Pseudo Codes for calculating features from blocks of samples of plant electrical signals

By

Shre Kumar Chatterjee,

Dr Koushik Maharatna,

University of Southampton

April 2014

**Introduction**

The following figure shows the major blocks which needs to be implemented on hardware in order to replicate the design of classification scheme analysed so far.

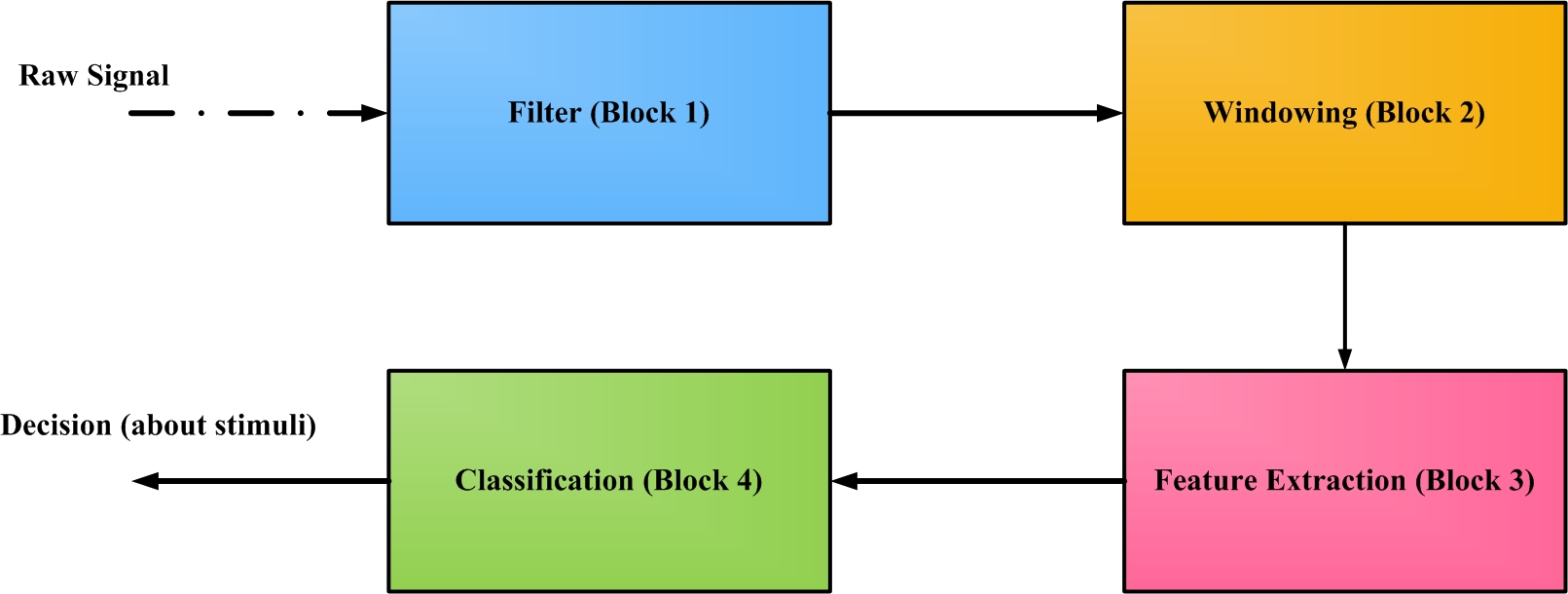


Figure : Block diagram showing classification scheme

**Block 1: Filter**

Through our exploration, we found a 6th order, *Chebyshev type II* filter provided the best optimization of the cost function using a cut-off frequency of 0.77Hz and stop-band ripple of 100. This filter parameter were used to on the raw signals.

**Block 2: Windowing**

Windowing length is 1024 samples. Therefor the samples needs to be divided into non-overlapping blocks of 1024 samples.

