ARYA KUMARI

Vellore Institute of Technology, Vellore

Made under the guidance of Dr. R Charles Darvin,  
(Sc ‘F’, CASDIC, DRDO)

employing machine learning algorithms for Detection of False Alarms in Passive Ranging Radar System

**INDEX**

|  |  |
| --- | --- |
| Overview of Active Ranging and Passive Ranging | 2 |
| Methods used for classification of False alarms | 4 |
| Employment of Machine Learning algorithms | 5 |
| Classification using Recurrent Neural Networks (RNN) | 8 |
| Classification using Random Forests | 19 |
| Conclusion | 26 |

**Detection of False Alarms in Passive Ranging Radar System**

Active ranging and passive ranging are two methods used in radar systems to determine the distance to an object. Here’s a brief overview of each:

**ACTIVE RANGING**

Active ranging involves the radar system actively transmitting a signal and then listening for the reflection (echo) of that signal from the target. The time it takes for the signal to travel to the target and back is measured, and this time delay is used to calculate the distance to the target.

**Key Points:**

1. Transmission and Reflection : The radar emits a pulse of radio waves that travels to the target and reflects back.

2. Time Measurement : The round-trip time of the pulse is measured.

3. Distance Calculation :The distance is calculated using the speed of light and the measured time delay.

**Advantages:**

- Provides precise distance measurements.

- Can detect and track moving targets.

**Disadvantages:**

- Requires the radar system to emit a signal, which can be detected by other systems.

- Can be affected by environmental factors like weather and terrain.

**PASSIVE RANGING**

Passive ranging, on the other hand, involves the radar system receiving signals that are emitted or reflected by the target without actively transmitting any signal itself. This method relies on the target emitting some form of signal, such as radio, infrared, or another type of electromagnetic radiation.

**Key Points:**

1. Signal Reception : The radar system receives signals emitted or reflected by the target.

2. Signal Analysis : The characteristics of the received signals (such as frequency, phase, and time difference of arrival) are analyzed.

3. Distance Estimation : Distance is estimated based on the analysis of the received signals, often requiring sophisticated algorithms and multiple sensors.

**Advantages:**

- Covert operation, as the radar system does not emit any signals.

- Less affected by environmental factors.

**Disadvantages:**

- Requires the target to emit a signal or be illuminated by an external source.

- Typically less precise than active ranging.

In summary, active ranging involves the radar system transmitting and then receiving its own signals to measure distance, while passive ranging relies on detecting signals emitted or reflected by the target itself. Both methods have their own advantages and disadvantages, making them suitable for different applications.

**Classifying True and False Alarms**

Distinguishing true targets from false alarms is a critical part of passive ranging. Several methods can be used:

**1. Signal Analysis:**

* **Signal Characteristics:** Analyze the signal's characteristics (e.g., frequency, modulation, pulse width) to match known target profiles.
* **Signal Strength and Consistency:** True targets usually produce consistent signals over time and across multiple sensors, while false alarms may be inconsistent or appear only briefly.

**2. Cross-Correlation:**

* **Multiple Sensors:** Cross-correlate signals received from multiple sensors to confirm the presence and position of a target.
* **Consistency Check:** Check for consistency in the time differences and angles of arrival among sensors.

**3. Statistical Filtering:**

* **Thresholding:** Use statistical methods to set thresholds for signal detection. Signals below a certain threshold can be considered noise or false alarms.
* **Pattern Recognition:** Employ machine learning algorithms to recognize patterns in the signals that correspond to true targets versus false alarms.

**4. Sensor Fusion:**

* **Data Fusion:** Combine data from different types of sensors (e.g., radar, infrared, acoustic) to improve detection accuracy and reduce false alarms.
* **Redundancy and Validation:** Use redundant sensors to validate detections. If multiple sensors detect the same target in the same location, it is more likely to be a true target.

**Classification by employing Machine Learning Algorithms**

Employing machine learning algorithms for pattern recognition in passive ranging involves several key steps, from data collection and preprocessing to model training, validation, and deployment. Here’s a detailed explanation of how this process can be implemented:

**1. Data Collection**

**- Signal Data :** Collect raw signal data from the sensors, including various signal characteristics such as amplitude, frequency, phase, pulse width, and time of arrival.

**- Metadata :** Gather metadata related to the sensor's positions, environmental conditions, and any known characteristics of targets and false alarms.

