a strong approach for training models using data collected from the specified sensor setup. This algorithm employs ensemble learning, creating numerous decision trees during the training process. Each tree is trained on a random subset of the data, which helps prevent overfitting and enhances the model's ability to make accurate predictions on new data. By combining the predictions of these individual trees, the algorithm generates a final prediction that typically demonstrates improved accuracy and reliability compared to using a single decision tree.[4]

Additionally, Random Forest Classifier is adept at handling high-dimensional data, making it well-suited for the complex, multi-dimensional nature of sensor data. These attributes make Random Forest Classifier a compelling choice for training models designed to analyze data obtained from the Red-Pitaya measurement board equipped with an ultrasonic sensor.

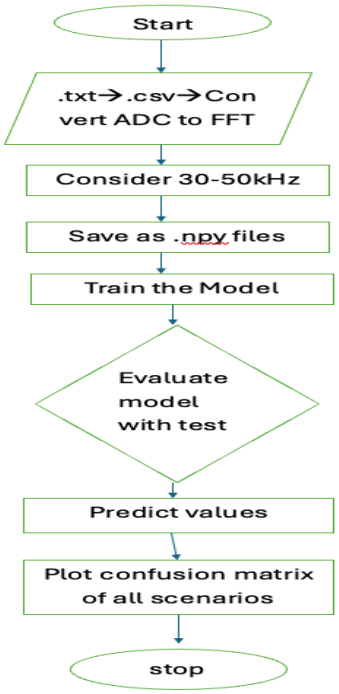

Fig.14. Flow Chart of model

i. Data Pre-processing:

When the data is first extracted from the Red-Pitaya measuring board, which includes an ultrasonic sensor, it is initially saved in the .txt format. This data is then converted into an .xlsx file for easier handling. After this conversion, we further process the data by converting the ADC (Analog-to-Digital Converter) data into FFT (Fast Fourier Transform) data. During this step, we focus specifically on the frequency range between **30-50 kHz**, which is relevant to our analysis. This frequency-filtered FFT data is then used for training machine learning models. The data is split into two parts: 80% of it is used to train the models, while the remaining 20% is

reserved for testing and prediction tasks. we manually labelled the data, assigning 'Class 1' for seats occupied by a person and 'Class 0' for unoccupied seats

ii. Building and Training Random Forest Classifier:

The Random Forest Classifier is part of Python's sklearn library and can be imported from *sklearn.ensemble*. Popular Python machine learning toolkit scikit-learn, often known as sklearn, offers effective tools for data mining and analysis, including a number of different algorithms for dimensionality reduction, clustering, regression, and classification. Building upon NumPy, SciPy, and matplotlib, its straightforward and effective design makes it very popular.

Numpy is used to load the numpy arrays (.npy) which were saved from the .xlsx data.

From *skelarn.metrics*, the functions *confusion_matrix* and *accuracy_score* are used to build the confusion matrix and calculate the accuracy of the training data

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,precision_score,recall_score,f1_score
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

Fig.15. Python Modules used for RFC model

In our experiments, we utilized a Random Forest Classifier with specific hyperparameters to enhance the model's accuracy and robustness. The classifier was configured with 3 estimators (n_estimators=3) to balance computational efficiency and model performance. We limited the maximum depth of each tree to 5 (max_depth=5) and set the minimum number of samples required to split an internal node to 5 (min_samples_split=5) to prevent overfitting. Additionally, the minimum number of samples at a leaf node was set to 5 (min_samples_leaf=5), ensuring that each leaf contains sufficient data to make reliable predictions.

We employed the square root of the number of features (max_features='sqrt') when considering features at each split, which is a common practice in Random Forests to reduce variance. The model was trained using bootstrap samples (bootstrap=True) and the entropy criterion (criterion='entropy') was chosen to measure the quality of splits, aiming to maximize information gain. All random processes were controlled with a fixed seed (random_state=42) to ensure reproducibility of the results.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf_classifier = RandomForestClassifier(
    n_estimators=3,  # Increase number of trees
    max_depth=5,      # Limit tree depth
    min_samples_split=5,  # Increase minimum samples to split
    min_samples_leaf=5,    # Increase minimum samples at leaf
    max_features='sqrt',   # Use square root of features
    bootstrap=True,        # Use bootstrap samples
    criterion='entropy',   # Use entropy criterion
    random_state=42
)
```

Fig.16. Parameters used for RFC model

Once the Model is trained with our training data, labels were predicted for the Testing data and calculate the accuracy of the true positive values.

```
# Predict the labels
y_pred = rf_classifier.predict(X_test)
y_train_pred = rf_classifier.predict(X_train)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Fig.17: Predicting the labels and calculating Accuracy.

iii. Plotting the Confusion Matrices

The python libraries matplotlib and seaborn are used to plotting the confusion matrices.

Matplotlib is a comprehensive library for creating static, interactive, and animated visualizations in Python. It offers a wide range of plotting functions and customization options, making it suitable for various data visualization tasks.

