Signals and Systems Project

Student Name: Iman Alizadeh Fakouri

Student Number: 401102134

Instructor: Hamid Karbalaei Aghajan

July 1, 2024

1 Introduction

This report details the process of analyzing EEG signals to predict epileptic seizures. The steps involve feature extraction, feature selection, and classification using Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) classifiers. Performance measures such as sensitivity, specificity, and latency are calculated to evaluate the effectiveness of the classifiers.

2 Load Database

In this part we will load seven edf files and then convert them into a matrix. Then we will extract 14,10 minute intervals from them.

```
%part1
dataPaths = {
    'C:\Users\imanf\Downloads\WOS1\chb01_02.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_03.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_04.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_15.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_18.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_16.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_26.edf'
};
%converting all data into a matrix
EEGData = cell(1, length(dataPaths));
for i = 1:length(dataPaths)
    EEGData{i} = edfread(dataPaths{i});
end
numericData = cell(1, length(EEGData));
for i = 1:length(EEGData)
    if istimetable(EEGData{i})
        numericData{i} = table2array(EEGData{i});
    else
        numericData{i} = EEGData{i};
    end
end
% converting each second which has 256 elements as a vector
   because of
%sampeling frequency into 256 main matrix elements
for i = 1:length(numericData)
    data = numericData{i};
    numSamples = size(data, 1) * 256;
    numChannels = size(data, 2);
```

```
    concatenatedData = zeros(numSamples, numChannels);
    for j = 1:size(data, 1)
        startIdx = (j-1)*256 + 1;
        endIdx = j*256;
        concatenatedData(startIdx:endIdx, :) = cell2mat(data(j,
            :));
    end
    numericData{i} = concatenatedData;
end



%Extracting 14 epochs with 10 minute length accroding to
    instructions
%and datasets seizure times
data = numericData{1};
intervalLength = 10 * 60 * 256;
A1 = data(1:intervalLength, :);
A2 = data(intervalLength+1:2*intervalLength, :);
A3 = data(2*intervalLength+1:3*intervalLength, :);
A4 = data(3*intervalLength+1:4*intervalLength, :);
A5 = data(4*intervalLength+1:5*intervalLength, :);
A6 = data(5*intervalLength+1:6*intervalLength, :);
data = numericData{2}; % 'chb01_03.edf'
seizureStart = 2996 * 256; % Seizure start time in samples
seizureEnd = 3036 * 256; % Seizure end time in samples
B1 = data(seizureStart-10*60*256:seizureStart-1, :);

data = numericData{3}; % 'chb01_04.edf'
seizureStart = 1467 * 256; % Seizure start time in samples
seizureEnd = 1494 * 256; % Seizure end time in samples
B2 = data(seizureEnd:seizureEnd+10*60*256-1, :);

data = numericData{2}; % 'chb01_03.edf'
C1 = data(seizureStart-5*60*256:seizureStart+5*60*256-1, :);

data = numericData{3}; % 'chb01_04.edf'
C2 = data(seizureStart-5*60*256:seizureStart+5*60*256-1, :);

data = numericData{4}; % 'chb01_15.edf'
seizureStart = 1732 * 256; % Seizure start time in samples
seizureEnd = 1772 * 256; % Seizure end time in samples
C3 = data(seizureStart-5*60*256:seizureStart+5*60*256-1, :);
```

```
data = numericData{6}; % 'chb01_16.edf'
seizureStart = 1015 * 256; % Seizure start time in samples
seizureEnd = 1066 * 256; % Seizure end time in samples
C4 = data(seizureStart-5*60*256:seizureStart+5*60*256-1, :);

data = numericData{5}; % 'chb01_18.edf'
seizureStart = 1720 * 256; % Seizure start time in samples
seizureEnd = 1810 * 256; % Seizure end time in samples
C5 = data(seizureStart-5*60*256:seizureStart+5*60*256-1, :);

data = numericData{7}; % 'chb01_26.edf'
seizureStart = 1720 * 256;
seizureEnd = 1810 * 256;
C6 = data(seizureStart-5*60*256:seizureStart+5*60*256-1, :);
```

3 Calculating Power Spectral Density (PSD)

3.1 Explanation

The Power Spectral Density (PSD) shows how the average power of a signal is distributed across different frequencies. It is a crucial measure in signal processing, providing insights into the frequency components of the signal and their respective power levels. The PSD can be calculated using direct calculation methods or autocorrelation methods. The formula for PSD calculation is:

$$S_x(f) = \lim_{T \to \infty} E\left[\frac{1}{2T}\left|\int_{-T}^{T} x(t)e^{-j2\pi ft}dt\right|^2\right] \tag{1}$$

Where:

- $S_x(f)$ is the PSD of the signal $x(t)$.
- T is the time period.
- E denotes the expected value.
- f is the frequency.

