Evolutionary Algorithms for Neuromorphic Anomaly Detection - Documentation

Project Overview

Author: Ivayla Markova
Institution: Technical University of Sofia, Bulgaria
Subject: Evolutionary Algorithms
Topic: Evolutionary Algorithms for Neuromorphic Anomaly Detection

Description

This project focuses on epilepsy detection, one of the most common neurological disorders affecting over 50 million people globally. Seizures appear as rare abnormal patterns inside EEG signals, making this a neuromorphic anomaly-detection problem. The approach targets automatic real-time recognition of seizure events from brain activity.

Database: CHB-MIT Scalp EEG Database

Table of Contents

Dependencies

Required Libraries

```python
```

```
pyedflib      # EDF file reading
numpy         # Numerical computations
matplotlib    # Visualization
scipy         # Statistical functions
PyWavelets    # Wavelet transforms
pandas        # Data manipulation
deap          # Evolutionary algorithms
sklearn       # Machine learning metrics
```

Installation

bash

```
pip install pyedflib numpy matplotlib scipy PyWavelets pandas deap
```

Data Loading and Preprocessing

Configuration Parameters

Parameter	Value	Description
`fs`	256 Hz	Sampling frequency
`channel`	0	EEG channel to analyze
`window_sec`	5 seconds	Window size for segmentation
`window`	1280 samples	Window size in samples (5s × 256 Hz)

Input Files

python

```
files = [
    "/chb01_01.edf", "/chb01_02.edf", "/chb01_03.edf",
    "/chb01_04.edf", "/chb01_05.edf", "/chb01_06.edf",
    "/chb01_07.edf"
]
```

Seizure Annotations

python

```
seizure_intervals = {
    "/chb01_03.edf": (2996, 3036),  # seconds
    "/chb01_04.edf": (1467, 1494)   # seconds
}
```

Data Segmentation Process

Function: Data loading and windowing

Input:

- EDF files containing EEG recordings
- Seizure interval annotations

Output:

- `signals`: numpy array of shape (N, 1280) - N windowed segments
- `labels`: numpy array of shape (N,) - Binary labels (0=normal, 1=seizure)

Algorithm:

1. Load EEG signal from specified channel
2. Divide signal into non-overlapping 5-second windows
3. For each window:
    - Calculate temporal range [t_start, t_end]
    - Assign label based on overlap with seizure intervals
    - Label = 1 if window overlaps with seizure period
    - Label = 0 otherwise

Dataset Statistics:

- Total windows: 5,040
- Seizure windows (label=1): 11
- Normal windows (label=0): 5,029
- Class balance: 0.2% seizures (highly imbalanced)

Feature Extraction

`extract_features(sig)`

Description: Extracts three time-domain features from each EEG window segment.

Parameters:

- `sig` (numpy.ndarray): Input signal segment of length 1280 samples

Returns:

- `numpy.ndarray`: Feature vector of length 3 containing [energy, line_length, entropy]

Features:

1. **Energy**

```python
energy = np.sum(sig ** 2)
```

- Measures signal power
- Higher values indicate increased amplitude activity
- Typically elevated during seizures

2. **Line Length**

```python
line_length = np.sum(np.abs(np.diff(sig)))
```

- Sum of absolute differences between consecutive samples
- Measures signal complexity and variability
- Sensitive to rapid fluctuations characteristic of seizures

3. **Entropy**

```python
entropy = entropy(np.abs(sig) + 1e-6)
```

- Shannon entropy of absolute signal values
- Measures signal randomness/predictability
- Small constant (1e-6) added for numerical stability

Output Shape: (5040, 3) - Feature matrix for all windows

Evolutionary Algorithm

Purpose

Optimize feature weights and threshold for anomaly detection using genetic algorithm.

Anomaly Score Calculation

```
score = w1 × energy + w2 × line_length + w3 × entropy
```

Classification Rule

```python
if score > threshold:
    prediction = 1  # seizure
else:
    prediction = 0  # normal
```

Algorithm Components

1. Individual Representation

Genome Structure:

- 4 genes: [w1, w2, w3, threshold]
- Each gene: continuous float in range [0, 1]

```python
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
```

2. Population Initialization

Parameters:

- Population size: 20 individuals
- Initialization: Random uniform [0, 1]

```python
toolbox.register("attr_float", lambda: random.uniform(0, 1))
toolbox.register("individual", tools.initRepeat, creator.Individual,
                 toolbox.attr_float, 4)
```

3. Fitness Function: `evaluate(ind)`

Description: Evaluates individual based on classification performance.

