Seizure Detection

By: Rutva Patel

Introduction

The motivation behind this seizure detection project stems from the urgent need to improve the quality of life and safety of individuals with epilepsy through timely, accurate, and accessible interventions. Seizures are often unpredictable and can lead to serious physical harm or disruption to daily activities; therefore, creating a system capable of automatically detecting and forecasting seizures from EEG data offers a proactive approach to patient care. By leveraging machine learning techniques to analyze brainwave patterns, this project aims to provide clinicians and patients with earlier warnings, enabling preemptive actions such as medication administration, caregiver notification, or preventive measures. Ultimately, this work aspires to bridge the gap between raw neurological data and actionable insights, reducing uncertainty and empowering individuals with epilepsy to live with greater confidence and independence.

From a technical perspective, this project integrates multiple machine learning models—including logistic regression, random forests, and feed-forward neural networks—to classify seizure and non-seizure states based on EEG data. The data pipeline involves crucial preprocessing steps such as data cleaning, noise reduction, and handling class imbalance to ensure high-quality inputs. Additionally, feature extraction and feature mining techniques are applied to capture the most informative signal characteristics, with signal transformations like Fourier and wavelet transforms used in some cases to analyze the frequency and time-frequency domains of EEG recordings. By combining these engineered features with diverse modeling approaches, the project aims to build a robust, generalizable system that can effectively detect and predict seizures across varying patient profiles and datasets.

Datasets

For testing purposes, we will be using three publicly available seizure datasets: the Bonn dataset from the University of Bonn, the CHB-MIT dataset from the Massachusetts Institute of Technology, and the Zenodo dataset from the Zenodo open-access repository. The Bonn dataset contains EEG data that is already prefiltered. In this context, prefiltered means that the signals have been processed to remove noise and artifacts, such as baseline drift, muscle artifacts, and powerline interference, resulting in clean signals that are ready for direct analysis and model input. In contrast, the CHB-MIT and Zenodo datasets contain raw, unfiltered EEG recordings that include various types of noise, such as muscle movement artifacts, eye blinks, electrode displacement, and 60 Hz powerline interference. Therefore, it is necessary to apply signal filtering techniques to these datasets to remove unwanted noise and artifacts before running our models, ensuring that the extracted features reflect true neural activity rather than noise-related distortions.

Dataset Sources:
Bonn University: https://www.ukbonn.de/epileptologie/arbeitsgruppen/ag-lehnertz-neurophysik/downloads/
CHB-MIT Dataset: https://physionet.org/content/chbmit/1.0.0/
Zenodo: https://zenodo.org/records/2547147#.Y7eU5uxBwlI

Seizure Detection vs. Prediction: Concepts and Dataset Suitability

Seizure detection refers to the process of identifying the occurrence of a seizure at or very close to the moment it happens, by analyzing ongoing EEG signals in real time or retrospectively. It aims to classify EEG segments as "seizure" or "non-seizure" based on characteristic patterns observed during the ictal (seizure) phase. In contrast, **seizure prediction** focuses on forecasting an upcoming seizure before it occurs, typically by analyzing EEG data during the preictal phase (the period leading up to a seizure) to distinguish it from interictal (between seizures) or baseline activity. Prediction requires identifying subtle changes in brain activity that precede seizures, which is more challenging and demands time-labeled preictal data.

Different types of datasets are required for these purposes. For seizure detection, datasets must contain well-annotated ictal events along with non-seizure segments, ideally with clear start and end times of seizures to enable accurate model training and evaluation. For seizure prediction, the dataset must include not only ictal and interictal data but also explicitly labeled preictal segments leading up to seizures, allowing the model to learn predictive features.

In our case, the Bonn dataset contains EEG segments that are pre-classified as either seizure (ictal), interictal, or normal; however, it lacks continuous recordings and preictal labels, making it more suitable for seizure detection rather than prediction. The CHB-MIT dataset and Zenodo dataset both provide long-term continuous EEG recordings with precise seizure onset and offset annotations, allowing us to extract both ictal and preictal windows. Therefore, CHB-MIT and Zenodo datasets can be used for both seizure detection and seizure prediction tasks, depending on how we segment and label the data for model training.

Filtering the Data

Libraries required: To implement the filtering process, we use the **MNE-Python** library for signal processing and filtering functions, along with **NumPy** for numerical operations and data manipulation.

