Hindawi Publishing Corporation
Computational Intelligence and Neuroscience
Volume 2011, Article ID 406391, 7 pages
doi:10.1155/2011/406391

Research Article

PyEEG: An Open Source Python Module for EEG/MEG Feature Extraction

Forrest Sheng Bao,[1] Xin Liu,[2] and Christina Zhang[3]

[1] *Department of Computer Science, Department of Electrical Engineering, Texas Tech University, Lubbock TX 79409-3104, USA*
[2] *ECHO Labs, Nanjing University of Posts and Telecommunications, Nanjing 210003, China*
[3] *Department of Physiology, McGill University, Montreal, QC, Canada H3G 1Y6*

Correspondence should be addressed to Forrest Sheng Bao, forrest.bao@gmail.com

Received 31 August 2010; Revised 26 October 2010; Accepted 31 December 2010

Academic Editor: Sylvain Baillet

Computer-aided diagnosis of neural diseases from EEG signals (or other physiological signals that can be treated as time series, e.g., MEG) is an emerging field that has gained much attention in past years. Extracting features is a key component in the analysis of EEG signals. In our previous works, we have implemented many EEG feature extraction functions in the Python programming language. As Python is gaining more ground in scientific computing, an open source Python module for extracting EEG features has the potential to save much time for computational neuroscientists. In this paper, we introduce PyEEG, an open source Python module for EEG feature extraction.

1. Introduction

Computer-aided diagnosis based on EEG has become possible in the last decade for several neurological diseases such as Alzheimer's disease [1, 2] and epilepsy [3, 4]. Implemented systems can be very useful in the early diagnosis of those diseases. For example, traditional epilepsy diagnosis may require trained physicians to visually screen lengthy EEG records whereas computer-aided systems can shorten this time-consuming procedure by detecting and picking out EEG segments of interest to physicians [5, 6]. On top of that, computers can extend our ability to analyze signals. Recently, researchers have developed systems [3, 4, 7, 8] that can hopefully use (any) random interictal (i.e., non-seizure) EEG records for epilepsy diagnosis in instances that are difficult for physicians to make diagnostic decisions with their naked eyes. In addition to analyzing existing signals, this computer-based approach can help us model the brain and predict future signals, for example, seizure prediction [9, 10].

All the above systems rely on characterizing the EEG signal into certain features, a step known as feature extraction. EEG features can come from different fields that study time series: power spectral density from signal processing, fractal dimensions from computational geometry, entropies from information theory, and so forth. An open source tool that can extract EEG features would benefit the computational neuroscience community since feature extraction is repeatedly invoked in the analysis of EEG signals. Because of Python's increasing popularity in scientific computing, and especially in computational neuroscience, a Python module for EEG feature extraction would be highly useful. In response, we have developed PyEEG, a Python module for EEG feature extraction, and have tested it in our previous epileptic EEG research [3, 8, 11].

Compared to other popular programming languages in scientific computing such as C++ or MATLAB, Python is an open source scripting language of simple syntax and various high-level libraries (for detailed advantages of Python, read http://www.python.org/about/), such as Scipy (http://www.scipy.org/) which allows users to run MATLAB codes after slight modification. There have been several popular open source Python projects in the neuroimaging community already, such as NIPY (http://nipy.sourceforge.net/). However, in neural physiology community, Python is not yet quite popular. As we are not aware of any open source tools in Python (or other programming languages) that can extract

EEG features as mentioned above, we introduce and release PyEEG in this paper.

Though originally designed for EEG, PyEEG can also be used to analyze other physiological signals that can be treated as time series, especially MEG signals that represent the magnetic fields induced by currents of neural electrical activities.

The rest of the paper is organized as follows. In Section 2, we introduce the framework of PyEEG. Section 3 gives the definitions to compute EEG features. A tutorial of applying PyEEG onto a public real EEG dataset is given in Section 4. Section 5 concludes the paper.