Parameters:

- `ind` (list): Individual with 4 genes [w1, w2, w3, threshold]

Returns:

- `tuple`: Single-value tuple containing fitness score

Algorithm:

1. Extract weights: `w = [w1, w2, w3]`
2. Extract threshold: `thresh = ind[3]`
3. Calculate anomaly scores: `scores = features @ w`
4. Generate predictions: `preds = (scores > thresh)`
5. Compute confusion matrix
6. Calculate metrics:
   - Sensitivity (Recall): `TP / (TP + FN)`
   - Specificity: `TN / (TN + FP)`
7. Fitness = 0.5 × (Sensitivity + Specificity)

Fitness Objective: Maximize balanced accuracy between sensitivity and specificity

4. Genetic Operators

Crossover (Mating):

```python
toolbox.register("mate", tools.cxBlend, alpha=0.3)
```

- Type: Blend crossover
- Alpha: 0.3 (extent of blending)

Mutation:

```python
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.2, indpb=0.5)
```

- Type: Gaussian mutation
- Mean (μ): 0
- Standard deviation (σ): 0.2
- Gene mutation probability: 0.5

Selection:

```python
toolbox.register("select", tools.selTournament, tournsize=3)
```

- Type: Tournament selection
- Tournament size: 3

5. Evolution Parameters

```python
algorithms.eaSimple(
    pop,
    toolbox,
    cxpb=0.5,    # Crossover probability: 50%
    mutpb=0.4,   # Mutation probability: 40%
    ngen=30,     # Number of generations: 30
    verbose=False
)
```

6. Best Solution

Selection:

```python
best = tools.selBest(pop, 1)[0]
```

Example Output:

```python
```

```
[-0.530, 0.096, 0.727, 0.201]
# Interpretation:
# w1 (energy weight): -0.530
# w2 (line_length weight): 0.096
# w3 (entropy weight): 0.727
# threshold: 0.201
```

Results Visualization

Seizure Detection Timeline Plot

Function: Visualizes predicted vs. actual seizure events over time

Code:

```python
plt.figure(figsize=(12, 4))
plt.plot(preds, label="Predicted", color='orange', alpha=1, linewidth=7)
plt.plot(labels, label="True", color='black', alpha=0.7, linewidth=1.5)
plt.fill_between(range(len(labels)), 0, 1, where=labels==1,
                 color='red', alpha=0.3, label="Seizure Periods")
plt.fill_between(range(len(labels)), 0, 1, where=labels==0,
                 color='green', alpha=0.15, label="Normal Periods")
```

Plot Elements:

- **Orange thick line:** EA predictions
- **Black thin line:** Ground truth labels
- **Red shaded regions:** Actual seizure periods
- **Green shaded regions:** Normal periods
- **X-axis:** Window index (each unit = 5 seconds)
- **Y-axis:** Binary label (0 or 1)

Interpretation:

- Overlap between orange and black lines indicates correct predictions
- Visual assessment of sensitivity (catching seizures) and specificity (avoiding false alarms)

Algorithm Workflow

mermaid

```
graph TD
    A[Load EEG Data] --> B[Segment into 5s Windows]
    B --> C[Extract Features]
    C --> D[Initialize Population]
    D --> E[Evaluate Fitness]
    E --> F{Convergence?}
    F -->|No| G[Selection]
    G --> H[Crossover]
    H --> I[Mutation]
    I --> E
    F -->|Yes| J[Select Best Individual]
    J --> K[Apply Weights & Threshold]
    K --> L[Generate Predictions]
    L --> M[Visualize Results]
```

Key Metrics

Confusion Matrix Components

- **TP (True Positives):** Seizures correctly detected
- **TN (True Negatives):** Normal windows correctly identified
- **FP (False Positives):** False alarms
- **FN (False Negatives):** Missed seizures

Performance Metrics

- **Sensitivity (Recall):** TP / (TP + FN) - Ability to detect seizures
- **Specificity:** TN / (TN + FP) - Ability to avoid false alarms
- **Balanced Accuracy:** (Sensitivity + Specificity) / 2

Usage Example

python

```
# 1. Load and preprocess data
signals, labels = load_eeg_data(files, seizure_intervals)

# 2. Extract features
features = np.array([extract_features(sig) for sig in signals])

# 3. Run evolutionary algorithm
best_individual = run_ea(features, labels)

# 4. Apply optimized weights
weights = best_individual[:3]
threshold = best_individual[3]
scores = features @ weights
predictions = (scores > threshold).astype(int)

# 5. Evaluate and visualize
plot_results(predictions, labels)
```

Challenges and Considerations

1. Class Imbalance

- **Problem:** Only 0.2% of windows contain seizures
- **Impact:** Model may bias toward predicting normal state
- **Solution:** Balanced fitness function (sensitivity + specificity)

2. Feature Selection

- **Current:** Time-domain features (energy, line length, entropy)
- **Potential improvements:** Frequency-domain features, wavelets, nonlinear dynamics

3. Overfitting Risk

- **Issue:** Small number of seizure examples (11 windows)
- **Mitigation:** Simple model with only 4 parameters, balanced fitness

4. Real-time Constraints

- **Window size:** 5 seconds introduces detection delay
- **Feature computation:** Fast time-domain calculations suitable for real-time

Future Enhancements

1. **Cross-validation:** Implement k-fold CV for robust evaluation
2. **Multi-objective optimization:** Balance sensitivity, specificity, and computational cost
3. **Feature engineering:** Add frequency-domain and wavelet features
4. **Patient-specific models:** Train individual models per patient
5. **Online learning:** Adapt weights during deployment
6. **Sliding windows:** Reduce detection latency with overlapping segments

---

References

- Database: Goldberger, A., et al. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation, 101(23), e215-e220.
- CHB-MIT Database: https://physionet.org/content/chbmit/1.0.0/
- DEAP Library: Fortin, F.A., et al. (2012). DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research, 13, 2171-2175.

License

Last Updated: December 2024
Version: 1.0