Automatic Seizure Detection Using Frequency and Entropy Clustering

Alexandre Gagné

**Introduction**

Seizures are the result of random electrical discharge in the brain. They can cause the loss of consciousness and motor control. It can be very dangerous in certain situations such as when operating a motor vehicle. Not only is it dangerous to the person driving, but also those around them. Being able to detect an oncoming seizure would allow for people with epilepsy to operate motor vehicles safely.

The methods used in the algorithm follow several previous works as described in *Automatic Epileptic Seizure Detection Using Scalp EEG and Advanced Artificial Intelligence techniques.* The paper has compared multiple methods. They focus on the extracted features and classification models. The classifiers used are machine learning algorithms.This paper uses some of the indicated methods.

**Methods**

The MATLAB code consists of three main parts: preprocessing, feature extraction and classification. The data is first processed to remove unwanted noise. In feature extraction, we observed time and frequency features in a moving window. Using a supervised machine learning algorithm, it is able to generate seizure detection markers on new data.

*Preprocessing*

Before the algorithm is able to preprocess the signal, the signal should be conditioned. From literature, it was found that the most common low frequency cutoff was 0.5 Hz. This is because activity below this threshold is often motion or electrical activity [1]. The commonly used high frequency cutoffs varied. 33 Hz was used in the design of the filter because EEG instruments rarely exceed 30-40 Hz [1]. With a passband between 0.5-33 Hz, the butterworth filter also removes powerline interference. A butterworth filter was chosen because of the maximally flat passband. It does not contain ripples in the passband like the Chebyshev filter. The cutoff however is not as sharp as the Chebyshev.

Figure 1: Frequency of filter

Synchronized averaging is a technique for removing random noise in the time domain (text). Averaging the 23 EEG channels was expected to remove any random noise and help isolate the seizure epoch. By averaging the channels, it also allows for the use of all of the data.

Figure 3: Filtered and unfiltered plots of same channel

Figure 2: Multiple channel plots of patient

*Feature Extraction*

The features are extracted from the averaged signal by using a moving rectangular window. Each of the signals is padded on both ends. The MATLAB function ‘padarray’ was used to add half the window size with replicate data. The replicate data is used instead of zero padding so that there is no roundoff at the beginning and end of the feature extraction. However, this probably isn’t very necessary since none of the seizures occur in the first or last several windows.

The features used are from both the time and frequency domains. The power spectral density (PSD) of each window was estimated using the ‘pwelch’ function. Peak frequency is the most prominent frequency found in the PSD of the window. It is a feature that is focused on in many detection methods. This is because the EEG has a major cyclic component during a seizure. This shows up as a large peak in the PSD [1]. The mean frequency of the PSD was also found to be useful for classifying seizure events. It provides a reference for the peak frequency of the window. Entropy is used to measure the complexity of a signal. It is a good estimator of seizure events because brain activity becomes more predictable during a seizure. The complexity of the signal is reflected by a drop in the signal entropy [1].

*Classification*

|  |  |
| --- | --- |
| **Number**  **of Neighbours** | **Accuracy** |
| 1 | 97.46 % |
| 2 | 96.28 % |
| 3 | 98.55 % |
| 4 | 98.67 % |
| 5 | 98.75 % |
| 6 | 98.55 % |

The extracted features are put into a vector where each row consists of the three features for the given window number. Using the feature vector, we used the K-nearest neighbours (KNN) function to train a model which predicts class labels of new data. KNN is a supervised learning classifier. It looks at the ‘n’ closest points and assigns the label based on the majority class. It is an effective clustering algorithm. To prepare the data for training, an 80% holdout method was used. The holdout method uses 80% of the data training and the remaining 20% is used for testing. Using the testing data, we adjusted the number of nearest neighbours. As we increased the number of neighbours, the accuracy trended upwards when the number of neighbours increased. It was found that odd numbers of closest neighbours gave higher accuracies. This is expected since there is no possibility for a tie with an odd number of points.

Table 1: Number of neighbours selection

This method of classification was used because it is a supervised algorithm. Supervised algorithms tend to be more effective at classification because they train the data using the ground truth labels. Scatter plots of the feature vector showed two distinct clusters. The large dense cluster appeared to be the non-seizure event cluster and the smaller cluster represented seizure events. An original attempt at using the unsupervised K means classification was unsuccessful. The K means method requires that the user defines a set number of clusters. Realistically there are more than one cluster that represents either seizure or non-seizure events.

*Validation Strategy*

The new validation data undergoes the same process as the training data. It is preprocessed, the features are extracted and then the seizure markers are classified. The algorithm has a validation function that breaks up the detected seizure marker and the ground truth vector into windows using a matrix reshape. The vector lengths are rounded up to the nearest factor of the window length. Zeros are used to pad the signal because they do not affect the validation process. Each window, or column of the reshape, is set to 1 if there is a seizure is detected within that window. Otherwise, it is left as a 0. Comparing the two vectors using a confusion matrix provides the parameters for the performance metrics.