**2. Data Preprocessing**

**- Signal Conditioning :** Clean the signal data to remove noise and irrelevant information. This might include filtering, normalization, and feature extraction.

**- Feature Engineering :** Extract relevant features from the raw signal data that are indicative of true targets and false alarms. Features could include statistical measures (mean, variance), signal shape descriptors, spectral features, etc.

**- Labeling :** Label the data based on known instances of true targets and false alarms. This step might require manual annotation or the use of automated labeling systems if historical data is available.

**3. Model Training**

**- Dataset Split :** Divide the data into training, validation, and test sets to ensure that the model can generalize well to unseen data.

**- Algorithm Selection :** Choose appropriate machine learning algorithms for the task. Common choices for pattern recognition include:

**- Supervised Learning :** Algorithms like Support Vector Machines (SVM), Random Forests, Gradient Boosting Machines (GBM), and Neural Networks.

**- Deep Learning :** Convolutional Neural Networks (CNNs) for image-like signal data or Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) for time-series data.

**- Training :** Train the chosen model on the training dataset. This involves feeding the input features and corresponding labels into the model and optimizing it to minimize the error in predictions.

**4. Model Validation**

**- Validation Set :** Evaluate the model on the validation set to tune hyperparameters and prevent overfitting.

**- Metrics :** Use metrics such as accuracy, precision, recall, F1 score, and Receiver Operating Characteristic (ROC) curves to assess model performance.

**- Cross-Validation :** Employ cross-validation techniques to ensure robustness and reliability of the model.

**5. Model Deployment**

**- Integration :** Integrate the trained model into the radar system for real-time or near-real-time signal processing.

**- Real-time Processing :** Ensure the model can handle real-time data processing requirements, with considerations for latency and computational efficiency.

**- Feedback Loop :** Implement a feedback loop where the model's predictions can be continuously monitored and corrected, improving the model over time with new data.

**6. Classification of True and False Alarms**

**- Decision Threshold :** Set an appropriate decision threshold for classifying signals as true targets or false alarms based on the model’s output probabilities.

**- Ensemble Methods :** Combine multiple models or use ensemble methods to improve classification accuracy and robustness.

**- Post-Processing :** Apply additional logic or rules to refine the model’s predictions. For example, spatial and temporal consistency checks can help validate true target detections.

**7. Continuous Learning and Improvement**

**- Active Learning :** Continuously collect new data, especially from real-world operations, to retrain and improve the model.

**- Anomaly Detection :** Implement systems to detect and analyze anomalies in the data, which could indicate new types of false alarms or evolving target characteristics.

**- User Feedback :** Incorporate user feedback to correct misclassifications and enhance the model’s performance.

Using machine learning for pattern recognition in passive ranging helps improve the accuracy and efficiency of radar systems by leveraging complex signal patterns that traditional methods might miss.

**Classification model using Recurrent Neural Networks (RNN)**

Training model

**Purpose:**

The script is designed to prepare, preprocess, and train a Recurrent Neural Network (RNN) model on a dataset consisting of 3D coordinates. The goal is to classify these coordinates as either real targets or false alarms.

**Steps:**

1. Data Preparation:

- Load data from multiple CSV files located in the `measurements` folder.

- Handle different delimiters (comma, tab, space) for reading the files.

- Convert data to numeric format and handle missing values by imputing the mean.

- Label the data points as real targets (1) or false alarms (0) based on specified row indices.

- Normalize the data using `StandardScaler` and reshape it for RNN input.

2. Model Definition:

- Define an RNN model using TensorFlow/Keras, consisting of LSTM layers, Batch Normalization, Dropout, and Dense layers.

- Use hyperparameter tuning with `RandomizedSearchCV` to find the best parameters for the model, including optimizer, initializer, dropout rate, number of neurons, and learning rate.

3. Model Training:

- Train the model using the best hyperparameters found during tuning.

- Save the trained model and the scaler to disk for later use.

**Key Functions and Libraries:**

**- Pandas:** For data manipulation and reading CSV files.

**- NumPy:** For numerical operations.

**- TensorFlow/Keras:** For building and training the RNN model.

**- Scikit-learn:** For preprocessing (StandardScaler), imputation (SimpleImputer), and hyperparameter tuning (RandomizedSearchCV).

File Outputs:

- `rnn\_model`: The trained model saved in TensorFlow SavedModel format.

- `scaler.pkl`: The scaler object used for data normalization, saved using `pickle`.