2. Main Framework

PyEEG's target users are programmers (anyone who writes programs) working on computational neuroscience. Figure 1 shows its framework. PyEEG is a Python module that focuses only on extracting features from EEG/MEG segments. Therefore, it does not contain functions to import data of various formats or export features to a classifier. This is due to the modularity and composition principles of building open source software which indicate that small programs that can work well together via simple interfaces are better than big monolithic programs. Since open source tools like EEG/MEG data importers (e.g., EEGLab, Biosig, etc.) and classifier front-ends are already available, there is no need for us to reinvent the wheel. Users can easily hook PyEEG up with various existing open source software to build toolchains for their EEG/MEG research.

PyEEG consists of two sets of functions.

(1) *Preprocessing functions*, which do not return any feature values. Only two such functions have been implemented so far. `embed_seq()` builds embedding sequence (from given lag and embedding dimension) and `first_order_diff()` computes first-order differential sequence. One can build differential sequences of higher orders by repeatedly applying first-order differential computing.

(2) *Feature extraction functions*, that return feature values. These are listed in Table 1.

PyEEG only uses functions in standard Python library and SciPy, the *de facto* Python module for scientific computing. PyEEG does not define any new data structure, but instead uses only standard Python and NumPy data structures. The reason is that we want to simplify the use of PyEEG, especially for users without much programming background. The inputs of all functions are a time sequence as a list of floating-point numbers and a set of optional feature extraction parameters. Parameters have default values. The output of a feature extraction function is a floating-point number if the feature is a scalar or a list of floating-point numbers (a vector) otherwise. Details about functions are available in the PyEEG reference guide at http://PyEEG.SourceForge.net/.

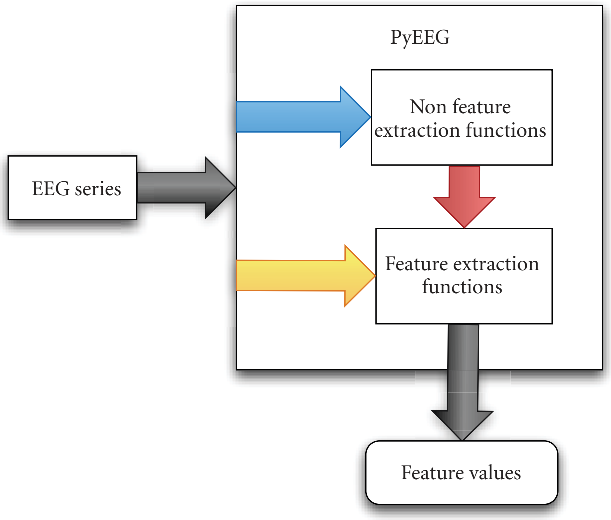
PyEEG
EEG series
Non feature extraction functions
Feature extraction functions
Feature values

Figure 1: PyEEG framework.

3. Supported Feature Extraction

In this section, we detail the definitions and computation procedures to extract EEG features (as shown in Table 1) in PyEEG. Since there are many parameters and various algorithms for one feature, the numerical value of a feature extracted by PyEEG may be different from that extracted by other toolboxes. Users may need to adjust our code or use non-default values for the parameters in order to meet their needs. Please note that the index of an array or a vector starts from 1 rather than 0 in this section.

3.1. Power Spectral Intensity and Relative Intensity Ratio. To a time series $[x_1, x_2, \ldots, x_N]$, denote its Fast Fourier Transform (FFT) result as $[X_1, X_2, \ldots, X_N]$. A continuous frequency band from f_{low} to f_{up} is sliced into K bins, which can be of equal width or not. Boundaries of bins are specified by a vector $band = [f_1, f_2, \ldots, f_K]$, such that the lower and upper frequencies of the ith bin are f_i and f_{i+1}, respectively. Commonly used unequal bins are EEG/MEG rhythms, which are, δ (0.5–4 Hz), θ (4–7 Hz), α (8–12 Hz), β (12–30 Hz), and γ (30–100 Hz). For these bins, we have $band = [0.5, 4, 7, 12, 30, 100]$.

The Power Spectral Intensity (PSI) [12] of the kth bin is evaluated as

$$\text{PSI}_k = \sum_{i=\lfloor N(f_k/f_s) \rfloor}^{\lfloor N(f_{k+1}/f_s) \rfloor} |X_i|, \quad k = 1, 2, \ldots, K-1, \tag{1}$$

where f_s is the sampling rate, and N is the series length.

