Harmonic Analysis and Fault Detection

1. Introduction

This document describes the implementation and usage of signal analysis functions designed to identify harmonics and detect fault-related frequency peaks in vibration signals. The analysis leverages FFT (Fast Fourier Transform) to extract frequency domain features and compares observed peaks to known fault characteristic frequencies using configurable thresholds.

2. Prerequisites

- **Programming Language:** Python 3.11+
- **Libraries:**
  - `numpy==2.0.2`
  - `pandas==1.5.3`
  - `scipy==1.15.3`
  - `matplotlib==3.10.0`

Install dependencies using:

```
pip install -r requirements.txt
```

3. Project Structure

```
signal_classification/
├── data/                 # Raw and processed signal data
│   ├── raw/
│   └── processed/        # Processed signals (.txt)
├── notebooks/            # Jupyter notebooks for training and evaluation
├── src/                # Source code modules
│   ├── preprocessing.py
│   ├── harmonic_analysis.py
│   └── evaluate.py
├── requirements.txt      # Python dependencies
└── README.md             # This file
```

4. Code Documentation

4.1. Preprocessing Script (`src/preprocessing.py`):

```
"""
This script handles data cleaning, normalization, and feature
engineering.

Functions:
- load_signal_and_fs(path): Load a vibration signal and compute its
sampling frequency from a .txt file.

"""


def load_signal_and_fs(path):
    """
    Load a vibration signal and compute its sampling frequency from
a .txt file.

    The function reads header metadata (such as Max_X and
NoOfItems) to extract
    the total duration and number of samples, and uses them to compute
the sampling rate.
    It also loads the signal values, ignoring comment lines starting with
'%'.

    Args:
        path (str): Path to the .txt file containing the signal and metadata.

    Returns:
```

        tuple:
            - signal (np.ndarray): The loaded 1D signal array.
            - fs (float): The computed sampling frequency (samples per second).

    Raises:
        ValueError: If the necessary metadata (duration or number of samples) cannot be extracted.
    """

4.2. Analysis Script (`src/harmonic_analysis.py`):

"""
This script handles vibration signal analysis including harmonic detection and fault identification.

Functions:
- analyze_harmonics_and_possible_faults(signal, fs, base_freq, ...):
    Analyze a time-domain signal to identify harmonics and detect possible fault-related frequency peaks.

"""

analyze_harmonics_and_possible_faults(signal, fs, base_freq, ...):
"""
    Analyze a time-domain signal to identify harmonics and detect possible fault-related frequency peaks.

    Parameters
    ----
    signal : numpy.ndarray
        Input time-domain signal data array.
    fs : float
        Sampling frequency of the signal in Hz.
    base_freq : float
        Base frequency (e.g., rotational speed frequency) to calculate harmonics.
    num_harmonics : int, optional, default=10
        Number of harmonics to analyze starting from base_freq.
    harmonics : array-like or None, optional

```
        Specific array of harmonic frequencies to consider. If None,
harmonics are generated as multiples of base_freq.
    fault_freqs : dict or None, optional
        Dictionary containing fault names with their fault characteristic
frequencies and optional threshold ratios:
        {
            "FaultName": {"freq": float, "threshold_ratio": float (optional)},
            ...
        }
    freq_tolerance_factor : float, optional, default=5
        Factor to multiply with frequency resolution to define tolerance
window for matching peaks to harmonics.
    default_prom_ratio : float, optional, default=0.3
        Default amplitude threshold ratio used for fault detection if not
specified in fault_freqs.

    Returns
    -------
    pandas.DataFrame
        DataFrame listing suspicious peaks associated with faults.
Columns include:
        - Fault: Name of the detected fault.
        - Match_Type: Either "Exact_On_Harmonic" (peak very close to
harmonic) or "Near_Harmonic" (peak near harmonic).
        - Harmonic: Harmonic frequency associated with the peak.
        - Peak_Freq: Frequency of the detected peak.
        - Amplitude: Amplitude of the detected peak.
        - Delta_amp: Absolute difference between peak amplitude and
fault threshold (ratio * harmonic amplitude).

    Detailed Explanation
    --------------------
    1. Compute the FFT of the input signal and normalize by the signal
length.
    2. Extract only the positive frequency components and their
corresponding amplitudes.
    3. Calculate frequency resolution and determine tolerance window
based on freq_tolerance_factor.
    4. Generate harmonics as multiples of base_freq unless explicitly
provided.
```

    5. For each harmonic:
       - Find the closest frequency bin in the FFT result.
       - Record the harmonic frequency and its corresponding amplitude.
    6. For each harmonic and each fault frequency:
       - Identify frequency indices within the tolerance window around the harmonic.
       - Find the peak frequency and amplitude within this window.
       - Calculate the frequency difference (delta_f) between the peak and harmonic.
       - Compute delta_amp as the absolute difference between the peak amplitude and the fault threshold.
       - If the peak lies very close to the harmonic (within frequency resolution) and is stronger than the previous harmonic's amplitude, mark it as "Exact_On_Harmonic".
         Skip the first harmonic as there is no previous harmonic.
       - Otherwise, if the peak amplitude exceeds the fault threshold (ratio * harmonic amplitude), mark it as "Near_Harmonic".
    7. Collect all suspicious peaks.
    8. From multiple faults detected at the same peak frequency, keep only the fault with the smallest delta_amp (closest to threshold).
    9. Plot the FFT spectrum with harmonics and detected fault peaks highlighted.
    10. Return a DataFrame of filtered suspicious fault peaks.

    """

4.3. Evaluation Script (`src/evaluate.py`):

"""

This script provides batch processing utilities to analyze multiple vibration signal files for harmonic content and fault detection.

Functions:

- batch_analyze_folder(folder_path, output_folder, fault_freqs):Process multiple signal files in a folder, analyze harmonics and possible faults, and save the results as CSV files.

"""

batch_analyze_folder(folder_path, output_folder, fault_freqs):
    """

```
    Process multiple signal files in a folder, analyze harmonics and possible faults,
    and save the results as CSV files.

    Parameters:
    -----------
    folder_path : str
        Path to the input folder containing signal text files (.txt).
    output_folder : str
        Path to the folder where analysis CSV files will be saved.
    fault_freqs : dict
        Dictionary defining fault frequencies and parameters (see analyze_harmonics_and_possible_faults).

    Returns:
    --------
    list of str
        List of file paths to the generated CSV analysis files.

    Description:
    ------------
    1. Checks if the output folder exists; creates it if not.
    2. Iterates over all '.txt' files in the input folder.
    3. For each file:
        - Loads the signal and sampling frequency using load_signal_and_fs.
        - Calls analyze_harmonics_and_possible_faults to perform the analysis.
        - Prints the resulting DataFrame.
        - Saves the analysis results to a CSV file in the output folder.
    4. Returns a list of all saved CSV file paths.
    """
```

7. How to Run

1. Clone this repository and install dependencies:

2. git clone https://github.com/fatememajdi/Harmonic_Analysis_and_Fault_Detection.git
3. cd signal_classification
4. pip install -r requirements.txt
5. Run the notebook:
   - First: notebooks/evaluate.ipynb
6. Input .txt signal files should be placed under data/raw/.

8. Outputs

- Analysis results are saved as CSV files in /data/processed directory, with plots displayed during execution.

9. Maintenance Instructions

- Update dependencies in `requirements.txt` as needed.
- Re-run analysis whenever significant changes are made to preprocessing or fault detection logic.
- Periodically review input data files for quality and consistency.