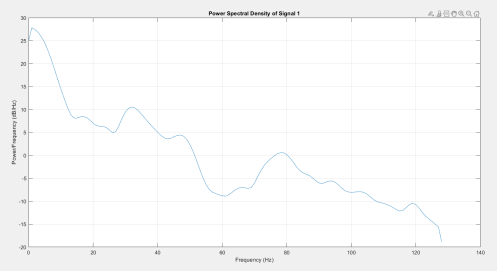

Figure 1: Plot of PSD of first dataset(without seizure)

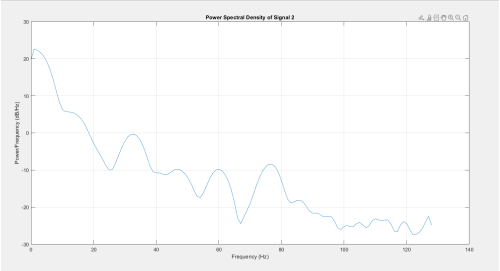

Figure 2: Plot of PSD of second dataset(with seizure)

3.2 Code Explanation

Here is the code for this part:

```
dataPaths = {
    'C:\Users\imanf\Downloads\WOS1\chb01_02.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_03.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_04.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_15.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_18.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_16.edf',
    'C:\Users\imanf\Downloads\WS1\chb01_26.edf'
};

EEGData = cell(1, length(dataPaths));
for i = 1:length(dataPaths)
    EEGData{i} = edfread(dataPaths{i});
end
numericData = cell(1, length(EEGData));
for i = 1:length(EEGData)
    if istimetable(EEGData{i})
        numericData{i} = EEGData{i}.Variables;
    else
        numericData{i} = EEGData{i};
    end
end
f = cell(1, length(EEGData));
PSD = cell(1, length(EEGData));
for i = 1:length(EEGData)

    [pxx, freq] = pwelch(numericData{i}, [], [], [], 256);
    PSD{i} = 10*log10(pxx);
    f{i} = freq;
disp(['Signal ' num2str(i) ' has unique PSD values: ' num2str(
   length(unique(PSD{i})))]);

end

for i = 1:length(PSD)
    figure;
    plot(f{i}, PSD{i});
    grid on;
    title(sprintf('Power Spectral Density of Signal %d', i));
    xlabel('Frequency (Hz)');
    ylabel('Power/Frequency (dB/Hz)');
end
```

The provided code calculates the PSD for EEG signals stored in multiple `.edf` files and plots the results.The code starts by defining the paths to the `.edf` files containing EEG data.It reads the data from each file into a cell array `EEGData` using the `edfread` function.The code checks if the data is in a timetable format and extracts the numeric data accordingly into the `numericData` cell array. For each EEG signal, the code uses the `pwelch` function to calculate the PSD. This function estimates the PSD using Welch's method, which involves segmenting the signal, computing a modified periodogram for each segment, and averaging the periodograms. The PSD values are converted to decibels (dB) using the formula $10 \log_{10}(pxx)$.

This approach provides a clear visualization of how the power of the EEG signals is distributed across different frequencies, which is crucial for analyzing and interpreting the data.

4 Calculating Shannon Entropy

4.1 Explanation

Entropy measures the randomness or disorder within a signal. Shannon entropy, in particular, quantifies this randomness by treating the Power Spectral Density (PSD) values as probability density functions. The formula for Shannon Entropy is:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{2}$$

Where:

- $H(X)$ is the Shannon Entropy.
- $p(x_i)$ is the probability of occurrence of x_i.

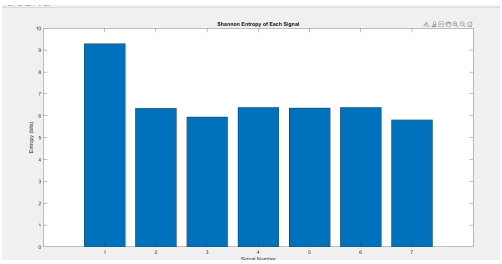

Figure 3: Plot of Shannon Entropy of all datasets

```
Shannon Entropy of Signal 1: 9.29786+65.6659i
Shannon Entropy of Signal 2: 6.3369+0.63946i
Shannon Entropy of Signal 3: 5.9348+1.2947i
Shannon Entropy of Signal 4: 6.3713+0.98486i
Shannon Entropy of Signal 5: 6.3574+0.34005i
Shannon Entropy of Signal 6: 6.3794+0.49641i
Shannon Entropy of Signal 7: 5.8022+2.4142i
```