Relative Intensity Ratio (RIR) [12] is defined on top of PSI

$$\text{RIR}_j = \frac{\text{PSI}_j}{\sum_{k=1}^{K-1} \text{PSI}_k}, \quad j = 1, 2, \ldots, K-1. \tag{2}$$

PSI and RIR are both vector features.

TABLE 1: PyEEG-supported features and extraction functions with their return types.

Feature name	Function name	Return type
Power Spectral Intensity (PSI) and Relative Intensity Ratio (RIR)	`bin_power()`	Two 1-D vectors
Petrosian Fractal Dimension (PFD)	`pdf()`	A scalar
Higuchi Fractal Dimension (HFD)	`hfd()`	A scalar
Hjorth mobility and complexity	`hjorth()`	Two scalars
Spectral Entropy (Shannon's entropy of RIRs)	`spectral_entropy()`	A scalar
SVD Entropy	`svd_entropy()`	A scalar
Fisher Information	`fisher_info()`	A scalar
Approximate Entropy (ApEn)	`ap_entropy()`	A scalar
Detrended Fluctuation Analysis (DFA)	`dfa()`	A scalar
Hurst Exponent (Hurst)	`hurst()`	A scalar

3.2. Petrosian Fractal Dimension (PFD). To a time series, PFD is defined as

$$\mathrm{PFD} = \frac{\log_{10} N}{\log_{10} N + \log_{10}(N/(N + 0.4N_\delta))}, \tag{3}$$

where N is the series length, and N_δ is the number of sign changes in the signal derivative [13]. PFD is a scalar feature.

3.3. Higuchi Fractal Dimension (HFD). Higuchi's algorithm [14] constructs k new series from the original series $[x_1, x_2, \ldots, x_N]$ by

$$x_m, x_{m+k}, x_{m+2k}, \ldots, x_{m+\lfloor (N-m)/k \rfloor k}, \tag{4}$$

where $m = 1, 2, \ldots, k$.

For each time series constructed from (4), the length $L(m, k)$ is computed by

$$L(m,k) = \frac{\sum_{i=2}^{\lfloor (N-m)/k \rfloor} |x_{m+ik} - x_{m+(i-1)k}| (N-1)}{\lfloor (N-m)/k \rfloor k}. \tag{5}$$

The average length is computed as $L(k) = [\sum_{i=1}^{k} L(i,k)]/k$.

This procedure repeats $k_{\max}$ times for each k from 1 to $k_{\max}$, and then uses a least-square method to determine the slope of the line that best fits the curve of $\ln(L(k))$ versus $\ln(1/k)$. The slope is the Higuchi Fractal Dimension. HFD is a scalar feature.

3.4. Hjorth Parameters. To a time series $[x_1, x_2, \ldots, x_N]$, the Hjorth mobility and complexity [15] are, respectively, defined as $\sqrt{M2/\mathrm{TP}}$ and $\sqrt{(M4 \cdot \mathrm{TP})/(M2 \cdot M2)}$, where $\mathrm{TP} = \sum x_i/N$, $M2 = \sum d_i/N$, $M4 = \sum (d_i - d_{i-1})^2/N$, and $d_i = x_i - x_{i-1}$. Hjorth mobility and complexity are both scalar features.

3.5. Spectral Entropy. The spectral entropy [16] is defined as follows

$$H = -\frac{1}{\log(K)} \sum_{i=1}^{K} \mathrm{RIR}_i \log \mathrm{RIR}_i, \tag{6}$$

where RIR_i and K are defined in (2). Spectral entropy is a scalar feature.

3.6. SVD Entropy. Reference [17] defines an entropy measure using Singular Value Decomposition (SVD). Let the input signal be $[x_1, x_2, \ldots, x_N]$. We construct delay vectors as

$$\mathbf{y}(i) = [x_i, x_{i+\tau}, \ldots, x_{i+(d_E-1)\tau}], \tag{7}$$

where τ is the delay and d_E is the embedding dimension. In this paper, $d_E = 20$ and $\tau = 2$. The embedding space is then constructed by

$$Y = [\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(N - (d_E - 1)\tau)]^T. \tag{8}$$

The SVD is then performed on matrix Y to produce M singular values, $\sigma_1, \ldots, \sigma_M$, known as the singular spectrum.