To prepare the raw EEG data(edf files) from CHB-MIT and Zenodo for analysis, we apply two stages of filtering: a bandpass filter and a notch filter. The bandpass filter is used to retain frequencies within a specific range (commonly 0.5–40 Hz), effectively removing low-frequency drifts and high-frequency noise that are not relevant to brain activity associated with seizures. The notch filter, typically set at 50 Hz or 60 Hz depending on the recording environment, is applied to eliminate powerline interference commonly present in EEG signals. These filtering steps are crucial for enhancing the signal-to-noise ratio, ensuring that the features extracted from the data accurately represent underlying neural patterns rather than artifacts or external electrical noise. By improving signal quality, filtering increases the reliability and performance of machine learning models trained on the data.

Seizure Detection

With the appropriate data for seizure detection successfully loaded, we will proceed to test various models to evaluate their performance and determine which yields the best results.

We will our model two times in the following ways:

1) Testing models on prefiltered data: Bonn Dataset
2) Testing models on raw data after preprocessing: CHB-MIT and Zenodo Datasets

Proposed models for analysis:

1) Feed Forward Neural Nets

Libraries*:* Scikit-learn, TensorFlow (Keras), NumPy, Matplotlib

Layers:
Input Layer: input_shape=(X.shape[1],)
Accepts an input vector with a length equal to the number of features in the dataset.

1st Hidden Layer: Dense(64, activation='relu')
A fully connected layer with 64 neurons using ReLU activation to capture non-linear patterns in the data.

1st Dropout Layer: Dropout(0.3)
Randomly deactivates 30% of neurons during training to reduce overfitting and improve generalization.

2nd Hidden Layer: Dense(32, activation='relu')
A fully connected layer with 32 neurons, again using ReLU to refine learned representations.

2nd Dropout Layer: Dropout(0.3)
Applies 30% dropout to further regularize the model and prevent overfitting.

Output Layer: Dense(1, activation='sigmoid')
Outputs a probability between 0 and 1, representing the likelihood of a seizure event.

Model Compilation:
-Optimized using the Adam optimizer, which adaptively adjusts learning rates for efficient training.
-Uses binary cross-entropy as the loss function, appropriate for binary classification tasks.
-Tracks accuracy as the evaluation metric during training and testing.

Training and Testing:
-Trained for 30 epochs with a batch size of 16 samples.
-Uses 10% of the training data as a validation set to monitor overfitting and generalization.
- Reports test accuracy to assess the model's performance on unseen test data.

2) Logistic Regression

Libraries: PyWavelets, Scikit-learn, NumPy, Matplotlib

a) Logistic Regression Model Using Frequency-Domain Features (Fourier Transform)

In the first approach, we transformed EEG signals from the time domain into the frequency domain using the Fourier Transform. This process converts the EEG waveform into its constituent frequency components, allowing the model to capture frequency-based patterns associated with seizures. We computed the absolute magnitude of the Fourier coefficients to represent the energy at each frequency, retaining only the first half of the spectrum due to its symmetry in real-valued signals.

The frequency-domain features were then standardized using z-score normalization via the StandardScaler to ensure all features contributed equally to the model. After scaling, the data was split into training and testing sets with 80% used for training and 20% for testing, stratifying to maintain class balance.

A logistic regression model was trained on the frequency-domain features, using a maximum of 1000 iterations to ensure convergence. Predictions were made on the test set, and model performance was evaluated using accuracy, a classification report (precision, recall, F1-score), and a confusion matrix visualized with a heatmap. This method aimed to explore whether seizure-related patterns could be captured solely through frequency information.

b) Logistic Regression Model Using Wavelet-Transformed Features

In the second approach, we extracted features from EEG signals using the discrete wavelet transform (DWT), implemented with the PyWavelets (pywt) library. Wavelet decomposition enables simultaneous analysis in both time and frequency domains, providing a multi-resolution representation of EEG signals. For each EEG signal, wavelet decomposition was performed up to four levels using the Daubechies 4 (db4) wavelet, a commonly used wavelet for EEG signal analysis. The resulting wavelet coefficients from all levels were flattened into a single feature vector for each sample.

These wavelet-transformed features were similarly standardized using z-score normalization to account for scale differences among coefficients. The data was split into training and testing sets, maintaining class balance as before. A logistic regression model with 1000 maximum iterations was trained on the wavelet features. Model predictions were evaluated using accuracy, a detailed classification report, and a confusion matrix visualized as a heatmap.

