# Notes

## Python Syntax Relevant to Code

### ▼ 1. **Visualizing Array Brackets: 1D vs 2D**

Think of Python arrays/lists as **tables**:

- `[ ]` — a simple row: **[a, b, c]**
- `[[ ]]` — a table: each `[ ... ]` inside is a **row**

**Examples:**

| Shape | Example | What is it? |
|---|---|---|
| (3,) | `[1, 2, 3]` | One row, 3 columns (1D vector) |
| (1, 3) | `[[1, 2, 3]]` | Table: 1 row, 3 columns (2D) |
| (3, 1) | `[[1], [2], [3]]` | Table: 3 rows, 1 column (2D) |

### ▼ 2. Translating to .mat EEG signals in python

**Printing a mat file:**

```
print(mat_data.keys())
#output
# ⟶ dict_keys(['__header__', '__version__', '__globals__', 'interictal_seg
ment_2'])

#If we check main key
print(mat_data['interictal_segment_2'])
'''
output:
array([[(array([[ -220,  -219, ...], ...], dtype=int16),
        array([[600]], dtype=uint16),
        array([[5000]], dtype=uint16),
        array([[array(['LD_1'], dtype='<U4'), ... ]], dtype=object),
        array([[2]], dtype=uint8))]],
      dtype=[('data', 'O'), ...])
'''
```

▼ Let’s **explain in more detail** what this line means:

`array([[( ... )]], dtype=[('data', 'O'), ...])`

## ▼ What is this really?

- This is a **NumPy structured array**.
- **The outer array**: shape (1, 1), just holds the tuple.
- The notation `array([[( ... )]], dtype=[('data', 'O'), ...])` tells you:
    - You have a **2D array** (because of the `[[( ... )]]`) with shape `(1, 1)` —**one row, one column**.
    - **Inside** that one "cell" is a **tuple** (parentheses: `( ... )`) containing different fields ; like a MATLAB struct):all the fields
    - The `dtype=[('data', 'O'), ...]` means this is a **structured array**—it has "named fields" just like a C struct or a MATLAB struct.
        - Each field has a name (like `'data'`, `'sampling_frequency'`, etc) and a type (`'O'` for "object," which is generic—can be any array, int, string, etc).

## ▼ Structure Look Closer:

### Let's map each field to what it holds: in interictal_segment_2

| Position in Tuple | dtype Name | Value Example | What it Means |
|---|---|---|---|
| 0 | `'data'` | `array([[ ... ]], dtype=int16)` | EEG data: 2D array (channels, samples) |
| 1 | `'data_length_sec'` | `array([[600]], dtype=uint16)` | Number of seconds in the recording |
| 2 | `'sampling_frequency'` | `array([[5000]], dtype=uint16)` | Sampling rate (Hz) |
| 3 | `'channels'` | `array([[array(['LD_1']), ... ]], dtype=object)` | List of channel names |
| 4 | `'sequence'` | `array([[2]], dtype=uint8)` | Sequence number or ID |

## ▼ Then How does it all map ?

- **The outer array**: shape (1, 1), just holds the tuple.
- **Inside the tuple**:
    - The **first entry** is an array of type `int16` with shape `(channels, samples)` — this is the EEG data itself.
    - The **second entry** is an array with a single value, `600` — the length of the data in seconds.
    - The **third entry** is an array with a single value, `5000` — the sampling frequency.
    - The **fourth entry** is an array holding another array of channel names.
    - The **fifth entry** is an array with a value like `2` — possibly a sequence number or something else.
- Each field (entry) has a name (like `'data'`, `'data_length_sec'`, etc) as listed in the **dtype**.

### ▼ The (1,1) outer array

**In Python:**

`array([[(ARRAY)]], dtype=[('data', 'O'), ...])`

**In MATLAB:**

`interictal_segment_2 = struct('data', ..., 'sampling_frequency', ..., 'channels', ...)`

It's a
**1×1 struct**.