The SVD entropy is then defined as

$$H_{\mathrm{SVD}} = -\sum_{i=1}^{M} \overline{\sigma}_i \log_2 \overline{\sigma}_i, \tag{9}$$

where M is the number of singular values and $\overline{\sigma}_1, \ldots, \overline{\sigma}_M$ are normalized singular values such that $\overline{\sigma}_i = \sigma_i / \sum_{j=1}^{M} \sigma_j$. SVD entropy is a scalar feature.

3.7. Fisher Information. The Fisher information [18] can be defined in normalized singular spectrum used in (9)

$$I = \sum_{i=1}^{M-1} \frac{(\overline{\sigma}_{i+1} - \overline{\sigma}_i)^2}{\overline{\sigma}_i}. \tag{10}$$

Fisher information is a scalar feature.

3.8. Approximate Entropy. Approximate entropy (ApEn) is a statistical parameter to quantify the regularity of a time series [19].

ApEn is computed by the following steps.

(1) Let the input signal be $[x_1, x_2, \ldots, x_N]$.

(2) Build subsequence $x(i, m) = [x_i, x_{i+1}, \ldots, x_{i+m-1}]$ for $1 \le i \le N - m$, where m is the length of the subsequence. In [7], $m = 1, 2$, or 3.

(3) Let r represent the noise filter level, defined as $r = k \times \mathrm{SD}$ for $k = 0, 0.1, 0.2, \ldots, 0.9$.

(4) Build a set of subsequences $\{x(j,m)\} = \{x(j,m) \mid j \in [1..N-m]\}$, where $x(j,m)$ is defined in step 2.

(5) For each $x(i,m) \in \{x(j,m)\}$, compute

$$C(i,m) = \frac{\sum_{j=1}^{N-m} k_j}{N-m}, \tag{11}$$

where

$$k_j = \begin{cases} 1 & \text{if } |x(i,m) - x(j,m)| < r, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

(6)

$$\text{ApEn}(m,r,N) = \frac{1}{N-M}\left[\sum_{i=1}^{N-m} \ln \frac{C(i,m)}{C(i,m+1)}\right]. \tag{13}$$

ApEn is a scalar feature.

3.9. Detrended Fluctuation Analysis. Detrended Fluctuation Analysis (DFA) is proposed in [20].

The procedures to compute DFA of a time series $[x_1, x_2, \ldots, x_N]$ are as follows.

(1) First integrate x into a new series $y = [y(1), \ldots, y(N)]$, where $y(k) = \sum_{i=1}^{k}(x_i - \overline{x})$ and $\overline{x}$ is the average of $x_1, x_2, \ldots, x_N$.

(2) The integrated series is then sliced into boxes of equal length n. In each box of length n, a least-squares line is fit to the data, representing the *trend* in that box. The y coordinate of the straight line segments is denoted by $y_n(k)$.

(3) The root-mean-square fluctuation of the integrated series is calculated by $F(n) = \sqrt{(1/N)\sum_{k=1}^{N}[y(k) - y_n(k)]^2}$, where the part $y(k) - y_n(k)$ is called detrending.

(4) The fluctuation can be defined as the slope of the line relating $\log F(n)$ to $\log n$.

DFA is a scalar feature.

3.10. Hurst Exponent. The hurst exponent (HURST) [21] is also called Rescaled Range statistics (R/S). To calculate the hurst exponent for time series $X = [x_1, x_2, \ldots, x_N]$, the first step is to calculate the accumulated deviation from the mean of time series within range T

$$X(t,T) = \sum_{i=1}^{t}(x_i - \overline{x}), \quad \text{where } \overline{x} = \frac{1}{T}\sum_{i=1}^{T} x_i, \ t \in [1..N]. \tag{14}$$

Then, $\mathrm{R}(T)/\mathrm{S}(T)$ is calculated as

$$\frac{\mathrm{R}(T)}{\mathrm{S}(T)} = \frac{\max(X(t,T)) - \min(X(t,T))}{\sqrt{(1/T)\sum_{t=1}^{T}[x(t) - \overline{x}]^2}}. \tag{15}$$

The Hurst Exponent is obtained by calculating the slope of the line produced by $\ln(\mathrm{R}(n)/\mathrm{S}(n))$ versus $\ln(n)$ for $n \in [2..N]$. Hurst Exponent is a scalar feature.