3) Random Forest

Libraries: Scikit-learn, NumPy, Matplotlib, Scipy

a) Random Forest Model Using Frequency-Domain Features (Fourier Transform)

In this method, the EEG signals were first transformed from the time domain to the frequency domain using the Fast Fourier Transform (FFT). The absolute magnitudes of the Fourier coefficients were extracted to represent the spectral power at each frequency, effectively capturing seizure-related patterns in the frequency spectrum. Due to the symmetry of real-valued signals, only the first half of the frequency components was retained for analysis.

The frequency-domain features were optionally scaled using z-score normalization with the StandardScaler, though scaling is not strictly necessary for Random Forests. The dataset was then split into training and testing sets with an 80:20 ratio, stratifying to preserve class balance.

A Random Forest classifier with 100 decision trees was trained on the frequency-domain features. The model made predictions on the test set, and performance was evaluated using accuracy, a classification report (precision, recall, F1-score), and a confusion matrix visualized via a heatmap.

To further interpret the model, feature importance scores were extracted from the trained Random Forest. These scores indicate which frequency components were most influential in distinguishing seizure from non-seizure events. A bar chart of feature importance across frequency indices was plotted to highlight informative frequency bands.

b) Random Forest Model Using Raw EEG Time-Domain Features

In the second approach, the Random Forest classifier was trained directly on the raw EEG time-domain data without any transformation into the frequency or time-frequency domains. The input features consisted of the raw signal amplitudes at each sampled timepoint.

Similar to the first method, the time-domain features were optionally standardized using z-score normalization. The dataset was again split into an 80% training set and 20% testing set with stratified sampling to maintain class distribution.

A Random Forest classifier with 100 decision trees was trained on the time-domain features. Predictions on the test set were evaluated using accuracy, a classification report, and a confusion matrix visualized as a heatmap to summarize correct and incorrect classifications.

Feature importance scores were extracted to determine which specific timepoints contributed most to the model's decision-making. These importance values were visualized using a bar plot, providing insight into the temporal regions of the EEG signal that were most indicative of seizure activity.

*******This approach doesn't work for CHB-MIT and Zenodo dataset as they are very big datasets, so we propose another solution for that******

Another b) Random Forest Model Using Statistical Features

In this approach, we extracted seven statistical features from each EEG signal segment: mean, standard deviation, minimum, maximum, root mean square (RMS), kurtosis, and skewness. These features summarize key properties of the signal's distribution and shape, reducing dimensionality while retaining relevant information for classification.

To address class imbalance, we down-sampled the majority class (non-seizure) to achieve a 5:1 ratio of non-seizure to seizure samples. The features were standardized using z-score normalization, and the data was split into 80% training and 20% testing sets with stratified sampling.

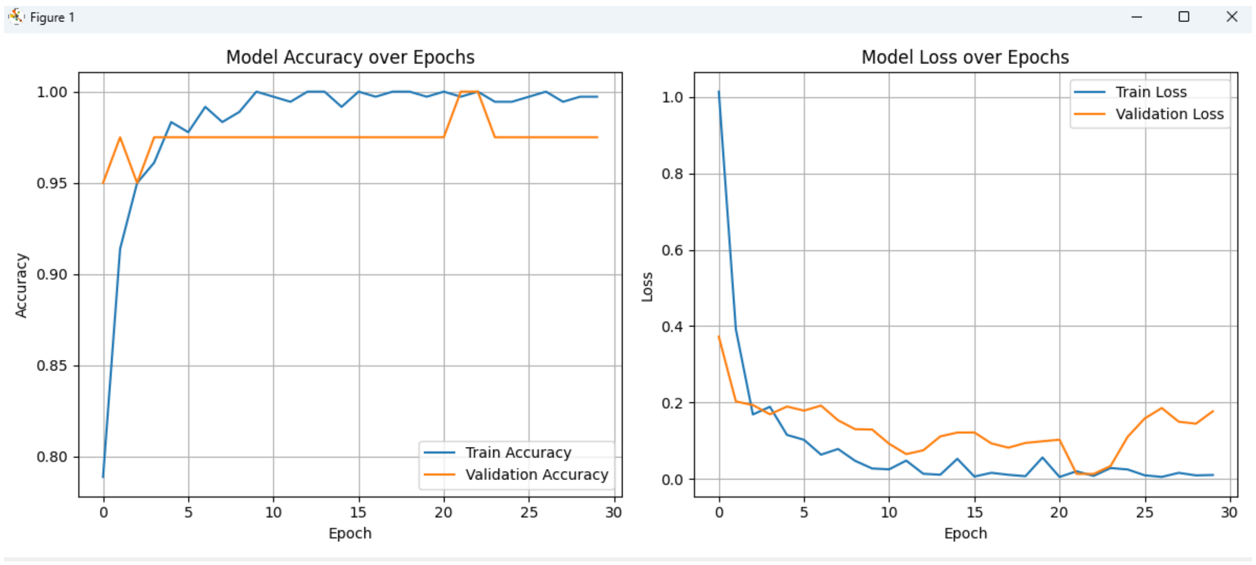
We trained a Random Forest classifier with 500 trees and a maximum depth of 40. Model performance was evaluated using accuracy, a classification report, and a confusion matrix visualized with a heatmap. Additionally, we extracted feature importance scores to identify which statistical features contributed most to the model's decisions. This method leverages summarized statistical descriptors instead of raw or transformed signals, offering a more interpretable and lower-dimensional input for seizure detection.