**Results**

From the plots of the filtered and synchronized averaged signals, it is apparent that averaging the channels increased the amount of noise in the signal. It had a negative effect on the preprocessing of the algorithm. While it did help to avoid channel selection, it proved to distort the signal further.

Figure 4: Synchronized averaged channels

The test results were obtained from the partition test data. The holdout data is set aside at random by MATLAB and is used for testing model parameters. Each window size and overlap was run 5 times and averaged. A larger sample size would have been ideal but time consuming.

Table 2: Validation results for different window and overlap sizes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Window (overlap)** | **Accuracy** | **F1 Score** | **Specificity** | **Sensitivity** | **Precision** |
| 4 s (25%) | 98.5504 | 0.4197 | 99.9653 | 0.2151 | 24.1666 |
| 2 s (25%) | 98.6173 | 0.57802 | 99.9801 | 0.29614 | 17.3628 |
| 2 s (12.5 %) | 98.6277 | 1.51804 | 99.974 | 0.77968 | 28.9846 |

*Table 2* summarizes the results. The rest of the data can be found in the *Appendix*. Since the event is mostly non-zero, certain performance metrics are better indicators of the algorithms performance. The focus should be on the number of true positives and false negatives. It was expected that a larger window size would lower the F1 score because the frequency of peaks tends to decay with time [1]. A smaller window also provides better time resolution. It was found that the smaller window gave the best results. By increasing the overlap, it also helps to improve the time resolution of the classification. While the smaller window size and overlap drastically improve the F1 score, it also has a large impact on the speed of the program. The feature extraction is extremely slow. The code runs with a 2 second window and 25% overlap to help speed it up. From the output, the seizure detection algorithm managed to detect several windows with a seizure label that align with the ground truth.

Figure 5: Final output of EEG with GT and EEG with Auto Marker

Normalizing the data typically helps improve clustering classifiers, however an attempt at normalizing the data reduced the performance of the model. Normalizing is used to reduce the weighting or pull one of the features may have. In this case, it seems that the dominant feature actually helps to serve as the primary feature.

**Discussion and Conclusion**

The two main areas of improvement needed for the automatic seizure detection program would be the preprocessing and overall speed of the program. The synchronized averaging really distorted the signal and is not an ideal method for using all 23 channels. If you do not need to re-train the model every time you run the code, it is a quick process. The feature extraction for the model training is very slow because it uses long nested for loops. Using matrix arithmetic to shift the window would be a better solution because MATLAB handles these processes well.

The use of different and more features could also significantly improve the classification. The mean frequency feature serves the algorithm better if the signal is broken up into the different EEG frequency bands.

The performance metrics of the code are very good if compared to the competition results. A 2 second window with 12.5% overlap resulted in a F1 score over 1%. However, this testing was performed on a random partition of the data. It would be interesting to see how it does with new samples.

**References**

[1] Fergus, P., Hignett, D., Hussain, A., Al-Jumeily, D., & Abdel-Aziz, K. (2015). Automatic epileptic seizure detection using scalp EEG and advanced artificial intelligence techniques.*BioMed Research International,*doi:http://dx.doi.org.ezproxy.lib.ryerson.ca/10.1155/2015/986736

[2] R. M. Rangayyan, “Filtering for removal of artifacts”, Biomedical Signal Analysis, 2nd ed. 2005, ch. 3, sec. 3.5, pp. 143-147.

**Appendix**

*Performance Metrics*

*Window 1024 Shift 256*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Accuracy** | **F1 Score** | **Specificity** | **Sensitivity** | **Precision** |
| 98.54 | 0.6873 | 100 | 0.3448 | 100 |
| 98.5896 | 0 | 99.9642 | 0 | 0 |
| 98.6802 | 0.7576 | 99.9643 | 0.3906 | 12.5 |
| 98.4687 | 0.6536 | 99.9438 | 0.3401 | 8.333 |
| 98.4737 | 0 | 99.9540 | 0 | 0 |
| **98.5504** | **0.4197** | **99.9653** | **0.2151** | **24.1666** |

*Window 512 Shift 128*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Accuracy** | **F1 Score** | **Specificity** | **Sensitivity** | **Precision** |
| 98.5869 | 0.7080 | 99.9745 | 0.3617 | 16.6667 |
| 98.5920 | 0.3565 | 99.9821 | 0.1808 | 12.5 |
| 98.6247 | 0.3650 | 99.9898 | 0.1842 | 20 |
| 98.6424 | 1.1009 | 99.9643 | 0.5682 | 17.6471 |
| 98.6406 | 0.3597 | 99.9898 | 0.1858 | 20 |
| **98.6173** | **0.57802** | **99.9801** | **0.29614** | **17.3628** |