4. Using PyEEG on Real Data

In this section, we use PyEEG on a real EEG dataset to demonstrate its use in everyday research.

The dataset (http://epileptologie-bonn.de/cms/front_content.php?idcat=193&lang=3), from Klinik für Epileptologie, Universität Bonn, Germany [22], has been widely used in previous epilepsy research. In total, there are five sets, each containing 100 single-channel EEG segments. Each segment has 4096 samples. Data in sets A and B are extracranial EEGs from 5 healthy volunteers with eyes open and eyes closed, respectively. Sets C and D are intracranial data over interictal periods while Set E over ictal periods. Segments in D are from within the epileptogenic zone, while those in C are from the hippocampal formation of the opposite hemisphere of the brain. Sets C, D, and E are composed from EEGs of 5 patients. The data had a spectral bandwidth of 0.5–85 Hz. Please refer to [22] for more details.

Using PyEEG is like using any other Python module. Users simply need to import PyEEG and then call its functions as needed. PyEEG is provided as a single Python file. Therefore, it only needs to be downloaded and placed under a directory on Python module search paths, such as the working directory. Alternatively, `PYTHONPATH` environment variable can be set to point to the location of PyEEG.

On Python interpreter, we first import PyEEG and load the data

```
>>> import pyeeg
>>> fid = open('Z001.txt', 'r')
>>> tmp = fid.readlines()
>>> data = [float(k) for k in tmp]
```

where `Z001.txt` is the first segment in set A. The data type of `data` is `list`. After loading EEG data, we can use PyEEG to extract features as follows (using all default parameters):

```
>>> DFA = pyeeg.dfa(data)
>>> DFA
0.81450526948129354
>>> Hurst_Exponent = pyeeg.hurst(data)
>>> Hurst_Exponent
0.68053321812240675
>>> PFD = pyeeg.pfd(data)
>>> PFD
0.58651018327048932
```

Due to space limitations, we are not able to print all feature values of all EEG segments. Instead, we visualize the averages of the features (except RIR and PSI) within each of the five sets in Figure 2. Error bars represent the variances of features in each set. PSIs for five sets are plotted in Figure 3. Users can replot these pictures and get averages of features on Python interpreter by a testing script (http://code.google.com/p/pyeeg/wiki/TestScript) from our project website.