**Code**

import os

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.impute import SimpleImputer as SI

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import RandomizedSearchCV

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LSTM

from tensorflow.keras.optimizers import Adam, RMSprop

from tensorflow.keras.wrappers.scikit\_learn import KerasClassifier

import warnings

import pickle

warnings.filterwarnings('ignore')

# Step 1: Data Preparation

t\_data = []

t\_labels = []

folder = 'measurements'

for i in range(1, 101):

f = os.path.join(folder, f'T\_{i}.csv')

# Try reading the file with different delimiters

try:

df = pd.read\_csv(f, header=None, delimiter=',', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(f, header=None, delimiter='\t', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(f, header=None, delimiter=' ', names=[0, 1, 2])

except pd.errors.ParserError as e:

print(f"Error reading {f}: {e}")

continue

df = df.to\_numpy()

df = pd.to\_numeric(df.flatten(), errors='coerce').reshape(df.shape)

if not np.isfinite(df).all():

df[~np.isfinite(df)] = np.nan

imp = SI(strategy='mean')

df = imp.fit\_transform(df)

for j in range(df.shape[0]):

if (1 <= j+1 <= 101) or (104 <= j+1 <= 204) or (206 <= j+1 <= 308):

t\_labels.append(1)

t\_data.append(df[j])

elif (310 <= j+1 <= 411) or (413 <= j+1 <= 514) or (516 <= j+1 <= 618):

t\_labels.append(0)

t\_data.append(df[j])

t\_data = np.array(t\_data)

t\_labels = np.array(t\_labels)

# Normalize the data

scaler = StandardScaler()

t\_data = scaler.fit\_transform(t\_data)

# Reshape data for RNN: (samples, time\_steps, features)

t\_data = t\_data.reshape((t\_data.shape[0], 1, t\_data.shape[1]))

with open('scaler.pkl', 'wb') as f:

pickle.dump(scaler, f)

# Step 2: Define the RNN Model

def create\_model(optimizer='adam', init='he\_uniform', dropout\_rate=0.5, neurons=100, learning\_rate=0.001):

if optimizer == 'adam':

optimizer = Adam(learning\_rate=learning\_rate)

elif optimizer == 'rmsprop':

optimizer = RMSprop(learning\_rate=learning\_rate)

model = Sequential([

LSTM(neurons, activation='relu', kernel\_initializer=init, input\_shape=(1, t\_data.shape[2])),

BatchNormalization(),

Dropout(dropout\_rate),

Dense(neurons//2, activation='relu', kernel\_initializer=init),

BatchNormalization(),

Dropout(dropout\_rate),

Dense(1, activation='sigmoid', kernel\_initializer=init)

])

model.compile(optimizer=optimizer, loss='binary\_crossentropy', metrics=['accuracy'])

return model

model = KerasClassifier(build\_fn=create\_model, verbose=0)

# Hyperparameter tuning

param\_dist = {

'batch\_size': [16, 32, 64],

'epochs': [50, 100],

'optimizer': ['adam', 'rmsprop'],

'init': ['he\_uniform', 'glorot\_uniform'],

'dropout\_rate': [0.3, 0.5],

'neurons': [32, 64, 128],

'learning\_rate': [0.001, 0.01]

}

random\_search = RandomizedSearchCV(estimator=model, param\_distributions=param\_dist, n\_iter=10, cv=3, n\_jobs=-1)

random\_result = random\_search.fit(t\_data, t\_labels)

# Step 3: Train the best model

best\_model = random\_result.best\_estimator\_.model

model\_file = 'rnn\_model'

best\_model.save(model\_file, save\_format='tf')

print(f"Model trained and saved as {model\_file}")

Testing model

**Purpose:**

The script is designed to load a previously trained RNN model and use it to classify new data points from a test dataset. The script also visualizes the classification results.

**Steps:**

1. Data Preparation:

- Load test data from a specified CSV file.

- Handle different delimiters (comma, tab, space) for reading the file.

- Convert data to numeric format and handle missing values by imputing the mean.

- Normalize the test data using the previously saved scaler.

2. Model Loading:

- Load the trained RNN model from disk.

- Load the scaler object used for data normalization.

3. Data Classification:

- Use the loaded model to classify the normalized test data.

- Convert the model's probabilistic outputs to binary classifications (real targets or false alarms).

4. Visualization:

- Plot the classified data points in a 3D scatter plot to visually distinguish real targets from false alarms.