******Note: Whenever the CHB-MIT and Zenodo datasets are used for random forest, we down-sample the non-seizure class to maintain a 5:1 ratio of non-seizure to seizure samples.******

Results for Both Prefiltered and Unfiltered Datasets

1) Prefiltered Data - Bonn Dataset

a) Feed forward neural network

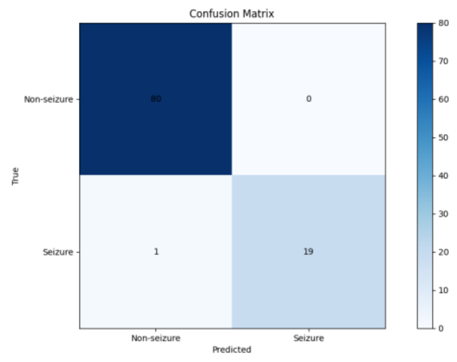


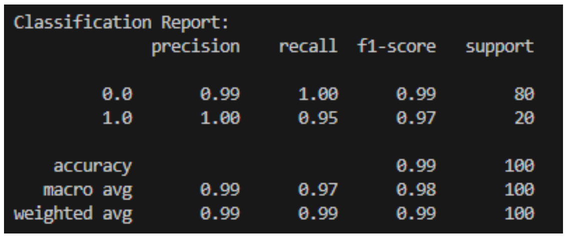
Test accuracy: 0.9800

b) Logistic Regression

-Using Frequency-Domain Features

Test Accuracy: 0.9900

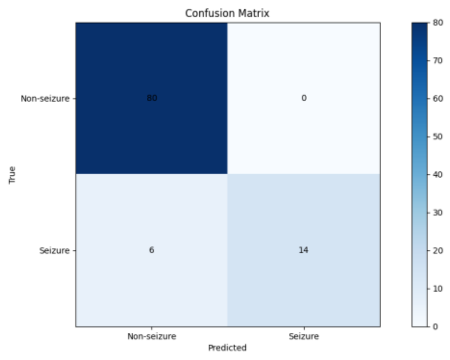


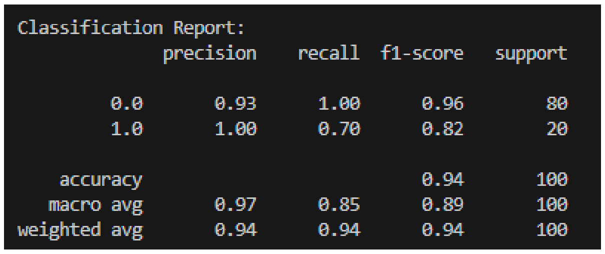


```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99        80
         1.0       1.00      0.95      0.97        20

    accuracy                           0.99       100
   macro avg       0.99      0.97      0.98       100
weighted avg       0.99      0.99      0.99       100
```