**Block 3: Feature Extraction**

The following are the features which needs to be extracted from the filtered signals.

|  |  |  |
| --- | --- | --- |
| **Descriptive Statistics** | | |
| **Feature** | **Description** | **Formula** |
| Mean | Average of *N* samples. | **=** |
| Variance | The spread of the observation from the mean. | **=** |
| Kurtosis | Defined as the measure of how flat or tapered the distribution is. |  |
| Interquartile range | The median divides a set of numbers into two halves. The median of the lower half, when subtracted from the median of the upper half, gives the interquartile range. | * Median = value. * Lower quartile (Q1) = value * Upper quartile (Q3) = value   Inter quartile range (IQR) = (Q1 - Q3) |

**Hurst Exponent**

Calculating the Hurst Exponent

* First the data is broken into small chunks (say of size N, with some minimum limit – usually 16).
* The mean is then computed for each chunk.
* This mean is subtracted from all the elements in the chunk thereby leaving a zero mean series in each chunk.
* Thereafter the *zero mean values* in the chunk are summed up (cumulative summation) and its range is calculated (the distance between the minimum and maximum values).
* The standard deviation of the original values is calculated next.
* The range (of the mean centred values) is divided by the standard deviation (of the original values) to get the new *rescaled* range. This needs to be calculated for all the chunks.
* Thereafter the rescaled range is averaged over all the chunks.
* This averaged rescaled range value and the chunk size follows a power law and the exponent of this power law gives the Hurst Exponent.

**Detrended Fluctuation Analysis (DFA)**

To compute DFA of a time series, we do the following:

* Divide the signal into segments of size 
* Calculate the sample deviations from the mean to obtain the integrated signal, given as  
* To this new series, a polynomial function is fit to the segment of size for all 
* We define,  

The average of the square of the deviations of the integrated signal from this trend over all such segments is the value of the DFA, for each segment of size.

**Hjorth’s parameters**

The Hjorth mobility and complexity, described in , quantify a signal from its mean slope and curvature by using the variances of the deflection of the curve and the variances of their first and second derivatives. Let the signal amplitudes at discrete time instants be at time. The measures of the complexity of the signal is based on the second moments in time domain of the signal and the signal’s first and second derivatives. The finite differences of the signal or time derivatives can be viewed in .

, where and

, where 

The variances are then computed as [2].

, , 

These variances are used to calculate the Hjorth mobility () and the Hjorth complexity () as shown in .

and  

**Wavelet Packed Entropy (Wentropy)**

The wavelet (Shannon) entropy gives an estimate of the measure of information of the probability distributions. This is calculated by converting the squared absolute values of the wavelet coefficients  of the *ith* wavelet decomposition level as shown in.



**Feature Normalization**

All extracted features were normalized using the following formula:



where is the feature value, and are the minimum and maximum values of the feature vector respectively.

**Block 4: Classification**

A multi-class classification setting is designed on One Versus One (OVO) configuration, as shown in the figure below:

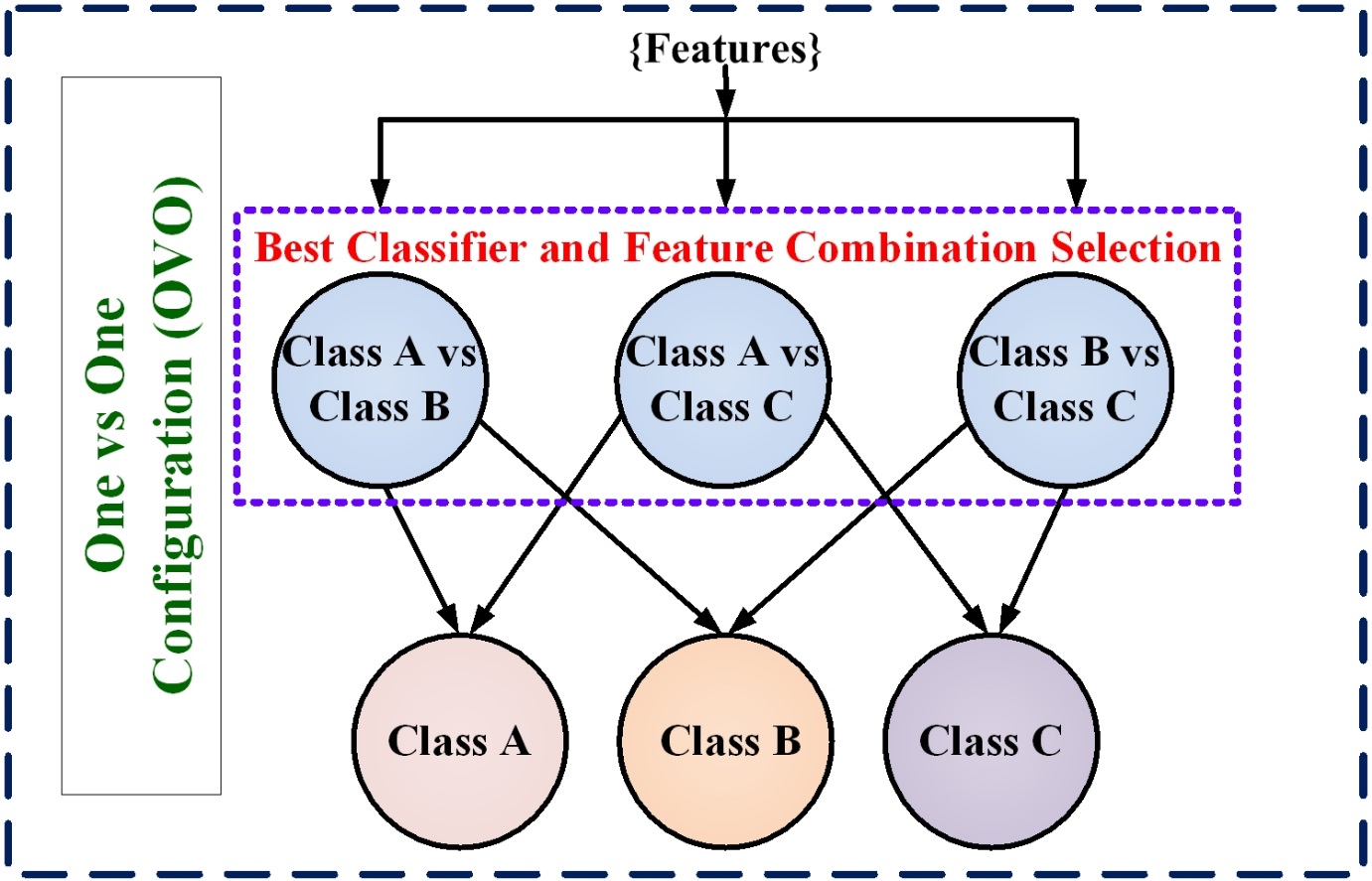


Figure : One Versus One classification scheme

Thus three classifiers are required, to produce a decision on which stimuli is affecting the plants. Our exploration is showing that all three classifiers could be Linear Discriminant Analysis (LDA) classifier, producing an accuracy of 75% during prospective study (un-known dataset).

**Appendix**

***Matlab codes***

**Hurst Exponent**

clc

n=length(xV);

d=[];

len=[];

y=xV;

y2=cumsum(y-mean(y),1); %cumulative sum of mean centred values

d(1)=mean(range(y2)./std(y,1)); %first element of the rescaled range vector d

i=1; %initialised i with 1

len(1)=size(y,1);

while size(y,1)>=16 %minimum number of rows is set to be 16

i=i+1; %increment of i starts from here

rnr=floor(n/(2^(i-1))); %length of number of rows for each chunk is obtained from here

rnc=floor(n/rnr); %length of number of columns for each chunk is obtained from here

y=reshape(xV(1:rnr\*rnc),rnr,rnc); %distributing the elements of the original data vector into a rnr x rnc matrix

