Software Engineering Department
Braude College

Maintenance Guide

EEG Classification Using Text Compression

25-1-R-2

By:- Mohammad khateeb
Jad taha

Advisors:- Dr. Samah Idrees Ghazawi
Dr. Anat Dahan

Link to github:-
https://github.com/mhmdkh1905/EEG-recordings.git

- **Purpose**

  This guide is intended to ensure continued use, extension, or enhancement of the project after its initial delivery. It enables future users or developers to understand how the system works, update algorithms, modify preprocessing, or analyze EEG datasets using different similarity or compression techniques. The guide is focused on maintaining the core structure of the system and applying improvements in a consistent environment.

- **Environment Setup**

  This system is designed to run on Google Colab and Google Drive.

  ❖ Required Libraries:
  All libraries are installed inside the notebook. To run the system smoothly, make sure the following Python packages are available:
  - !pip install pandas
  - !pip install numpy
  - !pip install scipy
  - !pip install matplotlib
  - !pip install tqdm

  ❖ Folder Structure
  Make sure your Google Drive includes the following folder structure under /finalProject/:

```
/finalProject/
│
├── adhdcsv/                # Raw EEG CSV files (ADHD group)
├── controlcsv/             # Raw EEG CSV files (Control group)
├── filteredadhdcsv/          # Bandpass-filtered signals (ADHD)
├── filteredcontrolcsv/       # Bandpass-filtered signals (Control)
│
├── brainwave_sequence_2s/
├── brainwave_sequence_5s/
├── brainwave_sequence_6s/
├── brainwave_sequence_8s/
├── brainwave_sequence_10s/    # Dominant region text sequences
│
├── parts_ncd_2s/
├── parts_ncd_5s/
├── parts_ncd_6s/
├── parts_ncd_8s/
├── parts_ncd_10s/           # NCD comparisons between signal parts
```

```
│
├── avg/                    # NCD summary using average
│   └── 2s/channelX/
│   └── 5s/channelX/
│   └── 6s/channelX/
│   └── 8s/channelX/
│   └── 10s/channelX/
├── median/                 # NCD summary using median
│   └── 2s/channelX/
│   └── 5s/channelX/
│   └── 6s/channelX/
│   └── 8s/channelX/
│   └── 10s/channelX/
├── min/                    # NCD summary using minimum
│   └── 2s/channelX/
│   └── 5s/channelX/
│   └── 6s/channelX/
│   └── 8s/channelX/
│   └── 10s/channelX/
│
├── classification_based_on_median/     # Final classification results
│   └── 2s/                              (avg/min/median per participant)
│   └── 5s/
│   └── 6s/
│   └── 8s/
│   └── 10s/
│
└── high_scores_summary/       # Excel + graphs of participants with
                                ≥60% classification
```

- **System Logic Summary**
  The system performs the following stages:
  1. **Filtering**: Applies bandpass filter (1–40 Hz) to raw EEG files.
  2. **Segmentation**: Divides signals into windows (2s, 5s, 6s, 8s, 10s).
  3. **Region Extraction**: Converts signals into dominant brainwave text sequences.
  4. **Compression-Based Comparison**:
     - Splits sequences into 1000-character parts.
     - Computes NCD between all part-pairs of two participants.
  5. **Statistical Summaries**: Computes min, average, and median of NCD scores.
  6. **Classification**:
     - For each participant, compare NCDs with others.
     - Classifies based on whether score is ≤ participant's median.

7. **Result Aggregation**:
    - Saves classification scores to Excel.
    - Generates bar charts for summary.

- **Future Expansion Options**
  Here are some ideas for extending the project:
    - **Compression Algorithms:** Replace zlib (ZB2) with other compressors (e.g., LZMA, BZIP2) and compare results.
    - **Classification Logic:** Experiment with KNN or SVM using NCD matrices as input.
    - **Time Series Handling:** Analyze transition between frequency bands over time (add temporal modeling)
    - **GUI:** Add Streamlit web interface to run experiments via UI

- **Modifying the Code**
  All operations are centralized in finalProject2.ipynb. The notebook is modular, divided by sections:

Section Title	Purpose
#Install libraries...	Setup Colab + Mount Drive
# **preprocessing functions**	Filter EEG signals
# **Extracting Dominant EEG Band Sequences**	Convert to dominant regions
# **NCD Function**	Define NCD compression similarity
# dividing to parts	Split sequences and compute part-wise NCD
# Classification using the parts and ...	Summarize NCDs by min/avg/median
# Classification based on median ...	Perform participant-level classification
# graphs	Generate performance charts

- **Customization and Extension Guidelines**

  This section provides detailed instructions on where and how to make changes if you want to update the system — for example, to add new participants, change segmentation length, or extend the analysis.

