Model Trainer & Predictor - Developer Documentation

This document describes the classification code responsible for training a machine learning model (using RandomForest), performing predictions, and handling real-time sensor data via serial communication. It covers the code structure, key functions, and the design decisions behind the GUI application built using Tkinter.

Overview

The application serves three main purposes:

1. **Training the Model:**
   - Users can select either raw sensor data or a pre-processed features file.
   - If raw data is chosen, features are extracted using sliding time windows (with configurable window length and stride) via a custom `DataProcessor` class.
   - The RandomForest classifier is then trained using these features, and the trained model (along with a label encoder) is saved for future use.
   - Training progress is displayed with a progress bar, and basic training metrics (e.g., window counts per label) are saved.
2. **Making Predictions:**
   - Similar to training, predictions can be performed on raw data or on an already processed features file.
   - If only raw data is available, the code processes the raw file, saves the extracted features, and then runs the prediction.
   - The predictions are saved to the processed file, and prediction metrics (accuracy, classification report, confusion matrix) are computed when ground truth is available.
   - Metrics can be viewed in a popup window.
3. **Real-time Predictions:**
   - The app connects to a sensor device over a serial port.
   - Sensor data is continuously read and buffered in batches.
   - The buffered data is processed (using the same sliding-window feature extraction method) to compute features.
   - The trained model then predicts the sensor’s state, and the current prediction is displayed on the GUI.
   - Serial ports are refreshed dynamically, and start/stop controls are provided for real-time prediction mode.

Dependencies

The code uses several standard and third-party libraries:

- **Tkinter:**
  For building the graphical user interface (dialogs, buttons, text widgets, etc.).
- **Pandas & NumPy:**
  For handling and processing sensor data in CSV files.
- **Scikit-learn:**
  For the RandomForest classifier, label encoding, and model evaluation metrics.
- **Joblib:**
  For saving and loading the trained model along with the label encoder.
- **Serial & Serial Tools:**
  For serial communication with external sensor devices.
- **JSON:**
  For saving and loading model metrics.
- **Threading & Time:**
  For background processing (such as feature extraction, training, and real-time predictions) so the GUI remains responsive.
- **Custom Module:**
  The `DataProcessor` class is imported to handle data extraction and feature computation.

Code Structure

The code is organized within the `ModelTrainerGUI` class, which creates a multi-section GUI comprising four major parts:

1. Train Model Frame

- **Purpose:**
  Enables users to train the model by selecting a raw data file or a pre-processed features file.
- **Key Components:**
  - **File Browsing:**
    Widgets to select a raw data file for training and an optional processed features file.
  - **Window & Stride Parameters:**
    Input fields to set the sliding window length and stride for feature extraction.
  - **Action Buttons:**
    - *Extract Features*: Spawns a background thread to process raw data and extract features.
    - *Train Model*: Initiates model training (also runs in a background thread).
  - **Progress Bar:**
    Provides visual feedback on the progress of feature extraction and training.

2. Model Prediction Frame

- **Purpose:**
  Allows users to make predictions using the trained model.

- **Key Components:**

  - **Input File Selection:**
    Widgets to choose a prediction input file (raw data) or an already processed features file.
  - **Prediction Options:**
    Input field for setting the prediction stride length.
  - **Action Buttons:**
    - *Extract Features*: For processing raw prediction data if necessary.
    - *Predict*: Runs prediction in a background thread.
    - *Show Metrics*: Opens a popup to display prediction metrics (accuracy, classification report, confusion matrix).
  - **Progress Bar:**
    Displays progress during prediction processing.

3. Real-time Predictions Frame

- **Purpose:**
  Enables real-time prediction by connecting to a sensor device over a serial port.

- **Key Components:**

  - **Serial Port Controls:**
    - Dropdown menu to select available serial ports.
    - Refresh, Connect, and Disconnect buttons.
  - **Batch & Timing Controls:**
    Input for setting the batch length (the time window for collecting serial data).
  - **Real-time Action Buttons:**
    - *Start Predictions*: Initiates real-time prediction mode.
    - *Stop Predictions*: Ends the real-time session.
  - **Display Areas:**
    - A scrolling text widget to display incoming sensor data.
    - A label showing the current prediction.
    - An indicator for seconds remaining in the current batch.

4. Terminal Updates Frame

- **Purpose:**
  Shows status messages and logs any updates from the application (e.g., errors, connection status).

Key Functions and Methods

Training and Feature Extraction

- **`extract_features()` and `run_extraction_bg()`:**
  These methods handle the background processing of raw training data to extract features using the `DataProcessor`. They update the progress bar and status messages as the data is processed.

- **`train_model()` and `run_training_bg()`:**
  Responsible for training the RandomForest model on the processed data. The model, along with a label encoder, is saved using joblib. Basic metrics (e.g., window counts per label) are computed and stored in a JSON file.

- **Progress Callbacks:**
  Functions like `train_window_progress_callback()` update the progress bar based on the number of processed windows during feature extraction.

Prediction

- **`predict_data()` and `run_prediction_bg_processed()`:**
  Similar in structure to the training functions, these methods handle predictions. They support both the use of raw and processed files, extract features if needed, and run predictions in a background thread.

- **Metrics Computation:**
  After predictions, accuracy, classification report, and confusion matrix are computed (if ground truth labels are available) and saved to a JSON file.

- **`display_metrics()`:**
  Opens a popup window displaying the prediction metrics.

Real-time Predictions

- **Serial Communication Setup:**
  Methods like `rt_connect_serial()` and `rt_disconnect_serial()` handle the connection to the sensor device.

- **Real-time Data Reading:**
  The `rt_read_serial_data()` method continuously reads data from the serial port in a background thread. It buffers the data for a set batch duration, processes the batch using the `DataProcessor`, computes features, and then uses the trained model to predict the current sensor state.

- **User Interface Updates:**
  The real-time section continuously updates the GUI with incoming data, the seconds remaining in the current batch, and the current prediction.

General Utility Methods

- **File Browsing:**
  A set of methods (e.g., `browse_file_train()`, `browse_file_predict()`) allow users to open file dialogs to select input and output files.

- **Status Updates:**
  The `update_status()` method updates a status label to give feedback on current operations.

- **Serial Port Refreshing:**
  `refresh_rt_ports()` retrieves the list of available serial ports and updates the dropdown menu

accordingly.

Design Considerations

- **Background Processing:**
  All lengthy operations (feature extraction, training, prediction, and real-time data reading) are performed in background threads. This design keeps the GUI responsive.

- **Modular Structure:**
  The code separates functionality into distinct GUI frames and processing functions. The use of a custom `DataProcessor` class ensures that feature extraction and data processing are decoupled from the GUI logic.

- **Error Handling & Feedback:**
  The application provides real-time feedback through status messages and pop-up dialogs. Errors during file I/O, serial communication, or processing are caught and reported to the user.

- **Extensibility:**
  The modular design allows easy extension. For example, adding a new prediction method or tweaking the feature extraction process is straightforward due to the clear separation between GUI and processing logic.

Conclusion

This classification module is a comprehensive GUI application that integrates data processing, model training, and real-time prediction. It is designed to be user-friendly with clear status updates and responsive controls, all while handling potentially large datasets and real-time serial communication. Developers can extend this code by adding new features, adjusting processing parameters, or integrating alternative machine learning models as needed.

Happy coding!