**Key Functions and Libraries:**

**- Pandas:** For data manipulation and reading CSV files.

**- NumPy:** For numerical operations.

**- TensorFlow/Keras:** For loading the trained RNN model.

**- Scikit-learn:** For preprocessing (StandardScaler) and imputation (SimpleImputer).

**- Matplotlib:** For data visualization in 3D scatter plots.

**- Pickle:** For loading the saved scaler object.

File Inputs:

- `rnn\_model`: The trained model loaded from TensorFlow SavedModel format.

- `scaler.pkl`: The scaler object loaded using `pickle`.

**Code**

import os

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.impute import SimpleImputer as SI

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import pickle

import warnings

warnings.filterwarnings('ignore')

# Prepare Data

test\_file = 'T\_47copy.csv'

ground\_truth\_file = 'GT\_47.csv' # Add the ground truth file

# Try reading the test file with different delimiters

try:

df = pd.read\_csv(test\_file, header=None, delimiter=',', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(test\_file, header=None, delimiter='\t', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(test\_file, header=None, delimiter=' ', names=[0, 1, 2])

except pd.errors.ParserError as e:

print(f"Error reading {test\_file}: {e}")

raise

df = df.to\_numpy()

df = pd.to\_numeric(df.flatten(), errors='coerce').reshape(df.shape)

if not np.isfinite(df).all():

df[~np.isfinite(df)] = np.nan

imp = SI(strategy='mean')

df = imp.fit\_transform(df)

with open('scaler.pkl', 'rb') as f:

scaler = pickle.load(f)

# Normalize test data

df = scaler.transform(df)

df = df.reshape((df.shape[0], 1, df.shape[1]))

model\_file = 'rnn\_model'

model = tf.keras.models.load\_model(model\_file)

# Classify test data

y\_pred = model.predict(df)

y\_pred = (y\_pred > 0.5).astype(int).flatten()

# Read the ground truth file

try:

gt\_df = pd.read\_csv(ground\_truth\_file, header=None, delimiter=',', names=[0, 1, 2])

except pd.errors.ParserError:

try:

gt\_df = pd.read\_csv(ground\_truth\_file, header=None, delimiter='\t', names=[0, 1, 2])

except pd.errors.ParserError:

try:

gt\_df = pd.read\_csv(ground\_truth\_file, header=None, delimiter=' ', names=[0, 1, 2])

except pd.errors.ParserError as e:

print(f"Error reading {ground\_truth\_file}: {e}")

raise

gt\_df = gt\_df.to\_numpy()

gt\_df = pd.to\_numeric(gt\_df.flatten(), errors='coerce').reshape(gt\_df.shape)

if not np.isfinite(gt\_df).all():

gt\_df[~np.isfinite(gt\_df)] = np.nan

gt\_df = imp.fit\_transform(gt\_df)

# Normalize ground truth

gt\_df = scaler.transform(gt\_df)

# Visualize predictions

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

real\_t = df[y\_pred == 1]

false\_a = df[y\_pred == 0]

ax.scatter(false\_a[:, 0, 0], false\_a[:, 0, 1], false\_a[:, 0, 2], c='red', label='False Alarms', alpha=0.7, zorder=1)

ax.scatter(gt\_df[:, 0], gt\_df[:, 1], gt\_df[:, 2], c='blue', label='Ground Truth', alpha=0.2, zorder=2)

ax.scatter(real\_t[:, 0, 0], real\_t[:, 0, 1], real\_t[:, 0, 2], c='green', label='Real Targets', alpha=1, zorder=3)

