TIME – DEPENDENT ENTROPY

- **Introduction**

  Entropy of a signal is a technique used to quantify the amount of regularity and the unpredictability of fluctuations over time-series data. It is used to numerically quantify the amount of irregularity or fluctuations in a given signal. Numerous methods exist to calculate the entropy of a given time-series. Some of these have been already applied to EEG data in the past. The entropies used in this project are Shannon, Tsallis, Renyi and Permutation entropy.

- **Time Dependent Entropy**

  Time Dependent Entropy corresponds to the trend of entropy of a given signal over time. To get a signal's Time Dependent Entropy, the signal is first split into windows. This done using to parameters namely the window size (w) and step size (delta). The first window is from first to w^{th} value of the discrete-time signal while the second window starts from (1+delta) to (1+w+delta) values of the signal and so on. Thus number of windows(M) created are, considering length of signal as K:

  $$M= K-w/delta$$

  The time dependent entropy is then calculated using each of these windows and thus M such values are obtained. These M values form a time-series called as the time-dependent entropy of the given time-series.

  To find the entropy of a given window, various methods can be used. We employ some of these methods. First the minimum and maximum values of the signal in that window is noted. The amplitude interval between these two values is then divided into L intervals where L is a variable parameter. Then each of these slots/intervals are assigned a probability value. The probability value corresponds to the probability of the signal being in that interval which is equal to number of points in that interval divided by the total number of points in the interval. These probabilities are then used to calculate different entropies. We have used four such entropies:

  1. Shannon Entropy
  2. Tsallis Entropy
  3. Renyi Entropy
  4. Permutation Entropy

- **Theory**

  1. Shannon Entropy

     In development of foundations of classical information theory, a rigorous proof of Shannon's entropy based on additivity law was presented. Suppose a system is divided into two subsystems A and B, and p_{ij} (A, B) is joint probability of finding A in i^{th} state and B in j^{th} state then:

     $$p_{ij} (A, B) = p_i(A) * p_{ij}(B|A)$$

     Shannon's entropy of this system is defined as

     $$S (A, B) = -k_b\sum p_{ij} (A, B) * \ln (p_{ij} (A, B))$$

     Similarly, for a discrete time signal, the probabilities can be calculated as mentioned above. Let these values be p_i where i belongs to (1, L). The Shannon entropy of this window will be:

     $$Sh\_ent = -\sum p_i * \ln(p_i)$$

2. Tsallis Entropy and Renyi Entropy

   Tsallis Entropy and Renyi entropy are modification of the well-known Shannon entropy. Both of them are non-additive and have been used to deal with EEG or EEG type discrete type signals before. They too rely on the same probability measures as Shannon entropy but have an additional parameter each, which can be used to bring out signal characteristics that Shannon entropy cannot highlight. Tsallis entropy can be modified by using its q value whereas Renyi entropy has a parameter known as alpha. Their formulas are:

   $$Te\_ent = -\sum(p_i^q - p_i)$$

   $$RE\_ent = -\sum p_i^\alpha/(1-\alpha)$$

   Thus both entropies can be modified by changing their parameters. Tsallis entropy can be used to estimate Shannon entropy as they tend to it at q tends to zero.

3. Permutation Entropy

   Permutation Entropy is a measure for arbitrary time series based on analysis of permutation patterns. It is different from the above entropies as it depends only on the shape of the time series rather than the actual value or amplitude of each point in the series. It is characterized by two parameters namely d and delay. D decides the number of points considered for the permutation. d points lead to different number of permutations. Delay is defined as the delay between two consecutive series considered. For example, a delay of 1 will not skip any points, delay of two will skip one point and so on. Then each permutation is given a respective probability and this is then used to calculate the permutation entropy of the given time series. If the probability values are p_k then permutation entropy is:

   $$Per\_ent = -\sum p_k * \ln(p_k)$$

- **<u>Methods and Materials</u>**

  The data used for the project consisted of two datasets. The first dataset consisted of mental fatigue data. The second dataset consisted of mental workload data. Similar techniques were applied to both datasets to see and analyse the trends in the TDE of the given time series.

  For mental workload data, data collection was done from 30 subjects with two trials each. Each of these 60 datasets, after pre-processing, consisted of 24 channels and multiple 2s epochs with each epoch containing 500 data points. Thus we had 60 matrices of size 500xEx24. Each such matrix X had varying number of epochs represented by E. These 24 channels were then separated and the remaining matrix was then opened into a time series of length 500*E. Thus, 24 time-series were obtained for each subject per trial. Similarly, for the workload data consisted of 7 subjects with 512 data points per 2s epochs and 540 such epochs for each of the 62 channels. With the same approach as above, 62-time series were obtained per subject. From now on each of these channels or time-series will be denoted by C_h. Each such C_h was then stored in a separate text file so that it can be easily read by multiple python programs whenever required.