Seaborn is built on top of Matplotlib and provides a higher-level interface for creating attractive and informative statistical graphics. It simplifies the process of creating complex visualizations by offering built-in themes and functions for common statistical plots.

```
conf_matrix = confusion_matrix(y_test, y_pred)
# Visualize the confusion matrix
plt.figure(figsize=(10, 7))
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Person', 'Person'], yticklabels=['Not Person', 'Person'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Random Forest Classifier)')
plt.show()
```

Fig.18. Confusion matrix generation

The confusion matrix for both the testing data and the training data is plotted, although we are more interested in the confusion matrix of the testing data to observe the model's capability to predict the right labels.

Finally we save the classifier as a .pkl file using the joblib library.

```
joblib.dump(rf_classifier, '/Users/shivakumarbiru/Desktop/individual_project/rfc/models/rf_classifier')
```

Fig.19. saving the model

iv. Plotting the Confusion Matrix for Each Scenario

For scenario wise data which is present in the dataset, we load the saved classifier and use it to predict the values for data belonging to a particular scenario or test case. The scenario could be different persons or different seated position

```
model_rfc = joblib.load('/Users/shivakumarbiru/Desktop/individual_project/rfc/models/rf_classifier')
y_pred = model_rfc.predict(normalized_data)
precision = precision_score(test_labels_df, y_pred)
recall = recall_score(test_labels_df, y_pred)
f1 = f1_score(test_labels_df, y_pred)

print(f"Accuracy: {accuracy_score}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

Fig.20.The Confusion Matrix for Each Scenario

D. SVM Implementation

SVM operates by locating the hyperplane in the feature space that best divides various classes. In order to increase the model's capacity for generalisation, its goal is to maximise the margin between the classes. By using several kernel functions, including linear, polynomial, and radial basis function (RBF) kernels, SVM can handle both linear and non-linear classification tasks. SVM is frequently utilised because of its efficiency, adaptability, and capacity to handle high-dimensional data in a variety of areas.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Instantiate the SVM classifier
clf = make_pipeline(SimpleImputer(strategy='mean'), SVC(C=60.0))

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

Fig.21. Training the Model using SVM Classifier

Using scikit-learn's `*make_pipeline*` function, this code creates a pipeline of pre-processing steps that are followed by the SVM classifier, thereby instantiating an SVM (Support Vector Machine) classifier. In this pipeline:

- The mean value along each column is used by {SimpleImputer(strategy='mean')} to impute missing values in the dataset.
- {SVC(C=60.0)} sets the regularisation parameter {C} to 60.0 when the SVM classifier is first initialised. This parameter manages the trade-off between maintaining the model's tiny weights (coefficients) to prevent overfitting and obtaining a low training error.

The rest of the methodology remains the same as we used in the RandomForestClassifier Implementation.

E. Logistic Regression

We implemented a logistic regression model using a pipeline to standardize features and optimize performance. The data was split into training and testing sets with an 80-20 ratio for robust evaluation.

The model was tuned with a regularization strength of 65 (C=65) and L2 regularization (penalty='l2') to balance training error and overfitting. We used the saga solver (solver='saga'), suitable for large datasets, with a maximum of 10 iterations (max_iter=10) for timely convergence.

StandardScaler was applied to ensure that features had a mean of 0 and a standard deviation of 1. The model was trained on scaled data, and its effectiveness was evaluated using accuracy and classification reports.

```
model = LogisticRegression(solver='saga', C=0.001, multi_class='ovr', max_iter=2,penalty='l2',
                           random_state=42)
model.fit(X_train, y_train)
```

Fig.22. Training the Model using Logistic Regression Classifier

The rest of the methodology remains the same as we used in the RandomForestClassifier Implementation and testing.

F. XG Boost

we utilized the XGBoost classifier to optimize model performance on a given dataset. We employed hyperparameter tuning to identify the best parameters for the model, which included a subsample rate of 0.4, a regularization lambda of 2, and a regularization alpha of 0.1. The model was configured with 1000 estimators to facilitate early stopping, a minimum child weight of 2, a maximum depth of 11, a learning rate of 0.1, gamma set to 0, and a colsample_bytree rate of 0.8

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import joblib
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
best_params = {
    'subsample': 0.4,
    'reg_lambda': 2,
    'reg_alpha': 0.1,
    'n_estimators': 1000,  # Set a high number for early stopping
    'min_child_weight': 2,
    'max_depth': 11,
    'learning_rate': 0.1,
    'gamma': 0,
    'colsample_bytree': 0.8
}
xgb_classifier = xgb.XGBClassifier(**best_params, random_state=42)
xgb_classifier.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    verbose=True,
)
```

Fig.22. Training the Model using XG Boost Classifier

The XG Boost classifier was trained with the specified parameters and evaluated on the test set, allowing for monitoring of performance through early stopping. The implementation leveraged the XGBoost library for Python, and the model's effectiveness was gauged using accuracy and confusion matrix metrics. This approach aimed to balance model complexity and performance, ensuring robust evaluation on unseen data.

The rest of the methodology remains the same as we used in the RandomForestClassifier Implementation and testing.