  1. **Adding New Participants**

     What to do:

     - Place the new EEG CSV files in:
       - /adhdcsv/ for ADHD group
       - /controlcsv/ for Control group

     Filename format required:

     - v{ID}p.csv # Example: v122p.csv, v135p.csv
       - ID must be unique and numeric
       - Files must contain 19 columns (channels 0–18) and consistent sampling rate (128 Hz)

     Where it affects the code:

     - The participant ID range is often defined when calling the functions , or in functions themself:
       - Filtering

```
[ ]  # ================================================
     # ▶️ Apply Filtering Function to ADHD & Control Data
     # ================================================

     # Process ADHD participants: v1p.csv to v61p.csv
     process_files(adhd_folder, filtered_adhd_folder, start_idx=1, end_idx=61)

     # Process Control participants: v62p.csv to v121p.csv
     process_files(control_folder, filtered_control_folder, start_idx=62, end_idx=121)
```

       - Dominant region extraction

```
# ================================================================================
# 📂 Function: process_all_participants
# Purpose : Process all participants' EEG data across all 19 channels, compute dominant
#           brainwave sequences, and save them as text files (letters D/T/A/B/G)
# Input   : output_folder (destination to save labeled sequences)
# Output  : Saves one .txt file per participant per channel with dominant band sequence
# ================================================================================
def process_all_participants(output_folder):
    os.makedirs(output_folder, exist_ok=True)

    for i in range(1, 122):  # Participant IDs: v1p to v121p
        if i <= 61:
            input_path = os.path.join(filtered_adhd_folder, f"filtered_v{i}p.csv")
        else:
            input_path = os.path.join(filtered_control_folder, f"filtered_v{i}p.csv")
```

- Dividing to parts and NCD computation

```python
# ================================================================================
# 🚂 Function: process_unique_pairs_by_channel
# Purpose : For each channel, compare all unique participant pairs (v1p-v2p, v1p-v3p, ...,
#           v120p-v121p) and save NCD results using `compare_parts_and_save`
# Input   :
#   - sequence_folder (str): Folder where all sequences are stored (1 txt per participant/channel)
#   - output_base_folder (str): Where to save Excel files
#   - part_length (int): Length of parts to divide sequences into
# Output  : None (Excel files are written for each participant pair per channel)
# ================================================================================
def process_unique_pairs_by_channel(sequence_folder, output_base_folder, part_length):
    os.makedirs(output_base_folder, exist_ok=True)

    for channel in range(19):  # EEG channels 0–18
        print(f"\n🔁 Channel {channel}...")
        for p1 in tqdm(range(1, 121), desc=f"Channel {channel}"):
            for p2 in range(p1 + 1, 122):  # Only unique pairs (p2 > p1)
                compare_parts_and_save(
                    sequence_folder=sequence_folder,
                    output_folder=output_base_folder,
                    p1=p1,
                    p2=p2,
                    channel=channel,
                    part_length=part_length
                )
```

- Apply average method on NCD values of the parts

```python
def process_all_participants_all_channels_using_avg(version, base_path="/content/drive/MyDrive/finalProject/"):
    for channel in tqdm(range(19), desc="🔁 Channels"):
        input_folder = os.path.join(base_path, f"parts_ncd_{version}", f"channel{channel}")
        output_folder = os.path.join(base_path, f"avg/{version}", f"channel{channel}")
        os.makedirs(output_folder, exist_ok=True)

        if not os.path.exists(input_folder):
            print(f"⚠️ Skipping missing folder: {input_folder}")
            continue

        for participant_id in tqdm(range(1, 122), desc=f"Channel {channel}", leave=False):
            output_file = os.path.join(output_folder, f"participant_{participant_id}.txt")
```

- Apply median method on NCD values of the parts

```python
def process_all_participants_all_channels_using_median(version, base_path="/content/drive/MyDrive/finalProject/"):
    for channel in tqdm(range(19), desc="🔁 Channels"):
        input_folder = os.path.join(base_path, f"parts_ncd_{version}", f"channel{channel}")
        output_folder = os.path.join(base_path, f"median/{version}", f"channel{channel}")  # 📁 Output goes to /median
        os.makedirs(output_folder, exist_ok=True)

        if not os.path.exists(input_folder):
            print(f"⚠️ Skipping missing folder: {input_folder}")
            continue

        for participant_id in tqdm(range(1, 122), desc=f"Channel {channel}", leave=False):
            output_file = os.path.join(output_folder, f"participant_{participant_id}.txt")
```