Figure 4: Value of Shannon Entropy of all datasets

4.2 Code Explanation

The provided code calculates the Shannon Entropy for each EEG signal based on their PSD values and plots the results.Here is the code:

```
entropyValues = cell(1, length(PSD));

for i = 1:length(PSD)
    probabilities = PSD{i} / sum(PSD{i});
    entropyValues{i} = -sum(probabilities .* log2(probabilities)
        );
    disp(['Shannon Entropy of Signal ' num2str(i) ': ' num2str(
        entropyValues{i})]);
end
figure;
bar(cell2mat(entropyValues));
title('Shannon Entropy of Each Signal');
xlabel('Signal Number');
ylabel('Entropy (bits)');
```

For each signal, the code calculates the probabilities by normalizing the PSD values. This is done by dividing each PSD value by the sum of all PSD values for that signal. The code then calculates the Shannon Entropy for each signal using the formula $H(X) = -\sum p(x_i) \log_2 p(x_i)$. This involves summing the product of each probability and its logarithm (base 2). This approach provides a quantitative measure of the randomness or complexity of the EEG signals, which can be useful for various analyses and interpretations.

5 Feature Extraction

5.1 Explanation

Each 10-minute interval prior to a seizure is broken into 16-second epochs using a moving window. For each epoch, calculate the Fast Fourier Transform (FFT), then compute the Power Spectral Density (PSD) and normalize it. Calculate Shannon entropy and other statistical measures (average, standard deviation, minimum, and maximum) for each epoch.

5.2 Related Code

In code below we first extract 37 epochs which are 16 second intervals from each 10 minute epoch.Then we compute 5 features(mean-std-entropy-max-min-psd)for each epoch of every channel.Finally we handle nan values and reshape our matrix into 14*4255 matrix.

```
%part4
%Extracting 16 second epochs(37 epochs)from each one of 14 10
    minute
%intervals
matrices = {A1, A2, A3, A4, A5, B1, C1, C2, C3, C4,A6 ,B2,C5, C6
    };
epochLength = 16 * 256;
stepSize = epochLength;
features = cell(1, length(matrices));
for i = 1:length(matrices)
    signal = matrices{i};
    signalLength = size(signal, 1);
    numEpochs = floor(signalLength / stepSize);
    if numEpochs > 37
        numEpochs = 37;
    end
    features{i} = cell(numEpochs, size(signal, 2));

    %Extracting 5 features for each channel of 16 second epochs
    for j = 1:numEpochs
        startIdx = (j-1)*stepSize + 1;
        endIdx = startIdx + stepSize - 1;
        if endIdx > signalLength
            endIdx = signalLength;
        end
        epoch = signal(startIdx:endIdx, :);

        if iscell(epoch)
            epoch = cell2mat(epoch);
        end
        for k = 1:size(epoch, 2)
            channelEpoch = epoch(:, k);
            [pxx, freq] = pwelch(channelEpoch, [], [], [], 256);
            psd = 10*log10(pxx);
            psd = psd / sum(psd);
            entropy = -sum(psd .* log2(psd));
            meanVal = mean(channelEpoch);
            stdVal = std(channelEpoch);
```

```
            minVal = min(channelEpoch);
            maxVal = max(channelEpoch);
            features{i}{j, k} = [meanVal, stdVal, minVal, maxVal
                , entropy];
        end
    end
end

%Handeling possible nan values in features matrix
for i = 1:length(features)
    for j = 1:size(features{i}, 1)
        for k = 1:size(features{i}, 2)
            if iscell(features{i}{j, k})
                if all(cellfun(@isnumeric, features{i}{j, k}))
                    features{i}{j, k} = cell2mat(features{i}{j,
                        k});
                else
                    features{i}{j, k} = NaN;
                end
            end
        end
    end
end

%converting features from a 14*23*37*5 matrix into a
% 14*4255 matrix
vectorizedFeatures = zeros(length(features), 5 * 37 * 23);
for i = 1:length(features)
    featureVector = [];
    for j = 1:size(features{i}, 1)
        for k = 1:size(features{i}, 2)
            if isnumeric(features{i}{j, k})
                featureVector = [featureVector, features{i}{j, k
                    }];
            end
        end
    end
    vectorizedFeatures(i, :) = featureVector;
end
```

6 Feature Selection

6.1 Explanation

Apply a one-sample t-test to the extracted features and select those with a p-value less than 0.001. The test statistic is calculated as:

$$t = \frac{\bar{X} - \mu_0}{s/\sqrt{n}} \tag{3}$$

where $\bar{X}$ is the mean of the features, μ_0 is the hypothesized mean (considered as zero), s is the standard deviation, and n is the size of the feature vector.