  After each such C_h was written into a text file, it was read by a python program and entropies were generated. Different parameters were used for the two different tasks and

those were selected which gave highest accuracy or the best trend. Each task is described in detail subsequently:

1.) Classification Task:

The classification task consisted of classifying the subject as fatigued or not fatigued. This was done using a support vector machine. The input had 24 features namely the entropies of the 24 channels. 300 points were selected from the initial and final parts of the EEG data respectively. It was assumed that he is not fatigued at the start and fatigued at the end. This was then fed to a svm classifier that did a 5-fold validation and returned average accuracy of the svm over these 5 validations. This was done for each trial and thus average accuracy was obtained. To improve accuracy, the same algorithm was applied over a range of C and gamma values for the svm kernel and hence maximum accuracy was obtained.

For the workload class a svm was used for multi-class classification. For this the decision function shape used was 'ovo'. Datasets were collected in a similar pattern as earlier. Fixed number of points were collected from each of the workload times and fed to this svm. The svm used one to one comparison to output results. Thus for 3 classes , it did 3 (${}^3C_2$) classifications. This was followed by the 5-fold validation and average accuracy was noted over all seven subjects.

The data was also filtered into multiple EEG bands but lower accuracies were obtained for each band as expected.

2.) Monitoring Task:

The goal of the monitoring task was to find and analyse trends in the time-dependent entropy of the different channels of a subject over time and see how it is related to the development mental fatigue in the subject. As the channels by themselves did not have a significant trend, multiple channels were used with different weights to get a significant trend.

This was done using a technique that give weights according to the trend in the channel. A linear fitting was done for each channel. The channels with highest modulus of this slope were selected. Each was given a weight equal to slope of that channel. These weights were first normalized by dividing by the sum of modulus of slopes of all the selected channels. This yielded a continuous rising trend for all the 30 subjects for both the trials.

The same technique is currently being applied to different bands of the data to check whether better results can be obtained.

- **Requirements:**

1.) Python3

2.) Git

- **Installation:**

❖ Windows:

1.) Get Anaconda Prompt Python 3.6 Version from this site. It will install all packages needed for this repository to work properly. Any other way to run

python is fine too, the only difference being you will have to install the missing libraries yourself.

2.) Get Git for Windows from this site. Configure git and open git bash in the directory you want this repository in.

3.) Type the following command:
git clone https://github.com/harshsd/NUS-Intern.git
Installation Done!

❖ Linux:

1.) Install git using
sudo apt-get install git

2.) Type the command:
git clone https://github.com/harshsd/NUS-Intern.git

3.) Open Terminal in home directory of repository and type:
pip3 install numpy scipy os
pip3 install –U scikit-learn
pip3 install pyqt5
pip3 install itertools
Installation Done!

- **Folders containing codes used for this project:**

1.) Entropy-related:
This folder contains codes related to calculating and storing entropies.

2.) SVM-related:
This folder contains codes used for classification task

3.) GUI-related:
This folder has all codes for running the GUI.

4.) Filters:
This folder has code for filters.

5.) Final_Codes_Folder:
This folder contains the final arranged and documented functions and codes to be used by a user who is using this repository. Use the files in this folder to carry out your work. To use import files into your python program and use accordingly. More details regarding this folder is available in the next section.

- **Files containing all functions used to generate and analyse TDE:**

1.) entropy.py

i.) sig_entropy:
This function is used to find the time-dependent entropy of an entire time series with varying entropy parameters and types. It takes the discrete time signal and all the parameters as an input and outputs the TDE as a list. It also takes parametric inputs for the different types of entropies with a string that decides which entropy to calculate.
Arguments:

L: Number of partitions of amplitude
w: Width of the moving window
delta: Step size of the window
sig: Time series whose entropy is to be found
entropy_name: Name of entropy. Currently Shannon / renyi / tsallis / permutation are available
q: q for tsallis. Only if entropy is tsallis
alpha: alpha for renyi
d, delay: d, delay for permutation

Returns:

entropy: An array with the entropy values

ii.) partition_entropy:

This function returns a single entropy value while taking as input a complete or part of a discrete-time signal and other parametric inputs. It considers the entire signal to be divided into one partition and hence returns only one value. This function is used by the sig_entropy function.

Arguments:

partition: The signal/partition whose entropy is to be found
L: Number of partitions of amplitude
w: Width of the moving window
delta: Step size of the window
entropy_name: Name of entropy. Currently Shannon / renyi / tsallis / permutation are available
q: q for tsallis. Only if entropy is tsallis
alpha: alpha for renyi
d, delay: d, delay for permutation

Returns:

per_ent: A single float value containing the entropy the signal

iii.) P_m_l:

This function calculates the probability that a signal is present in one specific bin of the amplitude interval being considered. As discussed before the amplitude interval is defined as interval between minimum and maximum values of the signal. This is then split into different partitions. This function finds probability of signal being in one of these partitions.