G. Gradient Boosting Classifier

We applied the Gradient Boosting Classifier to model the dataset, focusing on optimizing its performance with specific hyperparameters. The Gradient Boosting Classifier was configured with 2 boosting stages, a learning rate of 0.3 to control the step size of each iteration, and a maximum tree depth of 100 to capture complex patterns. The model was initialized with a random seed of 42 to ensure reproducibility.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and training the model with parameters
gbm_model = GradientBoostingClassifier(
    n_estimators=2,     # Number of boosting stages to be run
    learning_rate=0.3,    # Step size for each iteration
    max_depth=100,          # Maximum depth of the individual trees
    random_state=42       # Seed for the random number generator
)
gbm_model.fit(X_train, y_train)
```

Fig.23. Training the Model using Gradient Boosting Classifier

After training the model on the training set, we evaluated its performance using accuracy, confusion matrix, and classification report metrics on the test set. This approach aimed to leverage the boosting technique to improve predictive accuracy and robustness in classification tasks.

The rest of the methodology remains the same as we used in the RandomForestClassifier Implementation and testing.

H. Graphical User Interface (GUI) Development and Integration

The Graphical User Interface (GUI) was developed using the Panel library, an open-source Python tool that enables the creation of interactive dashboards and complex applications. This GUI is designed to facilitate the optimization of classifiers for person detection within an office environment.

1) GUI Structure

a) **File upload** :
The left panel of the GUI provides a file input widget, allowing users to browse and upload datasets. This feature is essential for loading the data that will be used for classification.

b) **Classifier selection:**
Below the file upload section, a dropdown menu allows users to select a classifier. The classifiers available in this dropdown are integrated using machine learning libraries such as Scikit-learn. The selected classifier can then be trained and tested using the uploaded dataset.

c) **ExecutionControls**:
Once the user uploads a dataset and selects a classifier, the "Run" button initiates the training and evaluation process. The process is managed in the background, with the GUI providing real-time feedback.

d) **Compare accuracies**: The compare accuracies button is used to display the accuracies of the classifiers

e) **Output Display**:

i) Confusion Matrix: A confusion matrix is displayed to show the classifier's performance, highlighting both correctly and incorrectly classified instances.

ii) Metrics: Performance metrics such as accuracy, precision, recall, and F1 score are displayed to provide a detailed assessment of the classifier's effectiveness.

iii) Accuracy Comparison: This section enables users to compare the accuracies of different classifiers, aiding in the evaluation of the most effective model.

2) Integration with Machine Learning Models