n2=size(y,1); %number of rows in the new matrix,

pop=find(std(y,1)~=0); % a)Computes std. dev, normalized by N instead of N-1, b) Searches columns of the reshaped matrix for non-zero std. dev values

if ~isempty(pop) % For non-zero std. dev values, proceed as below

y=y(:,pop); % Use the columns which produced non-zero std. dev. values

len(i)=n2;

y2=cumsum(y-kron(mean(y),ones(n2,1)),1); % a) by using the kron function, 1xg dimensional mean vector becomes n2 x g dimensional

d(i)= mean(range(y2)./std(y,1)); %rescaling the new range and taking its average

else %for zero values of std. dev

len(i) = NaN;

d(i)=NaN;

end

end

dlog=log2(d(end:-1:1));

blog=log2(len(end:-1:1));

pone=polyfit(blog,dlog,1); %finding the co-efficients of a one dimensional polynomial which is fitted

Hurst=pone(1);

**DFA**

pord=1; % Polynomial order

d=[];

n=length(xV); %xV is data provided,

y=cumsum(xV-mean(xV)); %cumulative sum of mean centred values

p1=polyfit([1:n]',y,pord); %fitting a 1st order polynomial to the mean centred values and obtaining the co-efficients, the coefficients are sorted in descending order

er=y-polyval(p1,[1:n]'); % polyval finds the estimated values at each point x(1,2,..n)

d(1)=sqrt(er'\*er/n);% resulting in a scalar

i=1;

y2=y;

len=[];

len(1)=n;

while size(y2,1)>=16 %minimum number of rows is set to be 16

i=i+1;%increment of i starts from here

rnr=floor(n/(2^(i-1)));%length of number of rows for each chunk is obtained from here

rnc=floor(n/rnr);%length of number of columns for each chunk is obtained from here

y2=reshape(y(1:rnr\*rnc),rnr,rnc);%distributing the elements of the original data vector into a rnr x rnc matrix

pro=NaN\*ones(rnr\*rnc,1);%a column vector of length rnr x rnc is created, filled up with NaN's

for j=1:rnc

p1=polyfit([1:rnr]',y2(:,j),pord);%fitting a 1st order polynomial

pro(rnr\*(j-1)+1:rnr\*j)=polyval(p1,[1:rnr]'); %subsequently filling up the column vector with the estimated values (using the co-efficients from the previous step)

end

er=y(1:rnr\*rnc)-pro; %computing the error between actual and estimated values

d(i)=sqrt(er'\*er/(rnr\*rnc)); %scalar error value

len(i)=rnr; %length of each row of the reshaped matrix

end

dlog=log2(d(end:-1:1));

blog=log2(len(end:-1:1));

p=polyfit(blog,dlog,1); %fitting a polynomial to 'length vs error'

dfa=p(1);

**Hjorth Mobility and Hjorth Complexity**

function [mob\_1,mob\_2] = hjorth\_mobility\_complexity(data\_matrix)

xV = data\_matrix;

n = length(data\_matrix(:,1));

xV(2:n+1,:) = data\_matrix;

xV(1,:)=0;

dxV = diff(xV); %diff is used to compute difference between subsequent elements

ddxV = diff(dxV);

mx2 = mean(xV.^2); % removing any negative element by squaring

mdx2 = mean(dxV.^2); % removing any negative element by squaring

mddx2 = mean(ddxV.^2); % removing any negative element by squaring

mob = mdx2./mx2;

mobility = transpose(sqrt(mob));

complexity = sqrt((mddx2./mdx2) - mob);

hcomp = transpose(complexity);

[1] B. Hjorth, “EEG analysis based on time domain properties,” *Electroencephalography and clinical neurophysiology*, vol. 29, no. 3, pp. 306–310, 1970.

[2] B. Hjorth, “Time domain descriptors and their relation to a particular model for generation of EEG activity,” *CEAN-Computerized EEG analysis*, pp. 3–8, 1975.