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

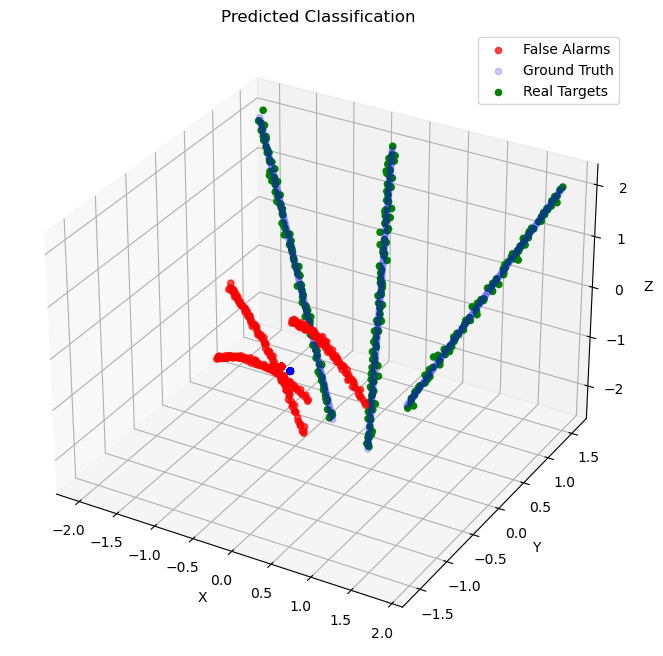
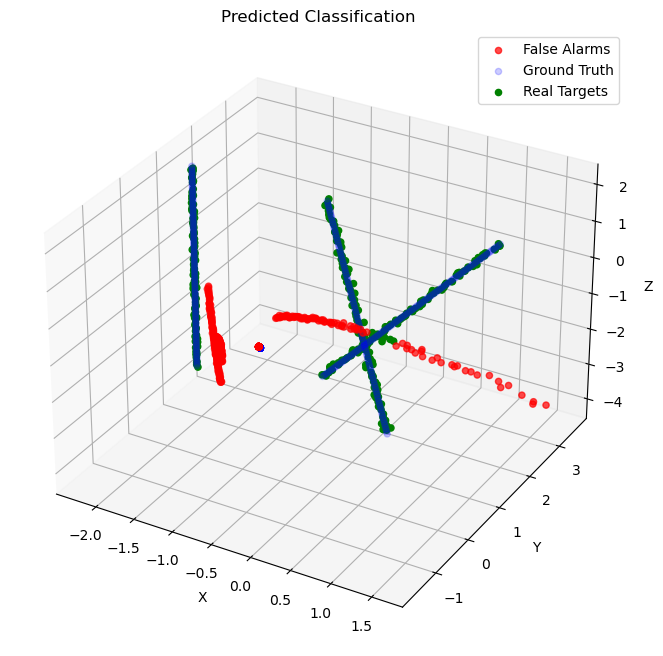
ax.set\_zlabel('Z')

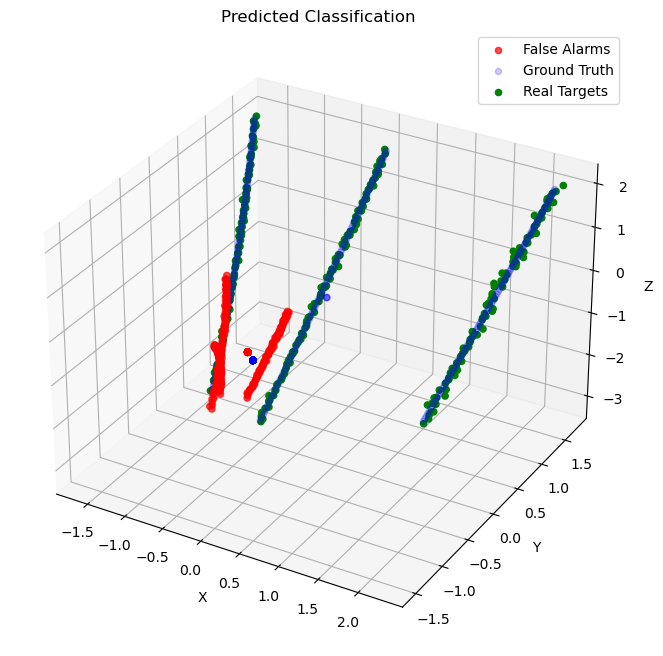
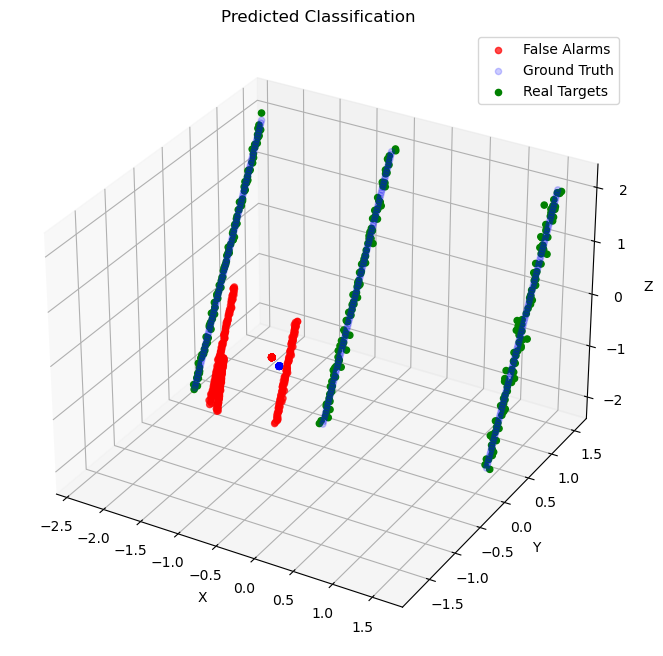
ax.set\_title('Predicted Classification)

ax.legend()

plt.show()