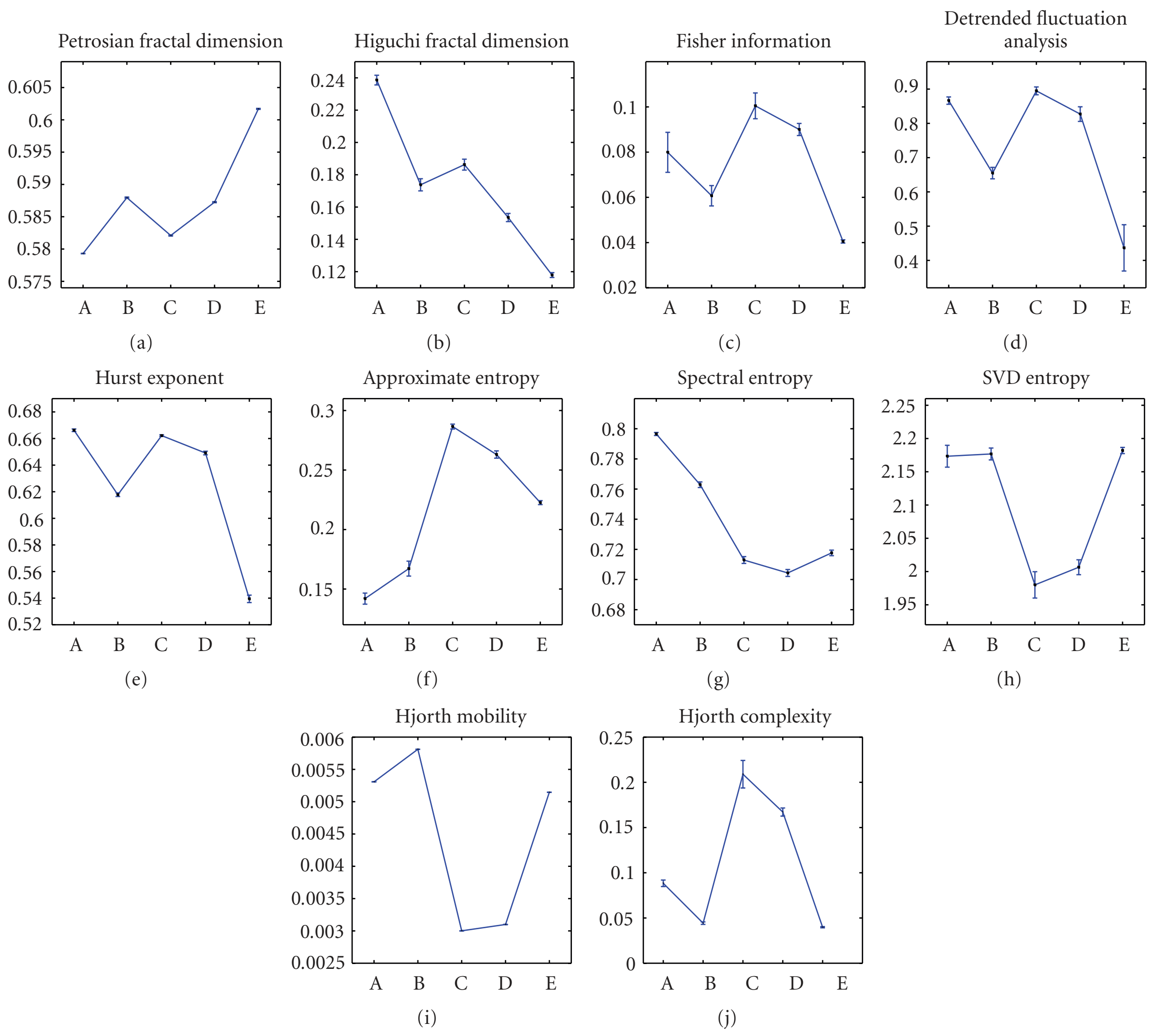

Figure 2: Distributions of ten features extracted by PyEEG in each set.

Table 2: Values of parameters used in our example.

Parameter name	Value	In feature(s)
$k_{\max}$	5	HFD
τ	4	SVD Entropy
d_E	10	Fisher Information
r	0.3σ[1]	ApEn
m	10	
f_s	173	Spectral Entropy
band	$[1, 3, 5, \ldots, 85]$	PSI and RIR

[1]σ is the standard deviation of the EEG segment.

From Figures 2 and 3, we can see that healthy, interictal, and ictal EEG signals have different distributions for most features. Table 2 lists parameters used in this experiment.

5. Discussion and Future Development

So far, we have listed features that can be extracted by PyEEG and their definitions. Our implementation sticks on their definitions precisely even though faster algorithms may exist. There are many other EEG features, such as Lyapunov Exponents, that have not been yet implemented in PyEEG. More EEG features will be added into PyEEG in the future while we finish unit testing and documentation for each function. In personal emails, some open source projects, such as ConnectomeViewer (http://www.connectomeviewer.org/viewer) and NIPY/PBrain (http://nipy.sourceforge.net/pbrain/), have expressed the interest in including PyEEG into their code. Therefore, we will keep maintaining PyEEG as long as it can benefit the entire computational neuroscience community.

Availability

The software is released under GNU GPL v.3 at Google Code: http://code.google.com/p/pyeeg/. No commercial software is required to run PyEEG. Because Python is cross-platform, PyEEG can run on Linux, Mac OS, and Windows.