- Using Wavelet-Transformed Features

Test Accuracy: 0.9400



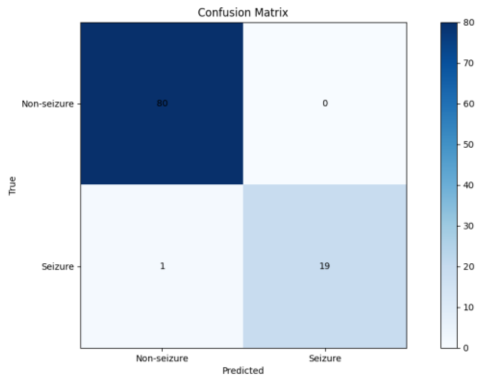


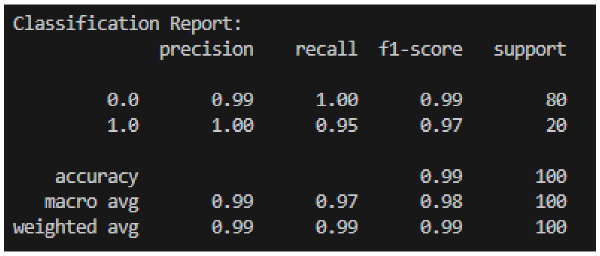
```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.96        80
         1.0       1.00      0.70      0.82        20

    accuracy                           0.94       100
   macro avg       0.97      0.85      0.89       100
weighted avg       0.94      0.94      0.94       100
```

c) Random Forest

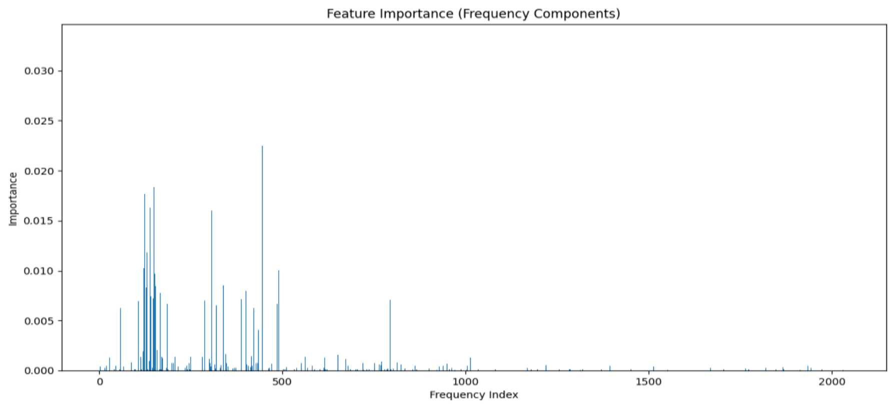
- Using Frequency-Domain Features

Test Accuracy: 0.9900



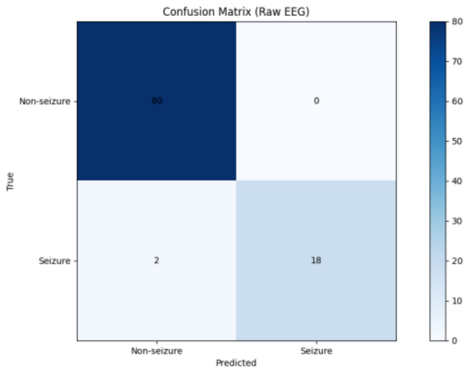


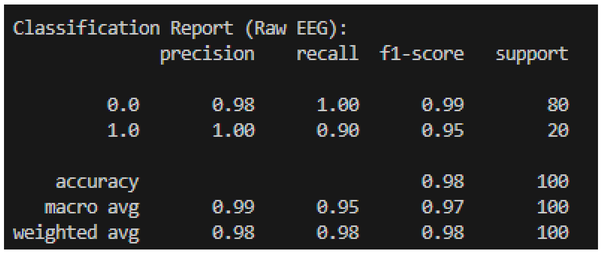
```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99        80
         1.0       1.00      0.95      0.97        20

    accuracy                           0.99       100
   macro avg       0.99      0.97      0.98       100
weighted avg       0.99      0.99      0.99       100
```