Arguments:

partition: input signal
k: Defines the part of the partition. Should be between 0 and L-1
slot_height: Height of each slot/part of the partition
maxp, minp: Maximum and Minimum values in the signal

Return:

Single float value containing probability of signal being in that slot of the partition

iv.) shannon_entropy:

This function returns a single value containing Shannon entropy of the given signal.

Arguments:

L: Number of partitions of amplitude

partition: Time series whose entropy is to be found

slot_height: Height of each slot/part of the partition

maxp, minp: Maximum and Minimum values in the signal

Returns:

sh_ent: Float value containing Shannon entropy

v.) Tsallis entropy:

This function returns Tsallis entropy of given signal. Note: q value of tsallis should be none zero. If you require entropy at q tending to zero, use the Shannon entropy function.

Arguments:

L: Number of partitions of amplitude

partition: Time series whose entropy is to be found

slot_height: Height of each slot/part of the partition

maxp, minp: Maximum and Minimum values in the signal

q: q parameter for tsallis entropy

Returns:

Float value containing tsallis entropy

vi.) renyi_entropy:

This function returns Renyi entropy of a given signal. Note: alpha value for renyi entropy must not be equal to 1. For alpha equal to 1, use Shannon entropy instead as renyi entropy tends to Shannon entropy when alpha tends to 1.

L: Number of partitions of amplitude

partition: Time series whose entropy is to be found

slot_height: Height of each slot/part of the partition

maxp, minp: Maximum and Minimum values in the signal

alpha: alpha parameter for renyi entropy

Returns:

Float value containing renyi entropy

vii.) permutation_entropy:

This function returns permutation entropy of a given signal.

Arguments:

time_series: Time series for analysis

m: Order of permutation entropy

delay: Time delay

Returns:

Vector containing Permutation Entropy

2.) read_write.py

The entire project was done by taking inputs from and writing outputs to respective text files. All inputs and outputs were of array format and hence were stored in '.txt' files with each new line containing a new in order element of the array. This file has two important functions namely the reading and writing functions to the files.

i.) read_file:

This function is used to read an array from a locally stored file. Before this, ensure the file has been given appropriate permissions and is in same directory as the working directory. If not, change the working directory by using the *os.chdir()* function.

Arguments:

file_name: Name of file.

Returns:

vec: Vector with values as in file

ii.) write_to_file:

This function is used to write an array to a particular file in the working directory. If this file is not present, then it creates a new file with the given name. If it is present it appends the current file. To change this, change the parameter of line 25 from 'a+' to 'w' or 'w+'.

Arguments:

file_name: Name of file.

vector_output: The vector to be written to the file

3.) process_and_plot.py:

This file is used in the monitoring task of mental fatigue or mental workload monitoring. It contains functions used for analysing the change in entropies over time.

i.) rolling_mean:

This is used to calculate rolling_mean of any discrete time signal. If length of input is l, then this returns an array of length l-n.

Arguments:

input_sig: List/array whose mean is to be calculated

n: The window size to be considered while calculating rolling mean. Should be greater than 0 and less than or equal to the length of signal

Returns:

Array with rolling mean of input signal

ii.) sorted_array_with_indices:

This is an optional function. This was used only to see which channels had a more prominent trend than the others. This function takes an array as an input and returns a sorted array along with another array denoting the original positions of the sorted elements in the new array. This function uses a copy of the passed array and hence can be used on an array which will be required later.

Arguments:

b = input_array

Returns:

Tuple with first element as sorted array and second containing array with indices of original positions

4.) svm.py

This file consists of code related to the classification task. It consists of only one function which performs the 5-fold classification over a dataset and return mean accuracy of the svm over that data. The details are as follows:

i.) accuracy_svm:

Arguments:

data_set1: The first data_set. All the inputs should have same number of features. Also each of them should belong to the same class.

data_set2: The second data_set. All inputs should have same number of features. Each should belong to same class other than that of data_set1. Also, length of the dataset must be same as the previous data_set.

cc: C for kernel

gg: gamma for kernel

Returns:

Tuple containing mean accuracy of selected svm and standard deviation of the included accuracies.

5.) filter.py

This file also contains a single function that takes a time series and frequencies as input and outputs the filtered results.

i.) butter_filter:

This functions filters the given input signal in the frequency range provided in the arguments and returns the filtered data.

Arguments:

input_sig: Input Signal
sample_freq: Sampling frequency of the input signal
lowf: Lower cut-off of the filter
highf: Higher cut-off of the filter
order: order of the filter to be used

Returns:

An array with the filtered signal

6.) gui.py

This file contains all codes used to make the GUI required for time dependent entropy estimations. This GUI can be used to edit the parameters of the TDE and also plot multiple entities like the EEG signal, a part of it, its TDE or TDE of a specific part of that signal.
For more information, look into the file or follow this reference:
https://pythonspot.com/pyqt5