The backend of the GUI is tightly integrated with machine learning models implemented in Python. The Panel library allows seamless binding of these models to the GUI components, enabling real-time interaction and feedback. When a user uploads a dataset and selects a classifier, the GUI triggers the appropriate machine learning pipeline, processes the data, trains the model, and updates the output displays accordingly.

This implementation provides a robust and interactive platform for users to experiment with different classifiers and optimize their models for person detection in an office environment.

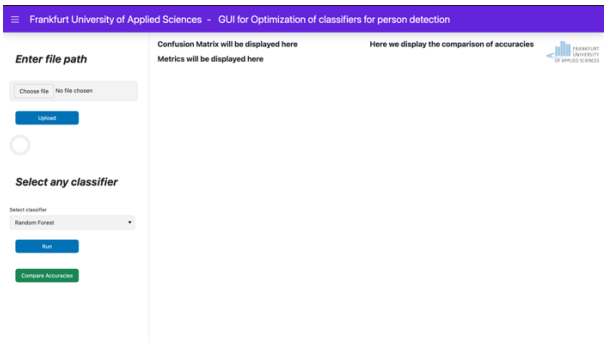

Fig.23. GUI for Optimization of classifier for person detection.

IV. RESULTS

The goal of this study was the optimization of classifiers for person detection in an office environment using data collected from redpitaya sensor. The performance of five different classifiers—Random Forest Classifier (RFC), Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting—was evaluated. The classifiers were tested using two datasets: the testing data derived from a 20% split of the collected data and an additional dataset created under varying scenarios to simulate real-world conditions. The evaluation was based on key performance metrics, including accuracy, recall, precision, specificity, and F1 score, derived from the confusion matrices for each classifier.

TABLE 1: Testing scenarios

Scenario	Description
Testing Data (20%)	20% of the overall data, encompassing all scenarios combined.
Seated Normally	The person is seated in a typical forward-facing position.
Seated Sideways	The person is seated sideways, posing a potentially more challenging detection.
Entering	The scenario where the person is entering the seat
Exiting	The scenario where the person is exiting the seat.

Label 0 - represents no person on the seat

Label 1- represents person on the seat

In the initial stage of our investigation, we focused on evaluating the accuracy and other performance metrics, such as the confusion matrix, precision, and recall, for each classifier under consideration. To achieve this, we employed a rigorous methodology where we divided our dataset into two distinct subsets. Specifically, 80% of the data was allocated for training purposes, allowing the classifiers to learn and adapt to the underlying patterns within the dataset. The remaining 20% was reserved as initial testing data to evaluate the classifiers' performance and ensure that they generalize well to unseen examples [12].

The classifiers subjected to this testing phase included Random Forest Classifier (RFC), Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting. By applying this approach, we aimed to gain a comprehensive understanding of each classifier's effectiveness and reliability. The performance metrics derived from the testing data, such as accuracy, confusion matrix, precision, and recall, provided valuable insights into how well each classifier performed in distinguishing between different classes and handling various types of errors. This systematic evaluation was crucial for identifying the most suitable classifier for our specific application and setting the stage for further refinements and optimizations.