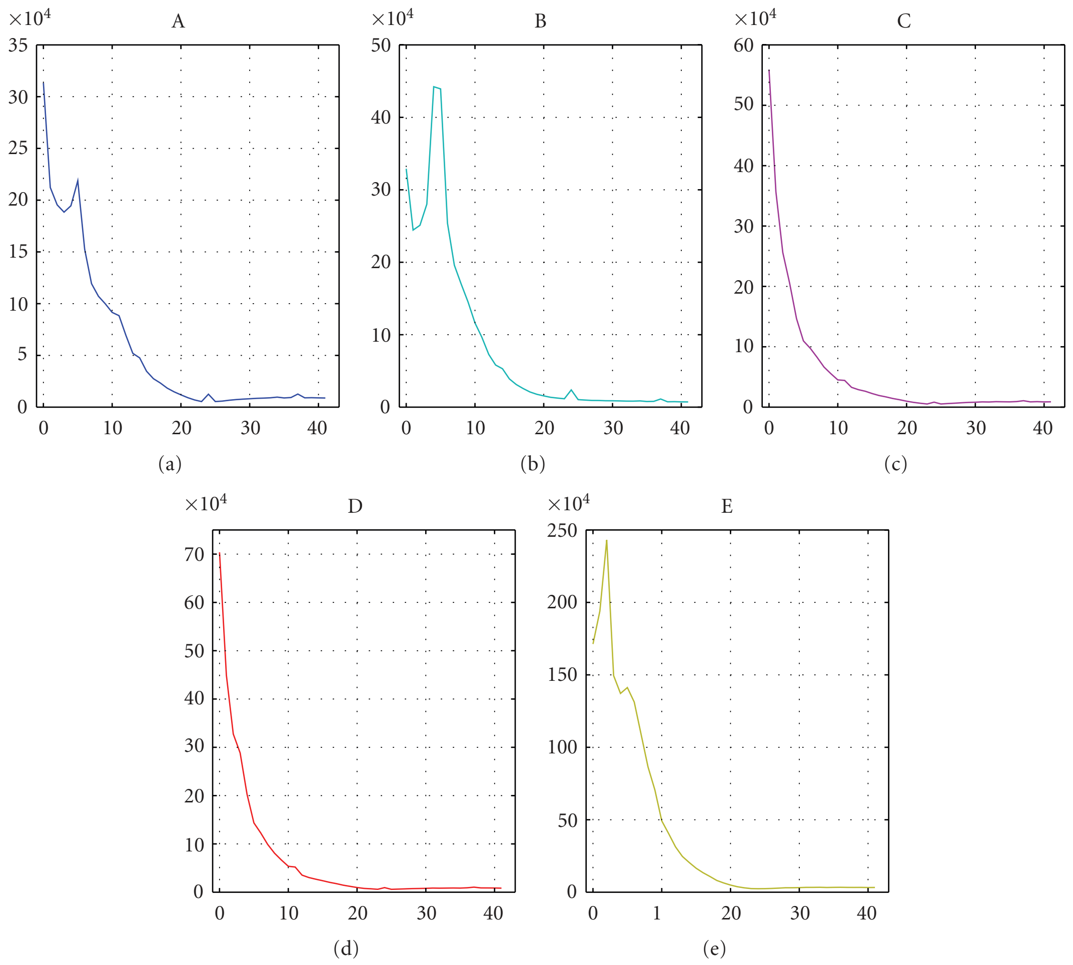

Figure 3: Average PSI of each set. Note that the scale in y-axis of set E is much larger than that of other sets.

References

[1] J. Dauwels, F. Vialatte, and A. Cichocki, "A comparative study of synchrony measures for the early detection of Alzheimer's disease based on EEG," in *Proceedings of the 14th International Conference on Neural Information Processing (ICONIP '07)*, vol. 4984 of *Lecture Notes in Computer Science*, pp. 112–125, 2008.

[2] A. A. Petrosian, D. V. Prokhorov, W. Lajara-Nanson, and R. B. Schiffer, "Recurrent neural network-based approach for early recognition of Alzheimer's disease in EEG," *Clinical Neurophysiology*, vol. 112, no. 8, pp. 1378–1387, 2001.

[3] F. S. Bao, D. Y. C. Lie, and Y. Zhang, "A new approach to automated epileptic diagnosis using EEG and probabilistic neural network," in *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '08)*, vol. 2, pp. 482–486, November 2008.

[4] J. Dauwels, E. Eskandar, and S. Cash, "Localization of seizure onset area from intracranial non-seizure EEG by exploiting locally enhanced synchrony," in *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '09)*, pp. 2180–2183, September 2009.