*Window 512 Shift 64*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Accuracy** | **F1 Score** | **Specificity** | **Sensitivity** | **Precision** |
| 98.6171 | 1.0811 | 99.9783 | 0.5520 | 26.087 |
| 98.6398 | 1.6393 | 99.9732 | 0.8427 | 30 |
| 98.6070 | 1.6014 | 99.9732 | 0.8227 | 30 |
| 98.6209 | 1.7937 | 99.9719 | 0.9234 | 31.250 |
| 98.6536 | 1.4747 | 99.9732 | 0.7576 | 27.5862 |
| **98.6277** | **1.51804** | **99.974** | **0.77968** | **28.9846** |

*Code*

window = 512;

sampleRate = 256;

Mdl = feature\_extract();

%load('Mdl.mat');

name = 'subject019.mat'; % Subject

load(['EEG\_',name]); % Load EEG data

load(['seizureGT\_',name]); % Load Seizure Ground truth data

test = mean([EEG.ch].'); % averages channels of test data

[seizureMarker\_auto] = classification(test,Mdl); % classifies using trained model

performanceMetrics = validation(seizureMarker\_auto, seizureGT); % results

%% plot auto marker and ground truth

seizure\_window = zeros(1,length(seizureGT));

seizure = find(seizureMarker\_auto); % finds seizure locations

% expand marker to entire window

for j = 1 : length(seizure)

% makes entire window 1

seizure\_window(seizure(j)\*window:((seizure(j)+1)\*window)-1) = 1;

end

t = (0:1:(length(EEG(1).ch)-1))/sampleRate; % create time variable

subplot(2,1,1);

plot(t,EEG(10).ch);

hold on;

plot(t,seizureGT\*1000);

axis tight;

xlabel('Time (s)');

legend('EEG Channel 10','Ground Truth');

subplot(2,1,2);

plot(t,EEG(10).ch);

hold on;

plot(t,seizure\_window\*1000);

axis tight;

xlabel('Time (s)');

legend('EEG Channel 10','Auto Marker');

%% plot 3 channels of 1 patient to show similarity

figure;

subplot(3,1,1);

plot(t,EEG(10).ch);

axis tight;

legend('Channel 10');

subplot(3,1,2);

plot(t,EEG(14).ch);

axis tight;

legend('Channel 14');

subplot(3,1,3);

plot(t,EEG(6).ch);

axis tight;

legend('Channel 6');

xlabel('Time (s)');

function Mdl = feature\_extract

sampleRate = 256; % sampling rate (needed for time conversion)

window = 512;

slide = 128;

filtered{20,24} = [];

eeg{20,2} = [];

features{4,21} = [];

for j = 0 : 20

load(['EEG\_subject0',num2str(j,'%02i'),'.mat']);

load(['seizureGT\_subject0',num2str(j,'%02i'),'.mat']);

for i = 0 : 22

buffer = EEG(i+1).ch;

wn=[0.5 33]/(sampleRate/3);

[b,a] = butter(2,wn,'bandpass');

filtered{j+1,i+1} = filter(b,a,buffer);

end

filtered{j+1,24} = seizureGT;

end

for j = 0 : 20

eeg{j+1,1} = mean([filtered{j+1,1:23}].');

eeg{j+1,2} = filtered{j+1,24};

end

%% moving window and pad signal

for k = 0 : 20

buffer = [eeg{k+1,1}].'; % load signal and GT into vectors

GT = [eeg{k+1,2}];

% pads the start and end of the array with copy of signal

padded = padarray(buffer,ceil(window/2),'replicate');

GT = padarray(GT,ceil(window/2),'replicate');

% number of windows used to span signal

steps = (length(padded)-window)/slide;

% allocate feature arrays

peak\_f = zeros(1,ceil(steps));

avg\_f = zeros(1,ceil(steps));

entropy = zeros(1,ceil(steps));

indicator = zeros(1,ceil(steps));

j=1; % window number

%% FEATURE EXTRACTION

for i = 0 : slide : (steps\*slide)

% PEAK FREQUENCY

[psd,f] = pwelch(padded(i+1:i+window),[],[],[],sampleRate); % psd of window

[~,loc] = max(psd); % finds largest peak in the psd

peak\_f(j) = f(loc); % peak frequency

% MEAN FREQUENCY

avg\_f(j) = meanfreq(psd,sampleRate); % mean frequency of psd

% ENTROPY

p = hist(padded(i+1:i+window)); % probability distribution

p = p/sum(p); % calculate probabilities

entropy(j) = -sum(p.\*log2(p)); % entropy of window

% INDICATOR

indicator(j) = ceil(mean(GT(i+1:i+window)));

j = j + 1; % window number

end

% put features in cell array

features{1,k+1} = peak\_f.';

features{2,k+1} = avg\_f.';

features{3,k+1} = entropy.';

features{4,k+1} = indicator.';

end

% Plot frequency response of filter

t = (0:1:(length(EEG(1).ch)-1))/sampleRate; % create time variable

figure;

freqz(b, a, 512, sampleRate);

title('Frequency Response of Butterworth filter');