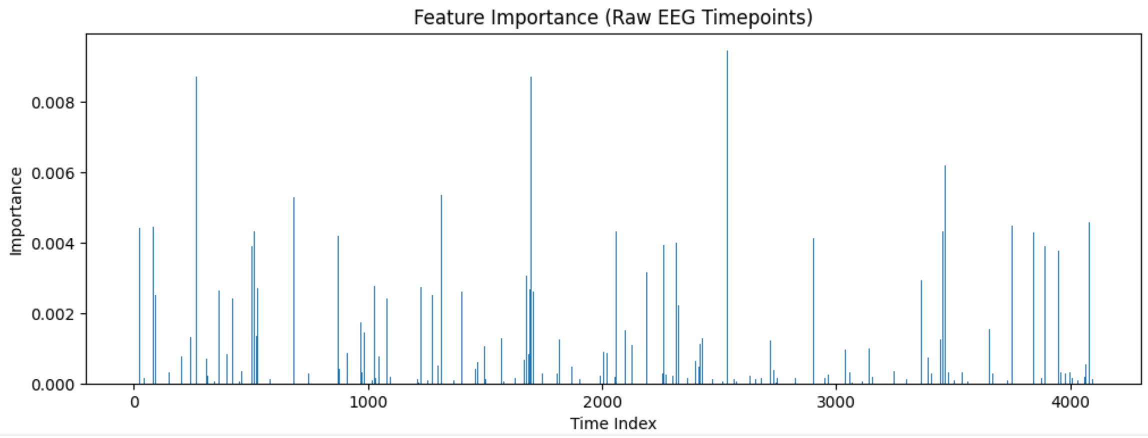
- Using Raw EEG Time-Domain Features

Test Accuracy (Raw EEG): 0.9800





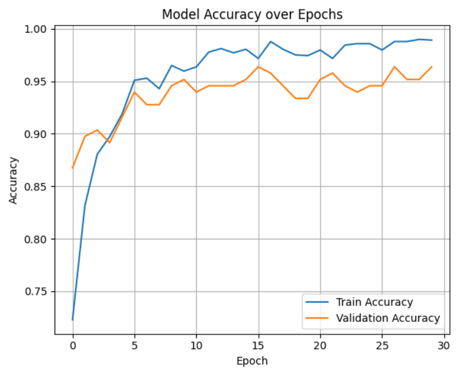
```
Classification Report (Raw EEG):
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99        80
         1.0       1.00      0.90      0.95        20

    accuracy                           0.98       100
   macro avg       0.99      0.95      0.97       100
weighted avg       0.98      0.98      0.98       100
```

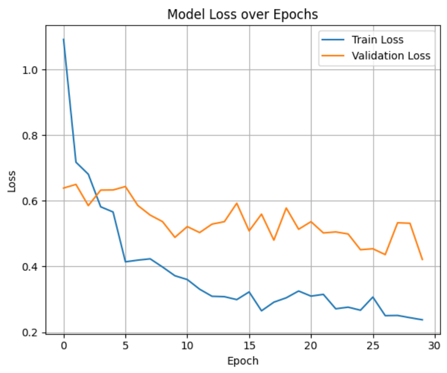


Based on the results from all five models, we can conclude that all models performed well, with the exception of the logistic regression model using wavelet-transformed features, which showed comparatively lower performance.

2) Unfiltered Data with down-sampling – CHB-MIT and Zenodo Datasets

a) Feed forward neural network





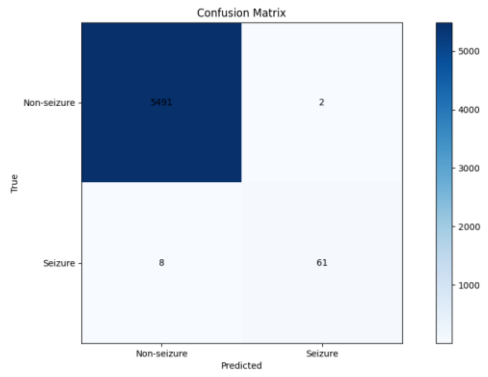
Test accuracy: 0.9493

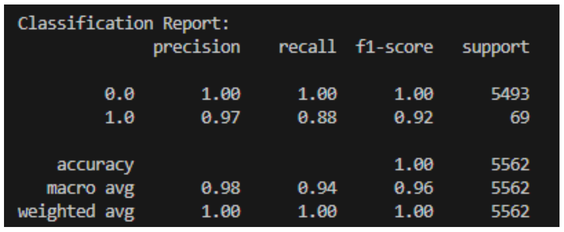
Needed to do L2 regularization in all layers, as without doing the regularization, the validation loss was increasing significantly instead of decreasing.