[5] A. B. Gardner, A. M. Krieger, G. Vachtsevanos, and B. Litt, "One-class novelty detection for seizure analysis from intracranial EEG," *Journal of Machine Learning Research*, vol. 7, pp. 1025–1044, 2006.

[6] C. W. Ko and H. W. Chung, "Automatic spike detection via an artificial neural network using raw EEG data: effects of data preparation and implications in the limitations of online recognition," *Clinical Neurophysiology*, vol. 111, no. 3, pp. 477–481, 2000.

[7] V. Srinivasan, C. Eswaran, and N. Sriraam, "Approximate entropy-based epileptic EEG detection using artificial neural networks," *IEEE Transactions on Information Technology in Biomedicine*, vol. 11, no. 3, pp. 288–295, 2007.

[8] F. S. Bao, J. M. Gao, J. Hu, D. Y. C. Lie, Y. Zhang, and K. J. Oommen, "Automated epilepsy diagnosis using interictal scalp EEG," in *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '09)*, pp. 6603–6607, September 2009.

[9] K. Lehnertz, F. Mormann, T. Kreuz et al., "Seizure prediction by nonlinear EEG analysis," *IEEE Engineering in Medicine and Biology Magazine*, vol. 22, no. 1, pp. 57–63, 2003.

[10] E. O'Sullivan-Greene, I. Mareels, D. Freestone, L. Kulhmann, and A. Burkitt, "A paradigm for epileptic seizure prediction using a coupled oscillator model of the brain," in *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '09)*, pp. 6428–6431, September 2009.

[11] F. S. Bao, Y.-L. Li, J.-M. Gao, and J. Hu, "Performance of dynamic features in classifying scalp epileptic interictal and normal EEG," in *Proceedings of 32nd International Conference of IEEE Engineering in Medicine and Biology Society (EMBC '10)*, 2010.

[12] R. Q. Quiroga, S. Blanco, O. A. Rosso, H. Garcia, and A. Rabinowicz, "Searching for hidden information with gabor transform in generalized tonic-clonic seizures," *Electroencephalography and Clinical Neurophysiology*, vol. 103, no. 4, pp. 434–439, 1997.

[13] A. Petrosian, "Kolmogorov complexity of finite sequences and recognition of different preictal EEG patterns," in *Proceedings of the 8th IEEE Symposium on Computer-Based Medical Systems*, pp. 212–217, June 1995.

[14] T. Higuchi, "Approach to an irregular time series on the basis of the fractal theory," *Physica D*, vol. 31, no. 2, pp. 277–283, 1988.

[15] B. Hjorth, "EEG analysis based on time domain properties," *Electroencephalography and Clinical Neurophysiology*, vol. 29, no. 3, pp. 306–310, 1970.

[16] T. Inouye, K. Shinosaki, H. Sakamoto et al., "Quantification of EEG irregularity by use of the entropy of the power spectrum," *Electroencephalography and Clinical Neurophysiology*, vol. 79, no. 3, pp. 204–210, 1991.

[17] S. J. Roberts, W. Penny, and I. Rezek, "Temporal and spatial complexity measures for electroencephalogram based brain-computer interfacing," *Medical and Biological Engineering and Computing*, vol. 37, no. 1, pp. 93–98, 1999.

[18] C. J. James and D. Lowe, "Extracting multisource brain activity from a single electromagnetic channel," *Artificial Intelligence in Medicine*, vol. 28, no. 1, pp. 89–104, 2003.

[19] S. M. Pincus, I. M. Gladstone, and R. A. Ehrenkranz, "A regularity statistic for medical data analysis," *Journal of Clinical Monitoring and Computing*, vol. 7, no. 4, pp. 335–345, 1991.

[20] C.-K. Peng, S. Havlin, H. E. Stanley, and A. L. Goldberger, "Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series," *Chaos*, vol. 5, no. 1, pp. 82–87, 1995.

[21] T. Balli and R. Palaniappan, "A combined linear & nonlinear approach for classification of epileptic EEG signals," in *Proceedings of the 4th International IEEE/EMBS Conference on Neural Engineering (NER '09)*, pp. 714–717, May 2009.

[22] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: dependence on recording region and brain state," *Physical Review E*, vol. 64, no. 6, Article ID 061907, 8 pages, 2001.