## ▼ 3. Basic Syntax

### ▼ 1. `[]` — Lists

- **Mutable:** You can change, add, or remove items.
- **Ordered:** Keeps the order you add things in.
- **Example:**

```
my_list = [1, 2, 3]
my_list.append(4)  # [1, 2, 3, 4]
my_list[0] = 10    # [10, 2, 3, 4]
```

## ▼ 2. `{}` — Dictionaries and Sets

- **Dictionaries (`dict`):**
  - **Key-value pairs** (think of them like real dictionaries: `word: meaning`)
  - **Mutable** and **unordered** (order preserved only in Python 3.7+)
  - **Example:**

    ```
    my_dict = {'a': 1, 'b': 2}
    my_dict['c'] = 3       # {'a': 1, 'b': 2, 'c': 3}
    print(my_dict['b'])    # 2
    ```

  - **Sets (`set`):**
    - **Unique items**, unordered, no duplicates.
    - **Example:**

    ```
    my_set = {1, 2, 3}
    my_set.add(2)  # still {1, 2, 3}
    ```

    **Note:**

    `{}` by itself creates an empty **dict** (not a set).

    To make an empty set: `set()`

## ▼ 3. `()` — Tuples and Parentheses

**Tuples:**

- **Immutable:** Can't change after creation.
- **Ordered**
- **Example:**

  ```
  my_tuple = (1, 2, 3)
  # my_tuple[0] = 5   # ERROR!
  ```

- **Parentheses:** Used for grouping in math, or for function calls:

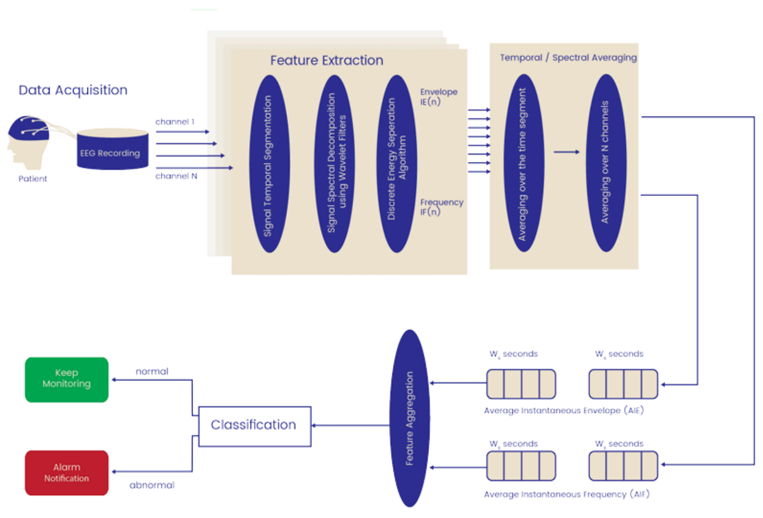
```
result = (1 + 2) * 3    # parentheses for grouping math
print("hi")           # parentheses for function calls
```

**In short:**

| Symbol | Type | Mutable? | Ordered? | Example |
|---|---|---|---|---|
| `[]` | List | Yes | Yes | `[1, 2, 3]` |
| `{}` | Dict/Set | Yes | (Dict: Yes 3.7+)Set: No | `{'a': 1}` or `{1,2,3}` |
| `()` | Tuple | No | Yes | `(1, 2, 3)` |