6.2 Related Code

The code below performs feature selection using the one-sample t-test.In this code i used only train data for feature selection.I used ttest2 for feature selection.

```
%part 5
%We only use train dataset for feature selection
trainMatrix = vectorizedFeatures(1:6, :);%Non seizure data
testMatrix = vectorizedFeatures(7:10, :);%Seizure data
%Feature extraction using p-value
selectedFeatures = [];
pValues = [];
for i = 1:size(trainMatrix, 2)
    [h, p, ci, stats] = ttest2(trainMatrix(:, i), testMatrix(:,
        i), 'Alpha', 0.001);
    if p < 0.001
        selectedFeatures = [selectedFeatures, i];
        pValues = [pValues, p];
    end
end

disp('Selected features:');
disp(selectedFeatures);
disp('P-values of selected features:');
disp(pValues);
selectedIndices = selectedFeatures;

%Updating train and test dataset so they have values
%only related to selected features
mySelectedFeatures = zeros(length(matrices), length(
    selectedIndices));
```

```
for i = 1:length(matrices)
    featureVector = vectorizedFeatures(i, :);
    mySelectedFeatures(i, :) = featureVector(selectedIndices);
end
```

7 Support Vector Machine (SVM) Classifier

7.1 Explanation

An SVM classifier is used to distinguish between seizure and non-seizure epochs. Use a linear kernel function and employ a least squares method to find the separating hyperplane. The SVM classifier aims to maximize the margin between different classes.

7.2 Related Code

The code below trains and tests an SVM classifier.I used tic and toc for latency measurment.

```
%part 6
%Labeling dataset(-1 for no seizure and 1 for seizure data)
labels = [-1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1];
%We have 10 train data according to project document
X_train = real(mySelectedFeatures(1:10, :));
X_test = real(mySelectedFeatures(11:14, :));
Y_train = labels(1:10);
Y_test = labels(11:14);

%I used tic and toc for latency measurment
tic;
%training SVM model by parameters explaind in document
svmModel = fitcsvm(X_train, Y_train, 'KernelFunction', 'linear',
    'Standardize', true);
Y_pred = predict(svmModel, X_test);
svm_latency = toc;
%Computing accuracy of model
numCorrect=0
for i=1:length(Y_test)
    if Y_pred(i)==Y_test(i)
        numCorrect=numCorrect+1;
    end
end
```

```
accuracy=numCorrect/4*100;
disp(['Accuracy of SVM classifier: ', num2str(accuracy), '%']);
disp(['Latency of SVM classifier: ', num2str(svm_latency), '
    seconds']);
```

8 K-Nearest Neighbor (KNN) Classifier

8.1 Task

Train and test a KNN classifier.

8.2 Explanation

KNN is a non-parametric method that classifies each object based on the majority vote of its neighbors. Use the Minkowski distance metric for measuring the distance between data points.

8.3 Related Code

The code below trains and tests a KNN classifier.

```
%part 7
%k=2 was best for my data and KNN model
k = 2 ;
tic;
%training KNN model by paramethers explaind in document
knnModel = fitcknn(X_train, Y_train, 'NumNeighbors', k, '
    Distance', 'minkowski');
Y_pred_knn= predict(knnModel, X_test);
knn_latency = toc;
%Computing accuracy of model
numCorrect_knn =0;
for i=1:length(Y_test)
    if Y_pred_knn(i)==Y_test(i)
        numCorrect_knn=numCorrect_knn+1;
    end
end
accuracy_knn=numCorrect_knn/4*100;
disp(['Accuracy of KNN classifier: ', num2str(accuracy_knn), '%'
    ]);
disp(['Latency of KNN classifier: ', num2str(knn_latency), '
    seconds']);
```

9 Performance Measures

9.1 Explanation

Calculate sensitivity, specificity, and latency to assess the performance of the seizure prediction algorithm. Sensitivity measures the true positive rate, and specificity measures the true negative rate.