**Plotted Results**

**Classification model using Random Forest Classification**

Training model

1. Data Preparation:

* **Importing Libraries**: Essential libraries like numpy, pandas, joblib, and RandomForestClassifier from sklearn are imported.
* **Reading Data**: Training data is collected from 100 CSV files in the measurements folder. Each file is read with potential delimiters (, , \t, ) to accommodate different formats.
* **Data Cleaning**: Data is converted to numeric format, and missing values are imputed using the mean strategy.
* **Labeling**: Data is labeled as 1 (real target) or 0 (false alarm) based on the index ranges specified in the dataset.
* **Storing Data**: All cleaned and labeled data is stored in arrays t\_data and t\_labels.

2. Model Training:

* **Initialization**: A RandomForestClassifier with 100 estimators and a fixed random state is initialized.
* **Fitting**: The model is trained on the prepared data (t\_data, t\_labels).
* **Model Saving**: The trained model is saved to a file named rf\_model.joblib.

**Code**

import os

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier as RFC

from sklearn.impute import SimpleImputer as SI

import joblib

# Data Preparation

t\_data = []

t\_labels = []

folder = 'measurements'

for i in range(1, 100):

f = os.path.join(folder, f'T\_{i}.csv')

# Try reading the file with different delimiters

try:

df = pd.read\_csv(f, header=None, delimiter=',', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(f, header=None, delimiter='\t', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(f, header=None, delimiter=' ', names=[0, 1, 2])

except pd.errors.ParserError as e:

print(f"Error reading {f}: {e}")

continue

# if df.shape[1] != 3:

# print(f"Unexpected columns in {f}. Skipping file.")

# continue

df = df.to\_numpy()

df = pd.to\_numeric(df.flatten(), errors='coerce').reshape(df.shape)

if not np.isfinite(df).all():

df[~np.isfinite(df)] = np.nan

imp = SI(strategy='mean')

df = imp.fit\_transform(df)

for j in range(df.shape[0]):

if (1 <= j+1 <= 101) or (104 <= j+1 <= 204) or (206 <= j+1 <= 308):

t\_labels.append(1)

t\_data.append(df[j])

elif (310 <= j+1 <= 411) or (413 <= j+1 <= 514) or (516 <= j+1 <= 618):

t\_labels.append(0)

t\_data.append(df[j])

t\_data = np.array(t\_data)

t\_labels = np.array(t\_labels)

# Model Training

clf = RFC(n\_estimators=100, random\_state=42)

clf.fit(t\_data, t\_labels)

model\_file = 'rf\_model.joblib'

joblib.dump(clf, model\_file)

print(f"Model trained and saved as {model\_file}")

Testing model

1. Loading the Model:

* **Importing Libraries**: Essential libraries like numpy, pandas, joblib, matplotlib, and SimpleImputer are imported.
* **Model Loading**: The trained RandomForest model is loaded from the rf\_model.joblib file.

2. Data Preparation:

* **Reading Data**: Test data is read from the T\_83copy.csv file using different delimiters to accommodate different formats.
* **Data Cleaning**: Data is converted to numeric format, and missing values are imputed using the mean strategy.

3. Data Classification:

* **Prediction**: The model classifies the test data into real targets and false alarms.

4. Visualization:

* **3D Plotting**: Real targets are plotted in green, and false alarms are plotted in red.