Following this evaluation, we present the confusion matrix images and accuracy scores for each classifier. The confusion matrix images visually represent the performance of each classifier by illustrating the true positive, true negative, false positive, and false negative counts, providing a clear view of how each classifier handled the classification task. Alongside these images, we list the accuracy scores, which quantify the overall proportion of correctly classified instances for each model. These visual and numerical results offer a comprehensive snapshot of the classifiers performance, facilitating a straightforward comparison and aiding in the selection of the most effective model for our needs.

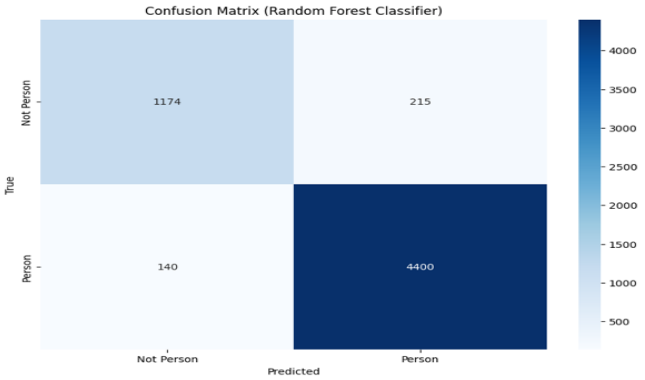

Fig 24. Confusion matrix of testing data of RFC

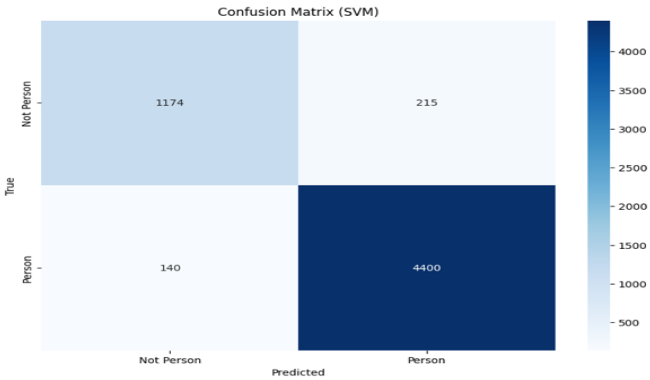

Fig 25. Confusion matrix of testing data of SVM

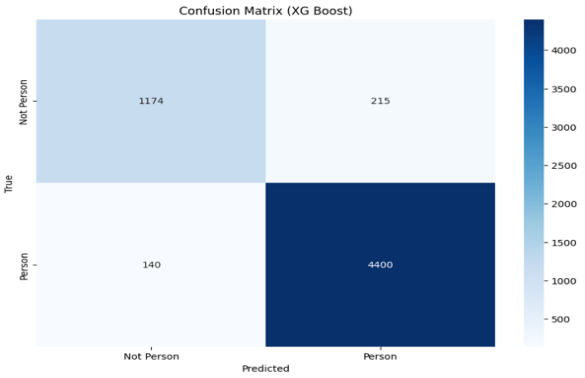

Fig 25. Confusion matrix of testing data of XG Boost

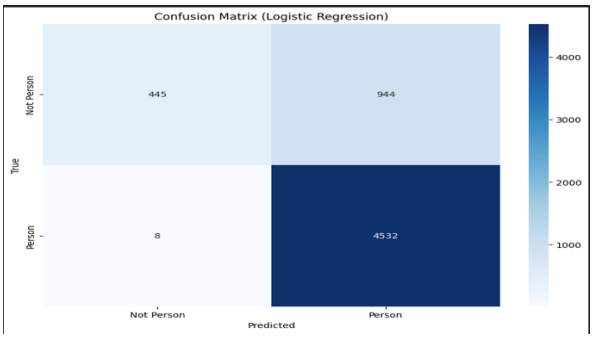

Fig 26. Confusion matrix of testing data of Logistic Regression

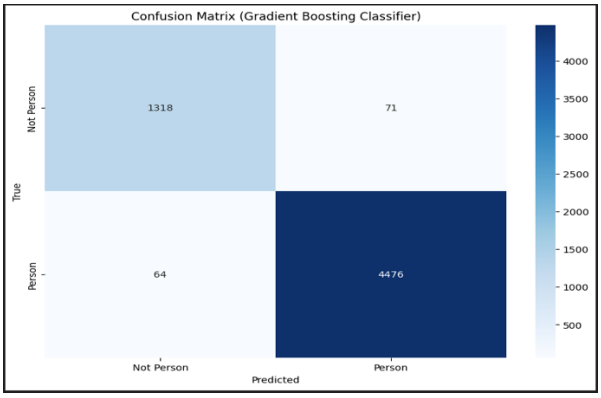

Fig 27. Confusion matrix of testing data of Gradient Boosting classifier

Person Seated Normally:

This scenario involves evaluating the classifiers' performance in detecting individuals seated in a typical forward-facing position, which serves as a baseline for standard seating conditions. This forward-facing posture represents a common and relatively straightforward setup, making it an ideal reference point for assessing each classifier's ability to handle basic seating arrangements. The confusion matrices for each classifier in this scenario provide detailed insights into their accuracy, sensitivity, and specificity. By examining these matrices, we can evaluate how well each classifier performs under these standard conditions, setting a benchmark before assessing their effectiveness in more complex scenarios[13].

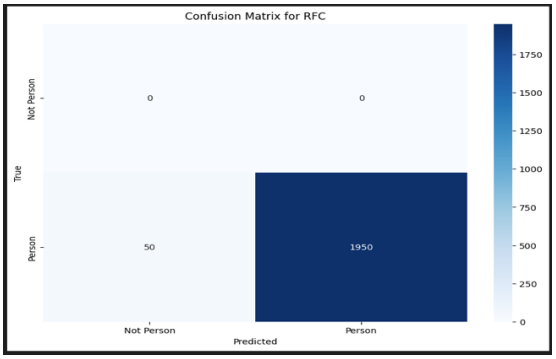

Fig 28. person seated normally of Gradient Boosting classifier

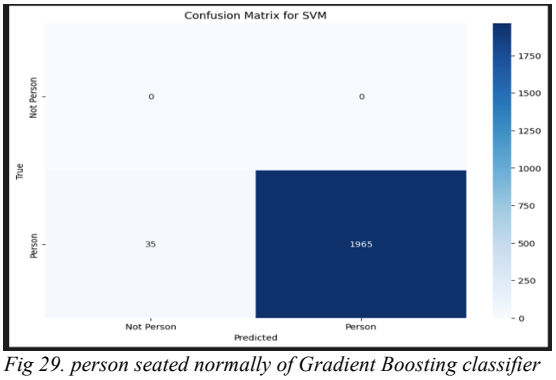

Fig 29. person seated normally of Gradient Boosting classifier

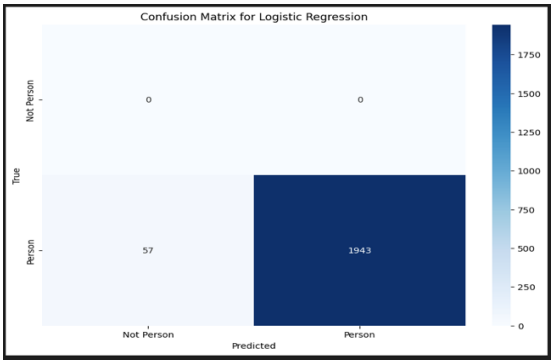

Fig 30 . person seated normally of Gradient Boosting classifier

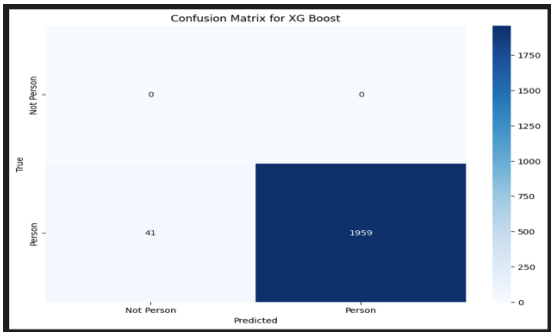

Fig 31. person seated normally of Gradient Boosting classifier

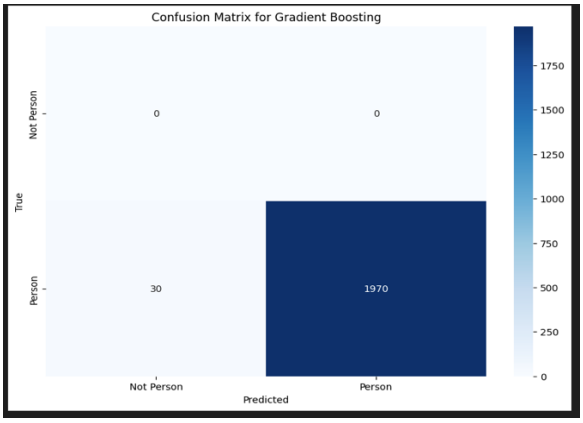

Fig 32. person seated normally of Gradient Boosting classifier

Person seated sidewise:

This scenario involves evaluating the classifiers' performance in detecting individuals seated in a side facing position, which serves as a one of the possibilities of a seating position conditions. The confusion matrices for each classifier in this scenario provide detailed insights into their accuracy, sensitivity, and specificity. By examining these matrices, we can evaluate how well each classifier performs under this condition.

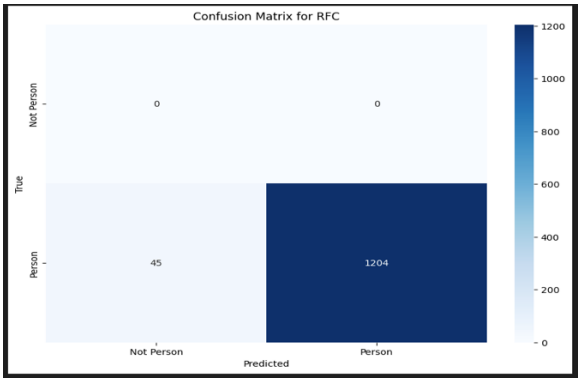