# ResearchPaper2

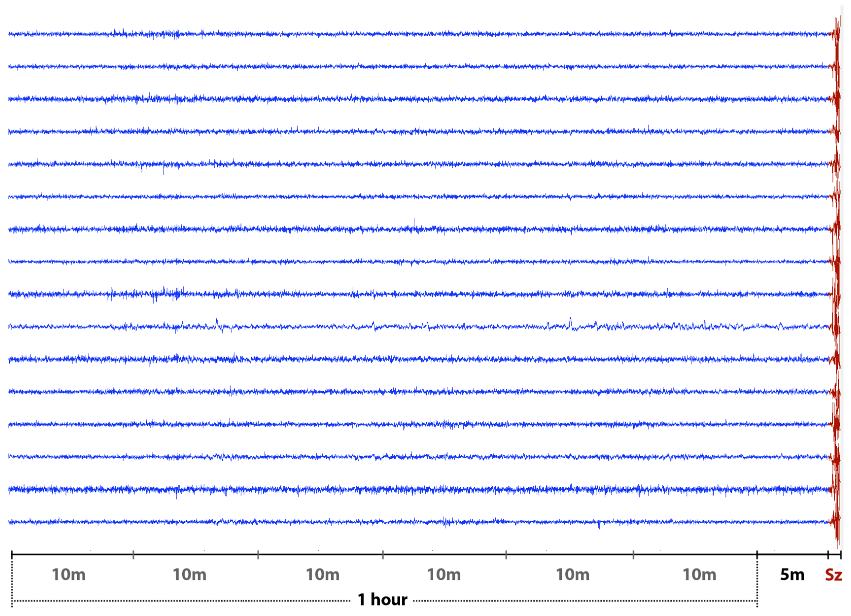
## Patient-Aware EEG-Based Feature and Classifier Selection for e-Health Epileptic Seizure Prediction


Data Acquisition
Patient
EEG Recording
channel 1
channel N
Feature Extraction
Signal Temporal Segmentation
Signal Spectral Decomposition using Wavelet Filters
Discrete Energy Seperation Algorithm
Envelope IE(n)
Frequency IF(n)
Temporal / Spectral Averaging
Averaging over the time segment
Averaging over N channels
W seconds
W seconds
Average Instantaneous Envelope (AIE)
W seconds
W seconds
Average Instantaneous Frequency (AIF)
Feature Aggregation
Classification
normal
Keep Monitoring
abnormal
Alarm Notification

## ▼ Data Acquisition

**American Epilepsy Society Seizure Prediction Challenge**

https://www.kaggle.com/competitions/seizure-prediction/data



Each .mat file contains a data structure with fields as follow:

- **data**: a matrix of EEG sample values arranged row x column as electrode x time.
- **data_length_sec**: the time duration of each data row
- **sampling_frequency**: the number of data samples representing 1 second of EEG data.
- **channels**: a list of electrode names corresponding to the rows in the data field
- **sequence**: the index of the data segment within the one hour series of clips. For example, preictal_segment_6.mat has a sequence number of 6, and represents the iEEG data from 50 to 60 minutes into the preictal data.
- **Contains 3 seizures** (each 60 minutes) that are cut up into 10 minute intervals **each of 15 channels**

## ▼ Feature Extraction

### ▼ Signal Temporal Segmentation

**General Definition**: In EEG or signal processing, *spatial segmentation* refers to dividing or processing the data based on spatial information—in this case, **across different EEG channels/electrodes** on the scalp.

In the paper:

> The first includes EEG data preprocessing where the records are
> organized into vectors of predetermined length. EEG signal in
> each channel is segmented into non-overlapping Ws epochs,
> where Ws is the window size in seconds

▼ Achieved in: **Function:** `create_feature_vectors`

- You give it an instance of `ProperSeizurePredictor` and get `params` needed to append to segment data when passed.
- it cuts the large segment to Ne samples then each segment sample gets fed to `extract_features_for_segment` as `segment_data` with its `params` , frequency `fs` , and start index `seg_start` (just for knowledge) as a element in list `all_extracted_features`

▼ Achieved in: **Function:** `extract_features_for_segment`

- Gets tuple of all parameters mentioned in `create_feature_vectors`
- It takes one single time segment of EEG data, which includes all channels for that specific time window ( `Ws` )
- For that single sample Ws cut segment, it calculates the features (using DWT and DESA) by `extract_segment_features` for each channel individually and then averages those features across all the channels.
- **Output**: It returns a single 1D NumPy array ( `avg_features` ). This array represents the spatially-averaged (across all channels) features for that one moment in time.

▼ The Data Flow: From Raw Signal to Final Features (Hints at next steps)

The process uses two main functions where one orchestrates the work and the other performs the calculations.