% Plot filteref/unfiltered channel

figure;

subplot(2,1,1);

plot(t,EEG(10).ch); % plots unfiltered channel

legend('Unfiltered channel');

axis tight;

subplot(2,1,2);

plot(t,filtered{21,11}); % plots filtered channel

legend('Filtered channel');

xlabel('Time (s)');

axis tight;

% Plot averaged signal

figure;

plot(t,eeg{10,1}); % plot synchronized average

axis tight;

xlabel('Time (s)');

%% TRAIN KNN MODEL

% combine features of each subject in columns

peak\_f = vertcat(features{1,:}); % combines features in single column

avg\_f = vertcat(features{2,:});

entropy = vertcat(features{3,:});

indicator = vertcat(features{4,:});

% create feature array by concatenating features horizontally

data = horzcat(peak\_f,avg\_f,entropy);

% Create data partitions (80% training, 20% testing)

cvp = cvpartition(length(data),'Holdout',0.2);

Xtrain = data(training(cvp),:); % Training set indices

Ytrain = indicator(training(cvp),:);

Xtest = data(test(cvp),:); % Test set indices

Ytest = indicator(test(cvp),:);

% train KNN model using training set

Mdl = fitcknn(Xtrain,Ytrain,'NumNeighbors',5);

% Test model

labels = predict(Mdl,Xtest); % model generated labels on test set

C = confusionmat(Ytest,labels); % confusion matrix of test set

accuracy = (C(1,1)+C(2,2))/sum(sum(C))\*100

F1score = 200\*C(2,2)/(2\*C(2,2)+C(2,1)+C(1,2))

specificity = 100\*C(1,1)/(C(1,1)+C(1,2))

sensitivity = 100\*C(2,2)/(C(2,2)+C(2,1))

precision = 100\*C(2,2)/(C(2,2)+C(1,2))

end

function [test\_window] = classification(test,Mdl)

sampleRate = 256;

window = 512;

slide = 128;

% filter

wn=[0.5 33]/(sampleRate/3);

[b,a] = butter(2,wn,'bandpass');

test = filter(b,a,test);

% pads the start and end of the array with copy of signal

test = padarray(test.',ceil(window/2),'replicate');

% number of windows used to span signal

steps = (length(test)-window)/slide;

% allocate feature arrays

peak\_f = zeros(1,ceil(steps));

avg\_f = zeros(1,ceil(steps));

entropy = zeros(1,ceil(steps));

j = 1;

for i = 0 : slide : (steps\*slide)

% PEAK FREQUENCY

[psd,f] = pwelch(test(i+1:i+window),[],[],[],sampleRate); % psd of window

[~,loc] = max(psd); % finds largest peak in the psd

peak\_f(j) = f(loc); % peak frequency

% MEAN FREQUENCY

avg\_f(j) = meanfreq(psd,sampleRate); % mean frequency of psd

% ENTROPY

p = hist(test(i+1:i+window)); % probability distribution

p = p/sum(p); % calculate probabilities

entropy(j) = -sum(p.\*log2(p)); % entropy of window

j = j + 1; % window number

end

%% CLASSIFICATION

% predicted class array

class = predict(Mdl,horzcat(peak\_f.',avg\_f.',entropy.'));

% fit to window size

a = zeros(1,round((length(class))/(window/slide))\*(window/slide));

a(1:length(class)) = class.';

% Divide GT and predicted labels into windows (each column = 1 window)

test\_window = reshape(a,(window/slide),[]);

% Determine label for entire window

test\_window = ceil(mean(test\_window));

end

function performanceMetrics = validation(seizureMarker\_auto, seizureGT)

window = 512;

% fit to window size

GT = zeros(1,ceil(length(seizureGT)/window)\*window);

GT(1:length(seizureGT)) = seizureGT.';

% Divide GT and predicted labels into windows (each column = 1 window)

GT\_window = reshape(GT,window,[]);

% Determine label for entire window

GT\_window = ceil(mean(GT\_window));

C = confusionmat(GT\_window,seizureMarker\_auto(1:length(GT\_window)));

performanceMetrics = 2\*C(1,1)/(2\*C(1,1)+C(1,2)+C(2,1)); % F1 score

accuracy = (C(1,1)+C(2,2))/sum(sum(C))\*100;

specificity = 100\*C(1,1)/(C(1,1)+C(1,2));

sensitivity = 100\*C(2,2)/(C(2,2)+C(2,1));

precision = 100\*C(2,2)/(C(2,2)+C(1,2));

end