b) Logistic Regression

-Using Frequency-Domain Features

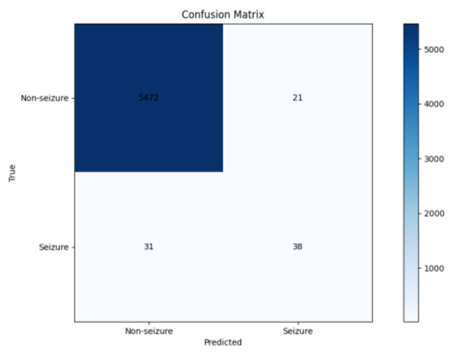
Test Accuracy: 0.9982

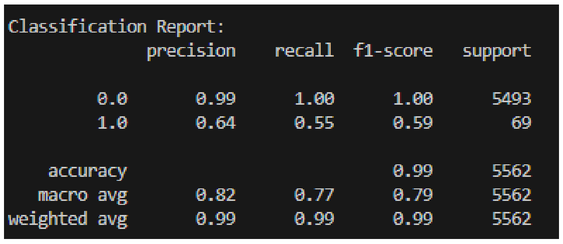




```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      5493
         1.0       0.97      0.88      0.92        69

    accuracy                           1.00      5562
   macro avg       0.98      0.94      0.96      5562
weighted avg       1.00      1.00      1.00      5562
```

- Using Wavelet-Transformed Features





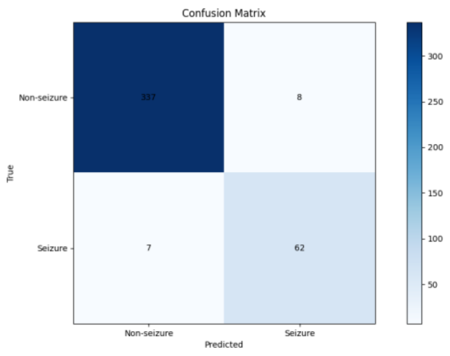
```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00      5493
         1.0       0.64      0.55      0.59        69

    accuracy                           0.99      5562
   macro avg       0.82      0.77      0.79      5562
weighted avg       0.99      0.99      0.99      5562
```

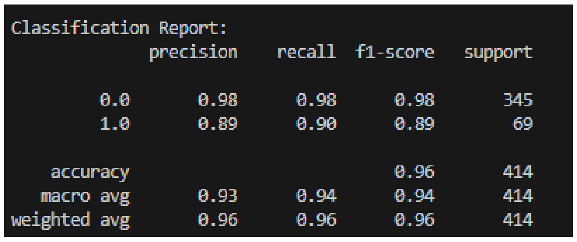
Test Accuracy: 0.9907

c) Random Forest

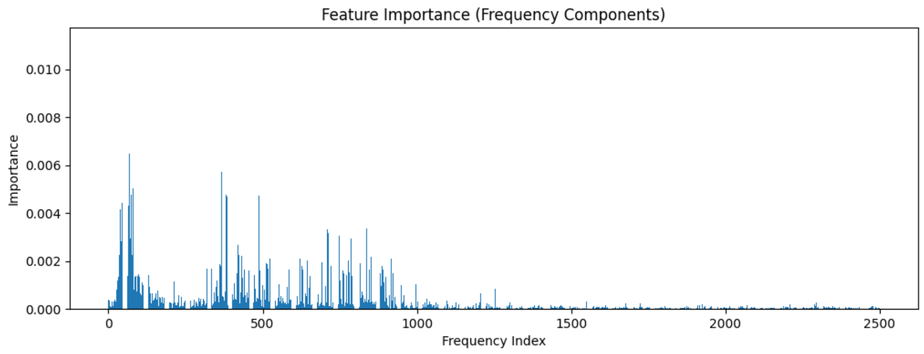
- Using Frequency-Domain Features

Test Accuracy: 0.9638



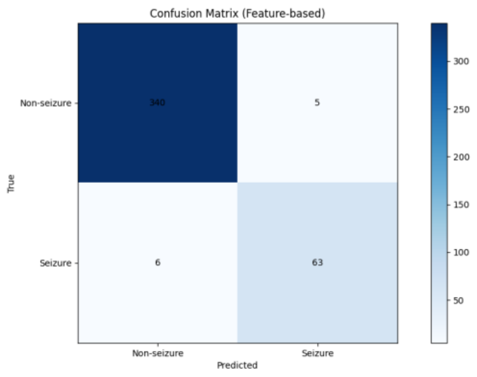


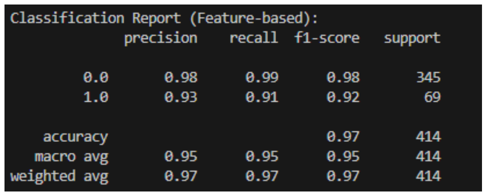
```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98       345
         1.0       0.89      0.90      0.89        69

    accuracy                           0.96       414
   macro avg       0.93      0.94      0.94       414
weighted avg       0.96      0.96      0.96       414
```