**Code**

import numpy as np

import pandas as pd

import joblib

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.impute import SimpleImputer as SI

# Load the trained model

model\_file = 'rf\_model.joblib'

clf = joblib.load(model\_file)

# Prepare and Classify Test Data

test\_file = 'T\_83copy.csv'

ground\_truth\_file = 'GT\_83.csv'

# Try reading the test file with different delimiters

try:

df = pd.read\_csv(test\_file, header=None, delimiter=',', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(test\_file, header=None, delimiter='\t', names=[0, 1, 2])

except pd.errors.ParserError:

try:

df = pd.read\_csv(test\_file, header=None, delimiter=' ', names=[0, 1, 2])

except pd.errors.ParserError as e:

print(f"Error reading {test\_file}: {e}")

raise

df = df.to\_numpy()

df = pd.to\_numeric(df.flatten(), errors='coerce').reshape(df.shape)

if not np.isfinite(df).all():

df[~np.isfinite(df)] = np.nan

imp = SI(strategy='mean')

df = imp.fit\_transform(df)

# Classify test data

y\_pred = clf.predict(df)

# Read the ground truth file

try:

gt\_df = pd.read\_csv(ground\_truth\_file, header=None, delimiter=',', names=[0, 1, 2])

except pd.errors.ParserError:

try:

gt\_df = pd.read\_csv(ground\_truth\_file, header=None, delimiter='\t', names=[0, 1, 2])

except pd.errors.ParserError:

try:

gt\_df = pd.read\_csv(ground\_truth\_file, header=None, delimiter=' ', names=[0, 1, 2])

except pd.errors.ParserError as e:

print(f"Error reading {ground\_truth\_file}: {e}")

raise

gt\_df = gt\_df.to\_numpy()

gt\_df = pd.to\_numeric(gt\_df.flatten(), errors='coerce').reshape(gt\_df.shape)

if not np.isfinite(gt\_df).all():

gt\_df[~np.isfinite(gt\_df)] = np.nan

gt\_df = imp.fit\_transform(gt\_df)

# Plotting predictions

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

real\_t = df[y\_pred == 1]

false\_a = df[y\_pred == 0]

ax.scatter(gt\_df[:, 0], gt\_df[:, 1], gt\_df[:, 2], c='blue', label='Ground Truth', alpha=0.7, zorder=1)

ax.scatter(false\_a[:, 0], false\_a[:, 1], false\_a[:, 2], c='red', label='False Alarms', alpha=0.7, zorder=2)

ax.scatter(real\_t[:, 0], real\_t[:, 1], real\_t[:, 2], c='green', label='Real Targets', alpha=0.5, zorder=3)

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

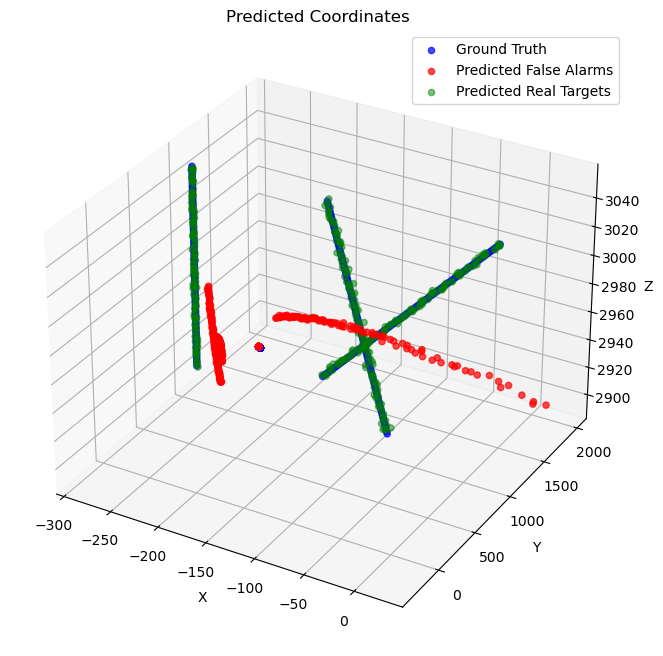
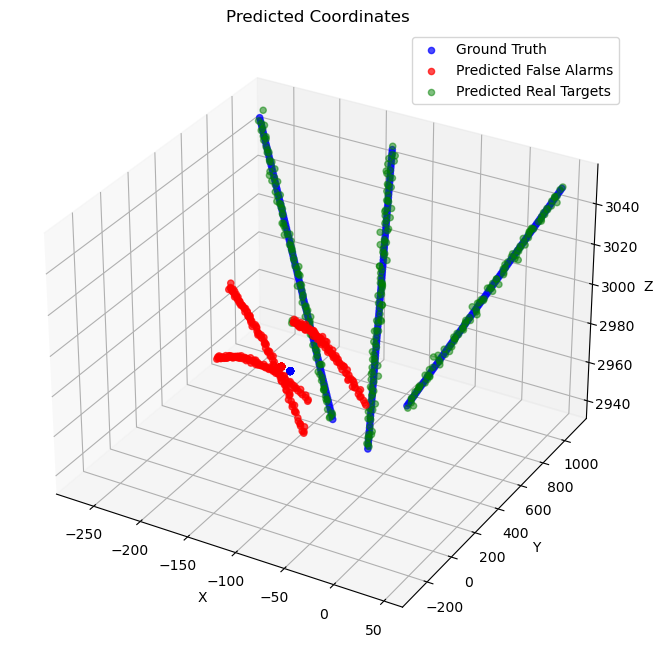
ax.set\_zlabel('Z')