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4}$$

$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} \tag{5}$$

9.2 Related Code

The code below calculates the performance measures for the classifiers.

```
%part 8
truePositives_svm = 0;
falsePositives_svm = 0;
trueNegatives_svm = 0;
falseNegatives_svm = 0;
truePositives_knn = 0;
falsePositives_knn = 0;
trueNegatives_knn = 0;
falseNegatives_knn = 0;

%Iterating in Y-pred and Y-test and Y-pred-knn
%in order to compute number of TP,FM,TN,FP predictins
%in each model
for i = 1:length(Y_test)
    if Y_pred(i) == 1 && Y_test(i) == 1
        truePositives_svm = truePositives_svm + 1;
    elseif Y_pred(i) == 1 && Y_test(i) == -1
        falsePositives_svm = falsePositives_svm + 1;
    elseif Y_pred(i) == -1 && Y_test(i) == -1
        trueNegatives_svm = trueNegatives_svm + 1;
    elseif Y_pred(i) == -1 && Y_test(i) == 1
        falseNegatives_svm = falseNegatives_svm + 1;
    end
    if Y_pred_knn(i) == 1 && Y_test(i) == 1
        truePositives_knn = truePositives_knn + 1;
    elseif Y_pred_knn(i) == 1 && Y_test(i) == -1
```

```
        falsePositives_knn = falsePositives_knn + 1;
    elseif Y_pred_knn(i) == -1 && Y_test(i) == -1
        trueNegatives_knn = trueNegatives_knn + 1;
    elseif Y_pred_knn(i) == -1 && Y_test(i) == 1
        falseNegatives_knn = falseNegatives_knn + 1;
    end
end
%Computing specifity and sensivity according to formula
sensitivity_svm = truePositives_svm / (truePositives_svm +
    falseNegatives_svm);
specificity_svm = trueNegatives_svm / (trueNegatives_svm +
    falsePositives_svm);
sensitivity_knn = truePositives_knn / (truePositives_knn +
    falseNegatives_knn);
specificity_knn = trueNegatives_knn / (trueNegatives_knn +
    falsePositives_knn);
disp(['Sensitivity of SVM classifier: ', num2str(sensitivity_svm
    )]);
disp(['Specificity of SVM classifier: ', num2str(specificity_svm
    )]);
disp(['Sensitivity of KNN classifier: ', num2str(sensitivity_knn
    )]);
disp(['Specificity of KNN classifier: ', num2str(specificity_knn
    )]);
```

10 Leave-One-Out Cross-Validation

10.1 Explanation

For more detailed evaluation, use leave-one-out cross-validation, where each sample is used once as a test sample while the rest are used for training.

10.2 Related Code

```
%Calculating leave one out cross validation
X = real(mySelectedFeatures);
Y = labels;
accuracies_svm = zeros(length(labels), 1);
accuracies_knn = zeros(length(labels), 1);

for i = 1:length(labels)
    X_train = X([1:i-1, i+1:end], :);
```

```
    Y_train = labels([1:i-1, i+1:end]);
    X_test = X(i, :);
    Y_test = labels(i);
    svmModel = fitcsvm(X_train, Y_train, 'KernelFunction', '
        linear', 'Standardize', true);
    Y_pred_svm = predict(svmModel, X_test);
    accuracies_svm(i) = (Y_pred_svm == Y_test);

    k = 2;
    knnModel = fitcknn(X_train, Y_train, 'NumNeighbors', k, '
        Distance', 'minkowski');
    Y_pred_knn = predict(knnModel, X_test);
    accuracies_knn(i) = (Y_pred_knn == Y_test);
end
average_accuracy_svm = mean(accuracies_svm) * 100;
average_accuracy_knn = mean(accuracies_knn) * 100;

disp(['Average accuracy of SVM classifier using LOOCV: ',
    num2str(average_accuracy_svm), '%']);
disp(['Average accuracy of KNN classifier using LOOCV: ',
    num2str(average_accuracy_knn), '%']);
```

11 Output Analysis

11.1 Output

The output of code was like this:
Accuracy of SVM classifier: 75 percent
Latency of SVM classifier: 0.082638 seconds
Accuracy of KNN classifier: 100 percent
Latency of KNN classifier: 0.073827 seconds
Sensitivity of SVM classifier: 1
Specificity of SVM classifier: 0.5
Sensitivity of KNN classifier: 1
Specificity of KNN classifier: 1
Average accuracy of SVM classifier using LOOCV: 78.5714 percent
Average accuracy of KNN classifier using LOOCV: 92.8571 percent

11.2 Analysis

The KNN model had a better performance than SVM model.The best k for KNN model was 2. Sensitivity of both models was really good but SVM model didn't have a good specificity.