Data Processor - Developer Documentation

This document provides an overview of the sensor data processing code, explaining the design, key functions, and how the various components work together to process sensor data. It's aimed at developers who want to understand, maintain, or extend the data processing functionality.

Overview

The primary goal of the data processor is to:

- **Extract features from sensor data:**
  It uses sliding time windows on CSV data to calculate statistical metrics (mean, standard deviation, minimum, and maximum) for multiple sensor measurements.
- **Map raw labels to human-friendly class names:**
  A label encoder (CSV file) is used to convert raw numeric or string labels into meaningful class names. If a new label is encountered, the code prompts the user to supply a class name.
- **Provide flexible processing pipelines:**
  The processor supports both window-by-window processing (returning a list of dictionaries) and batch processing (returning a DataFrame).
- **Integrate with GUI dialogs:**
  Uses Tkinter dialogs for prompting new label names.

Dependencies

The code relies on the following libraries:

- **NumPy and Pandas:**
  For numerical operations and efficient data handling.
- **OS:**
  For file path management.
- **Tkinter:**
  For simple GUI dialogs (via `simpledialog`).

Key Components

1. Feature Function Generation

`make_feature_func(column, func)`

- **Purpose:**
  Creates a custom function that applies a given statistical function (e.g., mean, std, min, max) on a

specific DataFrame column.

- **Key Details:**
  - Rounds the result (if floating point).
  - Returns `NaN` if there's an error.
- **Usage:**
  Used to generate functions that will later compute features for each sensor reading window.

`get_feature_functions(data_interval=3, sensor_count=8)`

- **Purpose:**
  Dynamically builds a dictionary of feature extraction functions.
- **What It Does:**
  - Iterates over each sensor and each measurement type (GasResistance, Temperature, Pressure, Humidity).
  - For every sensor, it creates four functions (Mean, StdDev, Min, Max) using `make_feature_func()`.
- **Returns:**
  A dictionary where each key is a descriptive feature name (e.g., `Temperature_Mean_Sensor1`) and the value is the corresponding function.

DataProcessor Class

The `DataProcessor` class encapsulates most of the data processing logic. Below is a breakdown of its key components and methods.

Initialization

```
def __init__(self, label_encoder_path=LABEL_ENCODER_PATH, features=None,
sensor_count=8, data_interval=3):
```

- **Purpose:**
  Sets up the data processor with the path to the label encoder, sensor count, and default or custom feature functions.
- **Key Actions:**
  - Loads label mapping from a CSV file.
  - Generates default feature functions if none are provided.
  - Initializes storage for processed output data.

Label Encoder Management

`load_label_encoder(path)`

- **Purpose:**
  Reads the CSV file containing raw label to class name mappings and loads it into a dictionary.
- **Error Handling:**
  If the file isn't found or reading fails, it logs a message and returns an empty dictionary.

`update_label_encoder(raw_label, new_class_name)`

- **Purpose:**
  Appends a new raw label mapping to the encoder file and updates the in-memory dictionary.
- **Key Note:**
  It appends to the file if it exists or creates a new one otherwise.

`ask_for_new_label_name(raw_label)`

- **Purpose:**
  Uses a Tkinter dialog to prompt the user for a class name when a new label is encountered.
- **Behavior:**
  - If the user cancels or provides no input, the raw label is used.
  - The new mapping is then updated in the label encoder.

CSV Data Reading and Validation

`read_csv(input_file)`

- **Purpose:**
  Reads sensor data from a given CSV file into a Pandas DataFrame.
- **Key Details:**
  Logs the number of rows read and raises appropriate errors if the file is missing or unreadable.

`check_required_columns(df)`

- **Purpose:**
  Ensures that the DataFrame has all mandatory columns before processing.
- **Mandatory Columns:**
  Includes general metadata (like `Real_Time`, `Timestamp_ms`, `Label_Tag`, `HeaterProfile_ID`) and sensor-specific measurements for each sensor.
- **Failure:**
  Throws an exception if any required columns are missing.

Data Processing

`process_data(df, window_size, stride, selected_features, data_interval=None, gap_threshold=None, progress_callback=None)`

- **Purpose:**
  Divides the DataFrame into sliding time windows, calculates selected features for each window, and assigns a class name to the window.
- **Key Steps:**
  - **Validation:**
    Calls `check_required_columns()` and sorts the DataFrame by `Real_Time`.
  - **Handling Gaps:**
    Marks gaps in the time series (using a gap threshold) and groups continuous data into blocks.

  - **Sliding Window:**
    Iterates over each continuous block, moving a window with a specified stride.
  - **Label Consistency:**
    Only processes windows where the label is consistent (i.e., only one unique label).
  - **Duration Check:**
    Ensures each window covers at least 80% of the expected duration.
  - **Label Mapping:**
    Converts raw labels to human-friendly names using the label encoder.
  - **Feature Calculation:**
    Calls `calculate_features()` for the selected features.
  - **Progress Reporting:**
    Optionally reports progress through a callback.
- **Returns:**
  A list of dictionaries where each dictionary represents the computed features and metadata for a window.

`process_batch(batch_df, window_size, stride, selected_features, data_interval=None, gap_threshold=None)`

- **Purpose:**
  Similar to `process_data()` but returns the results as a Pandas DataFrame.
- **Key Differences:**
  - Does not track progress via callback.
  - Returns a DataFrame with rounded values and reorganizes columns to place `Real_Time` first.

`calculate_features(window_df, selected_features)`

- **Purpose:**
  Applies the feature functions (previously generated) to the windowed DataFrame to compute the desired statistical features.
- **Returns:**
  A dictionary mapping each feature name (from `selected_features`) to its computed value. Handles errors by assigning `NaN`.

Saving Processed Data

`save_output(output_data, output_path)`

- **Purpose:**
  Saves the list of processed window data (or DataFrame) to a CSV file.
- **Key Behavior:**
  Rounds numerical outputs, ensures `Real_Time` is the first column, and writes the data to the specified file path.
- **Error Handling:**
  Catches exceptions during file writing and prints error messages.

Usage Notes

- **Feature Selection:**
  Before processing data, ensure you have a list of feature names that match the keys generated by `get_feature_functions()`. This determines which features are computed for each window.

- **Label Handling:**
  The system automatically prompts for new label names if it encounters an unmapped label. This is done via a simple GUI dialog and is meant for interactive use.

- **Time Window Parameters:**
  - **Window Size:** The duration of each sliding window in seconds.
  - **Stride:** The amount by which the window slides in seconds.
  - **Data Interval and Gap Threshold:** These help the processor determine data continuity and whether to treat gaps as separate segments.

- **Progress Callback:**
  If processing large datasets, you can pass a function as `progress_callback` to receive updates on the number of windows processed versus the total.

Conclusion

This code is designed to provide a flexible and interactive pipeline for processing sensor data. By breaking the task into clear components—feature extraction, label encoding, windowing, and file I/O—the code allows developers to easily extend or modify the processing steps. Whether you need to adjust the statistical functions or integrate additional validation steps, the structure provided in this module should serve as a robust foundation.

Happy coding!