---

### 1. Task Preparation (Function: `create_feature_vectors` )

- **Input**: A large block of raw EEG data.
- **Action**: It chops the raw data into many small time segments. It then packages the data for each small segment, along with all necessary

parameters, into a `task` tuple.

- **Output**: A list where every item is a `task` tuple, ready to be processed.

---

## 2. Feature Calculation for One Segment (Function: `extract_features_for_segment`)

- **Input**: A single `task` tuple from the list created in Step 1.
- **Action**: This is the "worker" function. It loops through each channel of the segment's data, calling `extract_segment_features` (the DWT/DESA calculator) to get the features for that channel. After processing all channels, it averages the results.
- **Output**: A single 1D NumPy array of spatially-averaged features for that one small time segment.

---

## 3. Final Assembly (Function: `create_feature_vectors`)

- **Input**: A list of the 1D averaged feature arrays returned from all the workers in Step 2.
- **Action**: It takes this list of feature arrays and groups them into chunks of `Ne` (e.g., 6). It then **concatenates** each chunk to create a single, larger feature vector.
- **Output**: A final 2D NumPy array where each row is a complete feature vector, ready for the machine learning classifier.

## Step 1: Signal Temporal Segmentation

- **Action**: Chops the raw EEG data into many small segments based on the window size `Ws`.
- **Function**: `create_feature_vectors`

---

## Step 2: Per-Channel Feature Calculation (DWT & DESA)

- **Action**: Takes a single channel's data and calculates its features (envelope and frequency) using the Wavelet Transform and DESA.
- **Function**: `extract_segment_features`

---

## Step 3: Spatial Averaging

- **Action**: Calls the function from Step 2 for every channel and then averages the resulting feature sets across all channels.
- **Function**: `extract_features_for_segment`

---

### Step 4: Final Feature Aggregation

- **Action**: Takes `Ne` of the averaged feature arrays from Step 3 and concatenates them into one final, large feature vector.
- **Function**: `create_feature_vectors`

## ▼ Ws , Ne Clarification

Let's assume you have **60 seconds** of data to analyze, and your parameters are:

- `Ws` = 5 seconds (the size of one small segment)
- `Ne` = 6 (the number of feature sets to combine)

Here is the process:

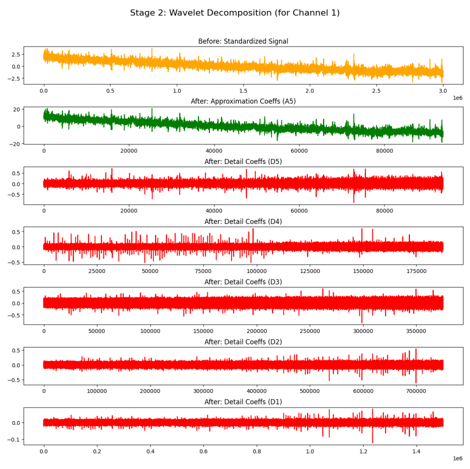
1. **Chopping the Data:** You first chop the 60-second signal into small 5-second segments.
   - 60 seconds / 5 seconds per segment = **12 small segments**.
   - The process starts with 12 segments, not 6.
2. **Calculating Features:** You calculate a feature set for each of the 12 small segments, so you get **12 feature sets**.
3. **Grouping the Features:** Now, you group these 12 feature sets into bundles of `Ne=6`.
   - The first 6 feature sets are combined to create **Final Feature Vector 1**.
   - The next 6 feature sets are combined to create **Final Feature Vector 2**.

The final output is **2** large feature vectors.

So, the number 6 ( `Ne` ) does not determine how many segments you start with. It determines how many feature sets you bundle together to create one final, bigger feature vector.

## ▼ Wavelet Decomposition & Filtering

## ▼ What is WVT?



Occurs at `extract_segment_features`

The continuous wavelet transform (CWT) can be described using the following integral:

$$X_w(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} x(t) * \Phi\left(\frac{t-\tau}{s}\right) dt \qquad (1)$$

- **What it does:**

  Splits the original signal into **multiple "bands"** (levels), each representing a specific range of frequencies.