Fig 33. Person seated side facing of Gradient Boosting classifier

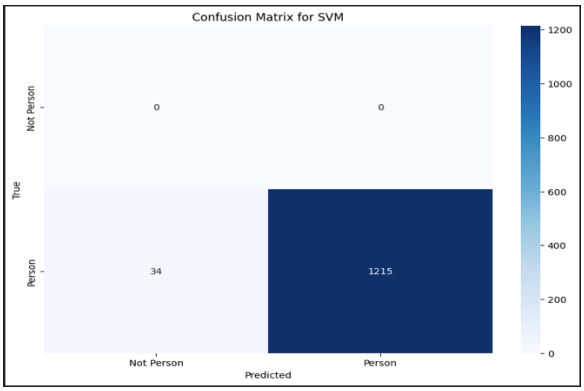

Fig 34. Person seated side facing of Gradient Boosting classifier

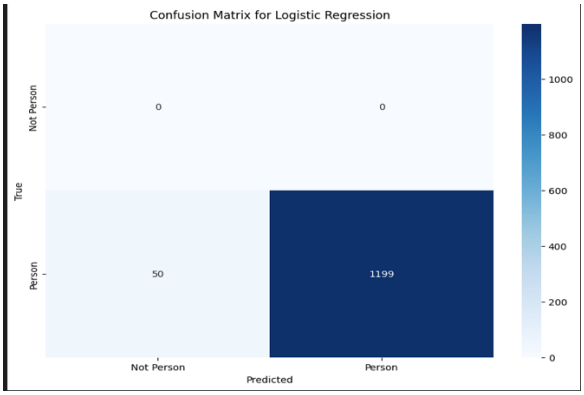

Fig 35. Person seated side facing of Gradient Boosting classifier

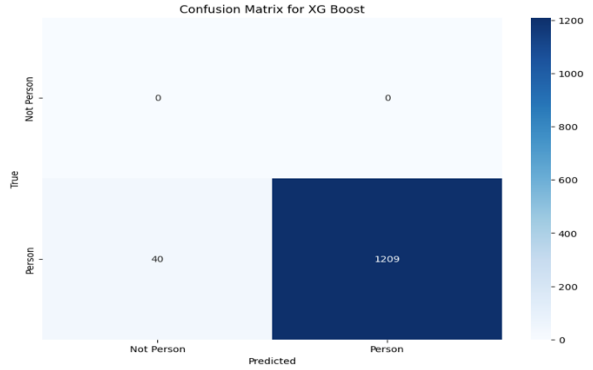

Fig 36. Person seated side facing of Gradient Boosting classifier

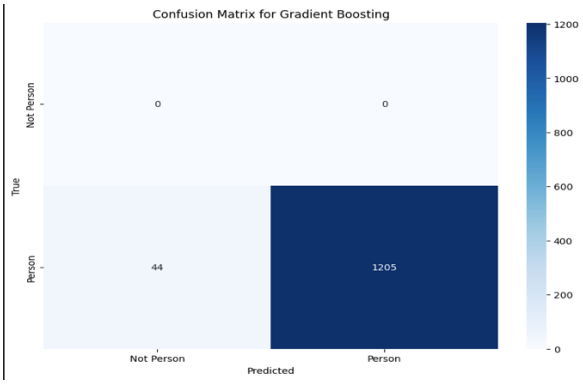

Fig 37. Person seated side facing of Gradient Boosting classifier

Scenario wise comparison results with the classifiers

Results were compared across five different classifiers: Random Forest Classifier (RFC), Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting.

Normal seated :

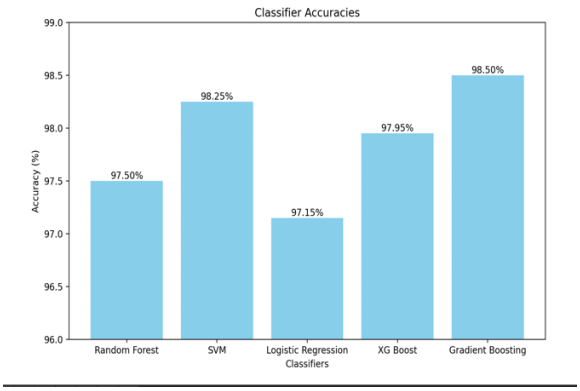

Figure 38: Accuracies of classifiers of person seated normally

The bar chart illustrates **Fig.31** the classification accuracies of five when tested on data where the subject is seated in a normal position. Among the classifiers, Gradient Boosting exhibited the highest accuracy at 98.5%, followed closely by SVM at 98.25%. XGBoost also performed well, achieving an accuracy of 97.95%. The Random Forest classifier reached 97.5%, while Logistic Regression recorded the lowest accuracy at 97.15%. These results indicate that boosting methods, particularly Gradient Boosting, are highly effective for this scenario, with SVM also providing competitive performance. In contrast, Logistic Regression was comparatively less effective in this context.

Side-Facing Seated Person:

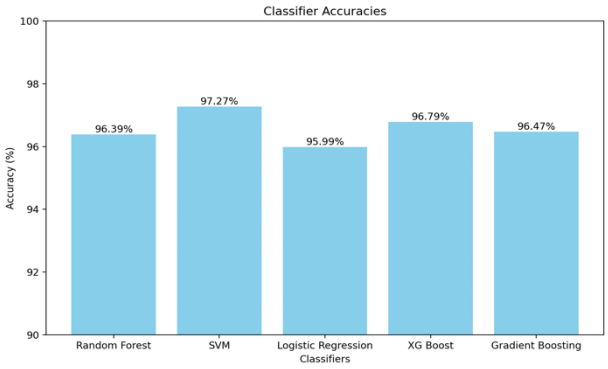

Figure 39: Accuracies of classifiers of a person seated side-facing

The above bar chart displays the classification accuracies of five different machine learning models—Random Forest, Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting—when applied to a specific dataset. In this scenario "Person seated side facing", the SVM model outperformed the others with an accuracy of 97.27%, demonstrating its effectiveness in the classification task. The XGBoost model followed with a close accuracy of 96.75%, indicating strong performance. Gradient Boosting achieved an accuracy of 96.42%, while Random Forest recorded a slightly lower accuracy of 96.19%. Logistic Regression, while still performing well, had the lowest accuracy among the classifiers at 95.95%. These results suggest that while all models performed effectively, SVM and XGBoost are particularly robust for this dataset,with Logistic Regression showing comparatively lower accuracy.

Exit Scenario:

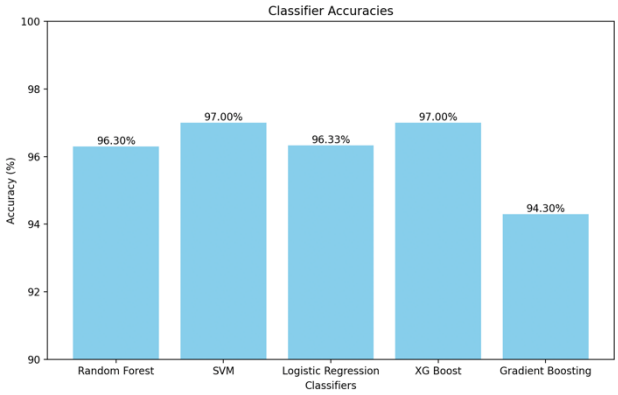

Figure.40: Accuracies of classifiers of a person exiting from the seat

The classification accuracies of the five machine learning models—Random Forest (RFC), Support Vector Machine (SVM), Logistic Regression (LR), XGBoost, and Gradient Boosting—were evaluated in the "Exit" scenario. SVM and XGBoost both achieved the highest accuracy, recording 97.0%, indicating their robustness in this scenario. Logistic Regression followed closely with an accuracy of 96.33%, demonstrating similar effectiveness to RFC, which recorded 96.3%. Gradient Boosting, however, showed the lowest accuracy at 94.3%, suggesting it may be less effective in scenarios involving an exit posture.