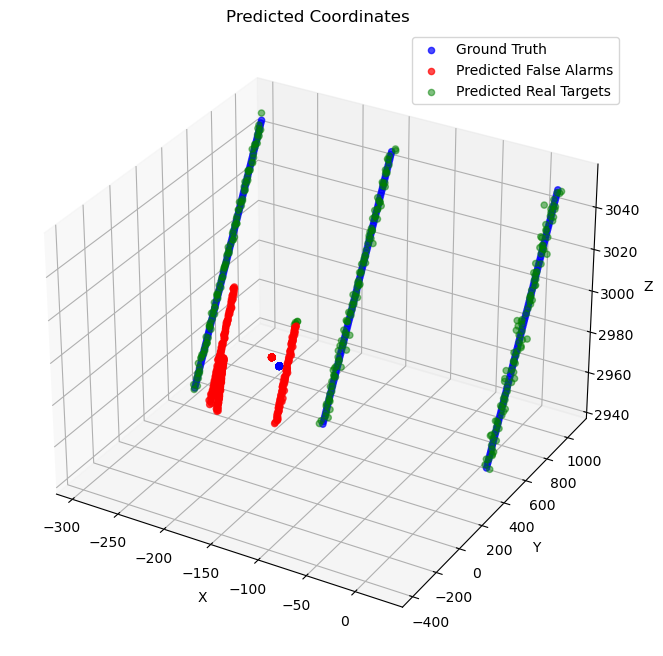
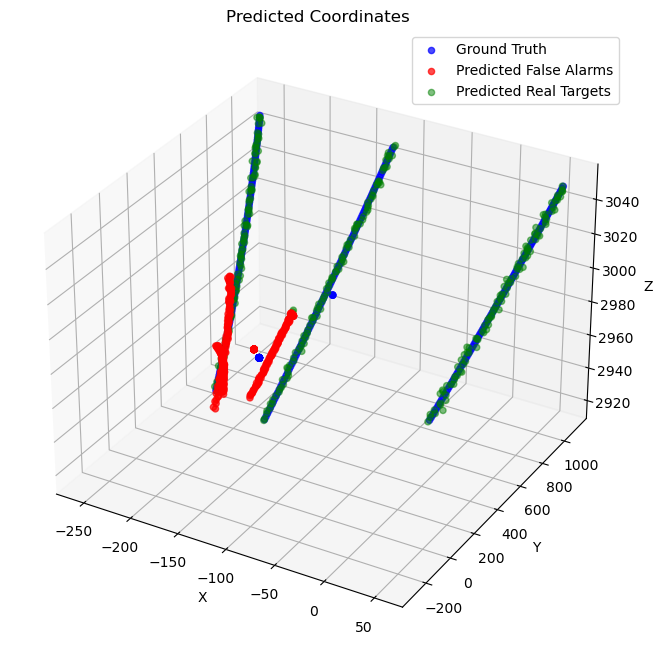
ax.set\_title('Predicted Coordiantes')

ax.legend()

plt.show()

**Plotted Results**

**Conclusion**

The Random Forest model, an ensemble learning method, trains multiple decision trees and aggregates their results, making it robust to overfitting and less sensitive to outliers. It handles high-dimensional data effectively and is computationally efficient during training, but it may struggle with capturing temporal dependencies in sequential data.

The RNN model, specifically using LSTM layers, is designed to handle sequential data by learning temporal dependencies, making it ideal for time-series analysis. However, RNNs are computationally intensive and require significant time and resources for training due to backpropagation through time. In terms of accuracy, the RNN model generally performs better on sequential tasks, capturing complex patterns in the data, while the Random Forest model provides strong performance on static, high-dimensional data with faster training times. The choice between these models depends on the specific nature of the data and the requirements of the task, with the Random Forest being preferable for non-sequential, high-dimensional data, and the RNN being superior for tasks involving sequential data and temporal dependencies.