- Apply minimum method on NCD values of the parts

```python
def process_all_participants_all_channels_using_min(version, base_path="/content/drive/MyDrive/finalProject/"):
    for channel in tqdm(range(19), desc="🔁 Channels"):
        input_folder = os.path.join(base_path, f"parts_ncd_{version}", f"channel{channel}")
        output_folder = os.path.join(base_path, f"min/{version}", f"channel{channel}")  # 📁 Save to /min folder
        os.makedirs(output_folder, exist_ok=True)

        if not os.path.exists(input_folder):
            print(f"⚠️ Skipping missing folder: {input_folder}")
            continue

        for participant_id in tqdm(range(1, 122), desc=f"Channel {channel}", leave=False):
            output_file = os.path.join(output_folder, f"participant_{participant_id}.txt")
```

- Classification based on median

```
def classification_accuracy_based_on_median(version, base_path="/content/drive/MyDrive/finalProject"):
    versions = ["avg", "median", "min"]
    channels = list(range(19))         # 0 to 18
    participants = list(range(1, 122)) # v1p to v121p
    output_base = os.path.join(base_path, "classification_based_on_median", version)
    os.makedirs(output_base, exist_ok=True)

    for participant_id in tqdm(participants, desc=f" Version {version}"):
        summary_data = []  # One row per method (avg, median, min)
```

2. Adding New Time Window Sizes

What to do:

- Duplicate existing blocks used for other versions (e.g., 2s, 5s...)
- Create new folders:
  - /brainwave_sequence_Xs/
  - /parts_ncd_Xs/
  - /avg/Xs/
  - /median/Xs/
  - /min/Xs/
  - /classification_based_on_median/Xs/

  Where X is the time window size.

Where to modify the code:

- Brainwave extraction cell:
  output_path =
  "/content/drive/MyDrive/finalProject/brainwave_sequence_Xs/"

```
[ ]  # Run dominant band processing with different window segment sizes (final output folders)
     output_path = "/content/drive/MyDrive/finalProject/brainwave_sequences_10s/"
     process_all_participants(output_path)
```

- Part-to-part NCD code:
  Sequence_folder =
  "/content/drive/MyDrive/finalProject/brainwave_sequence_Xs/"
  output_path =
  "/content/drive/MyDrive/finalProject/parts_ncd_Xs/"

```
[ ]  process_unique_pairs_by_channel(
         sequence_folder="/content/drive/MyDrive/finalProject/brainwave_sequences_2s/",
         output_base_folder="/content/drive/MyDrive/finalProject/parts_ncd_2s/",
         part_length=1000
     )
```

- Apply average method
  version = "Xs".

```
process_all_participants_all_channels_using_avg(version="2s")
```

- Apply median method
  insert in the function "Xs".

```
[ ]  process_all_participants_all_channels_using_median("10s")
```

- Apply minimum method
  insert in the function "Xs".

```
[ ]  process_all_participants_all_channels_using_min("10s")
```

- Classification based on median
  insert in the function "Xs".

```
  classification_accuracy_based_on_median("2s")
```

- Classification and plotting sections must also add new loops for Xs.
  Add “Xs” to the versions array.

```
def generate_high_classification_summary(base_path="/content/drive/MyDrive/finalProject"):
    versions = ["2s", "5s", "6s", "8s", "10s"]          # EEG window sizes
    methods = ["avg", "median", "min"]                  # Score aggregation methods
    summary_rows = []                                   # List to collect high-score entries
```

3. **Changing the Segmentation Size (Part Length)**
   - Current setup:
     part_length = 1000
   - What to do:
     Change the part_lenght for every time size window in the “dividing to parts” section.

```
process_unique_pairs_by_channel(
    sequence_folder="/content/drive/MyDrive/finalProject/brainwave_sequences_5s/",
    output_base_folder="/content/drive/MyDrive/finalProject/parts_ncd_5s/",
    part_length=1000
)
```

4. **Replacing the Compression Algorithm**
   - Current algorithm:
     import bz2
     compressed = bz2.compress(data.encode('utf-8'))
   - To switch to another (e.g., LZMA):
     import lzma

compressed = lzma.compress(data.encode('utf-8'))

- **How to Reproduce or Fix**
  If something breaks, follow this checklist:
  - Double-check all folder names in your Drive.
  - Ensure correct output path (e.g., /filteredadhdcsv/) is used in code.
  - If a method (min/avg/median) fails, confirm that all parts_ncd_*s/ folders exist.
  - Delete existing files in classification folders and re-run.