Leaned Back Scenario:

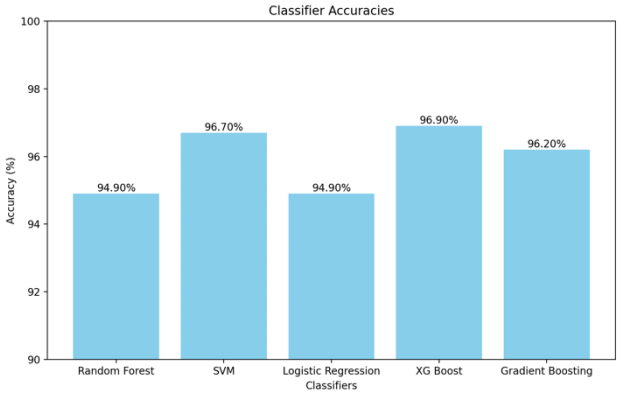

Figure 41: Accuracies of classifiers of a person leaned back on the seat

In the "Leaned Back" scenario, SVM and XGBoost again outperformed the other classifiers, with XGBoost slightly edging out SVM at 96.9% compared to 96.7%. Gradient Boosting followed closely with an accuracy of 96.2%, indicating a solid performance in this posture. Random Forest and Logistic Regression both recorded lower accuracies at 94.9%, suggesting that these models may struggle more with detecting persons in a leaned-back position compared to the more advanced boosting techniques.

Leaned Forward Scenario:

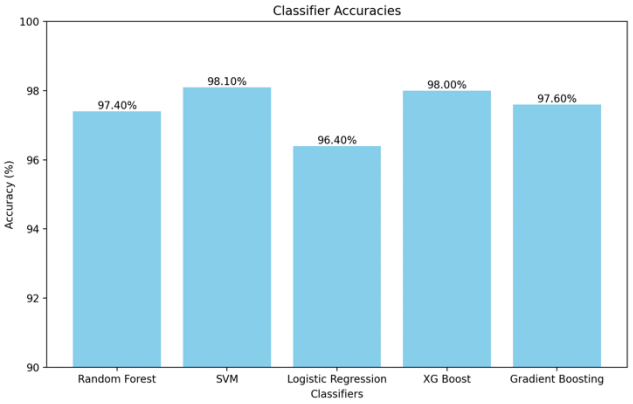

Figure 42: Accuracies of classifiers of a person leaned forward on the seat

The "Leaned Forward" scenario showed that SVM and XGBoost were the most effective classifiers, both achieving a high accuracy of 98.1%. Gradient Boosting also performed well in this scenario, with an accuracy of 97.6%, closely following the leading models. Random Forest showed a competitive performance with an accuracy of 97.4%, while Logistic Regression, despite performing well, recorded the lowest accuracy at 96.7%, indicating that it may be slightly less reliable in scenarios involving a forward-leaning posture.

GUI-Panel

The implementation of the graphical user interface (GUI) demonstrated robust functionality and performance

across multiple aspects. The file upload interface efficiently handled datasets,users reporting an intuitive and effective experience. The classifier selection feature, operated seamlessly, allowing users to select and apply different machine learning models without encountering issues. The execution controls, including the "Run" button, initiated the training and evaluation processes smoothly, providing real-time feedback as expected. The output display components, such as the confusion matrix and performance metrics, accurately represented the classifiers' performance. Clear visualizations and detailed metrics supported the interpretation of results, while the accuracy comparison section facilitated effective evaluation of different models, offering insights into their relative effectiveness. Successful integration with machine learning models ensured real-time interaction and efficient backend automation, resulting in a streamlined workflow from data upload to model evaluation. Overall, the system demonstrated high reliability and user satisfaction, proving to be a user-friendly and efficient tool for machine learning tasks.

V. CONCLUSION

This study evaluated the performance of five machine learning models—Random Forest (RFC), Support Vector Machine (SVM), Logistic Regression (LR), XGBoost, and Gradient Boosting—for person detection across various seated scenarios. SVM and XGBoost demonstrated superior performance, with SVM achieving accuracies of 98.1% and 97.0% in the leaned forward and exit positions, respectively. Gradient Boosting excelled in the normal seated scenario (98.5%) but showed reduced effectiveness in varied postures. Random Forest and Logistic Regression generally had lower accuracies of the scenarios mentioned