  - *Approximation* (A): Slow trends/low frequencies.
  - *Detail* (D): Rapid changes/high frequencies.

- **Output:**

  You get several *new* signals—one for each band. For example:

  - A3 (slowest), D3, D2, D1 (faster).

- **Purpose:**

  Let you see **what is happening at different speeds** (e.g., sleep vs. spike waves).

## ▼ Code WVT

- **Mechanism:** The core of this step is the line `coeffs = pywt.wavedec(segment, wavelet, level=levels)`. This calls the `wavedec` (wavelet decomposition) function from the `pywt` library.

  - `segment`: The input EEG epoch from the previous step.

    - `wavelet`: The "mother wavelet" (e.g., 'db6', 'coif1'), a key parameter being optimized.
    - `level=levels`: The number of decomposition levels, corresponding directly to the `NL` parameter from the paper.
    - cascade explained in ***The Data Flow: From Raw Signal to Final Features (Hints at next steps)***

▼ Discrete Energy Separation Algorithm (DESA)

## DESA

- **What it does:**

  For **each signal (usually after wavelet decomposition)**, computes for every time point:

    - *How strong is the wave here?* (**Envelope**)
    - *How fast is the wave oscillating?* (**Frequency**)

- **Output:**

  Two new arrays per signal:

    - Envelope (IE): Size/amplitude at every point.
    - Frequency (IF): Oscillation speed at every point.

- **Purpose:**

  Show **how amplitude and frequency change over time**—even within a single band.

▼ The frequency is the signals frequency:

### Sampling Frequency vs. Signal Frequency

A simple analogy is filming a car with a video camera.

- **Sampling Frequency (256 Hz)**: This is the frame rate of your camera (e.g., 24 frames per second). It's a fixed setting of your recording device that determines how often you take a picture. It does not change. In the code, this is `fs`.

        - **Signal's Frequency (what DESA calculates)**: This is the actual speed of the car you are filming. The car can speed up (high frequency) or slow down (low frequency). This property of the car itself is constantly changing.

    ▼ DESA Speed Calculations

    DESA tells you the **numerical value** of the frequency at each moment, not the *name* of the band (like "Delta" or "Alpha").

    For example, if the brain is producing an Alpha wave (which is between 8-12 Hz), the DESA algorithm will not output the word "Alpha". Instead, it will output a series of numerical frequency values that will hover in the 8-12 Hz range.

    The final feature used by the code is the *average* of these numerical frequency values, which the machine learning model then uses to help make its prediction.

## ▼ Temporal and Spatial Averaging

**What the paper proposes:** To create a robust feature set, the algorithm performs two averaging steps to reduce fluctuations.

- **Temporal Averaging:** For each wavelet level within a single segment, the mean of the estimated `a[n]` and `Ω[n]` values is calculated. This yields the

**Temporal IE (TIE)** and **Temporal IF (TIF)**.

- **Spatial Averaging:** The TIE and TIF values are then averaged across all EEG channels. This step compensates for variations between different electrode locations and produces the

**Average IE (AIE)** and **Average IF (AIF)** vectors.

**How the code achieves this**:

The two averaging steps are implemented in different functions, reflecting the hierarchy of the process.

- **Temporal Averaging** (Function: `extract_segment_features`):
    - Mechanism: After `_desa_algorithm` returns the envelope and frequency arrays, the code calculates their mean: `tie = np.mean(amplitude)` and `tif = np.mean(frequency)`. This is done for each wavelet level of a single segment from a single channel.
- **Spatial Averaging** (Function: `extract_features_for_segment`):

- Mechanism: This function orchestrates the process for a single time segment across all channels. It calls `extract_segment_features` for each channel to get a list of TIE/TIF feature vectors. Then, the line `avg_features = np.mean(channel_features, axis=0)` computes the final AIE/AIF vector for that segment by averaging across the channel dimension