The graphical user interface (GUI) greatly enhanced the study by enabling seamless dataset management and efficient model application via Scikit-learn integration. Its real-time feedback and accurate performance display ensured a smooth evaluation process, reinforcing the importance of selecting the right classifier based on task-specific requirements.

.

VI. ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Andreas Pech for his unwavering support and exceptional guidance throughout the course of this project. His expertise, insightful feedback, were invaluable in shaping my research and overcoming the challenges encountered. Dr. Pech's commitment to excellence and his readiness to assist at every stage of the project greatly contributed to its successful completion. His encouragement and support were crucial in advancing the work and achieving the project's objectives. I am profoundly grateful for his dedicated involvement and the significant impact he made on this project.

VII. REFERENCES

[1] Richards, Mike, Pi Updates and Red Pitaya, Radio User. Bournemouth, UK: PW Publishing Ltd. 11, 2016.

[2] R. C. Luo, S. L. Lee, Y. C. Wen and C. H. Hsu, "Modular ROS Based Autonomous Mobile Industrial Robot System for Automated Intelligent Manufacturing Applications," 2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), [Online]. Available: doi: 10.1109/AIM43001.2020.9158800..

[3] "SRF02 Ultrasonic range finder," robot-electronics, [Online]. Available: https://www.robot-electronics.co.uk/htm/srf02tech.htm.

[4] J. Ali, Random Forests and Decision Trees, https://www.researchgate.net, 2012.

[5] H. Z. S. a. M. H. Hayder Hasan, A Comparison Between Support Vector Machine (SVM) and Convolutional Neural Network (CNN) Models For Hyperspectral Image Classification, 2019.

[6] K. Balani, S. Deshpande, R. Nair and V. Rane, "Human detection for autonomous vehicles," IEEE International Transportation Electrification Conference (ITEC), 2015. [Online]. Available: doi: 10.1109/ITEC-India.2015.7386891..

[8] G. Louppe, Understanding Random Forests: From Theory to Practice, https://arxiv.org/abs/1407.7502.

[9] T. Chen, XGBoost: A Scalable Tree Boosting System, https://arxiv.org/pdf/1603.02754v3.

[10] D. K. S. &. B. D. Pathak, Hyperspectral image classification using support vector machine: a spectral spatial feature based approach. Evol. Intel. 15, 1809–1823 (2022). https://doi.org/10.1007/s12065-021-00591-0.

[11] A. Z. Piotr Florek, "Benchmarking state-of-the-art gradient boosting algorithms for classification," [Online]. Available: https://ar5iv.labs.arxiv.org/html/2305.17094.

[12] “TensorFlow,” [Online]. Available: https://www.tensorflow.org/guide/core/logistic_regression_core.

[13] Y. S. L. C. W. L. Y. A. Zhang, “Support Vector Machine Classification Algorithm and Its Application.Information Computing and Applications. ICICA 2012. Communications in Computer and Information Science,” 2012. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-34041-3_27#citeas.