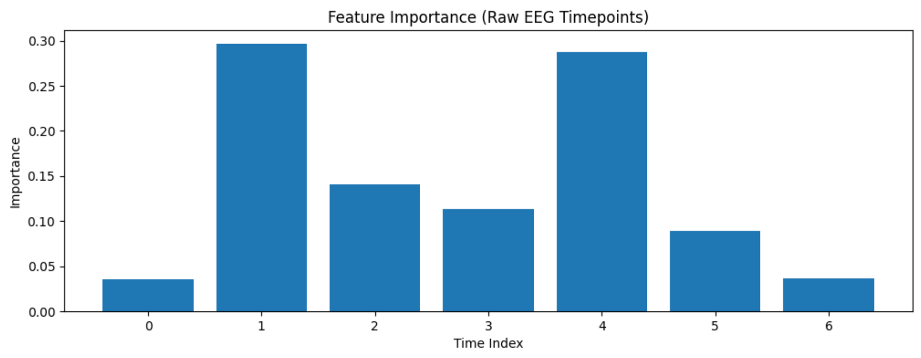
- Using Statistical Features

Test Accuracy (Feature-based): 0.9734





```
Classification Report (Feature-based):
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.98       345
         1.0       0.93      0.91      0.92        69

    accuracy                           0.97       414
   macro avg       0.95      0.95      0.95       414
weighted avg       0.97      0.97      0.97       414
```



Based on the accuracy and evaluation metrics from the models, we observe that the logistic regression model using wavelet-transformed features performs significantly worse in practice, despite reporting a high-test accuracy. This inflated accuracy is due to the model predominantly predicting the majority class (non-seizure), resulting in poor sensitivity to seizures while achieving high accuracy because non-seizure samples are much more frequent. In contrast, the other models demonstrate better balanced performance, with the logistic regression model using Fourier-transformed features achieving the best overall results. Additionally, we note that test accuracy on the CHB-MIT and Zenodo datasets is lower than the accuracy obtained using the Bonn dataset, indicating that residual noise or artifacts may still be present in the data and could benefit from further filtering.

Conclusion

In this project, we explored multiple machine learning models—including feed-forward neural networks, logistic regression, and random forests—for seizure detection and prediction using EEG data. Our analysis incorporated a range of preprocessing techniques, including Fourier and wavelet transforms, as well as statistical feature extraction, applied across both prefiltered and raw datasets. The results demonstrate that while all models achieved high accuracy on the prefiltered Bonn dataset, performance varied more substantially on the raw CHB-MIT and Zenodo datasets.

Among the models tested, logistic regression using frequency-domain features (Fourier transform) consistently achieved the highest accuracy across both prefiltered and raw datasets, highlighting the strong discriminative power of frequency-based patterns in seizure detection. In contrast, the logistic regression model with wavelet-transformed features showed comparatively weaker practical performance, primarily due to its bias toward predicting the majority class in imbalanced data, despite achieving high overall accuracy.

We also observed that models trained on the raw CHB-MIT and Zenodo datasets generally performed slightly worse than those trained on the prefiltered Bonn dataset, suggesting that residual noise and artifacts may still impact classification accuracy despite filtering efforts. This indicates an opportunity for further improvements through enhanced noise reduction, more advanced preprocessing, or additional data cleaning techniques.

Overall, this work underscores the potential of machine learning models for automated seizure detection while also revealing the importance of dataset characteristics, preprocessing strategies, and class balance in achieving reliable, generalizable results. For future work we can work in seizure prediction where we just have to change our method of preprocessing data and then simply use the models that we already have!

Code

For filtering data (CHB-MIT & Zenodo):

```
import mne
import numpy as np
#example paths for loading them
edf_files = [r"C:\Python\python\seizure_detection\CHB-MIT\chb01_03.edf",
             r"C:\Python\python\seizure_detection\CHB-MIT\chb01_04.edf",
             r"C:\Python\python\seizure_detection\CHB-MIT\chb01_15.edf",
             r"C:\Python\python\seizure_detection\CHB-MIT\chb01_16.edf",
             r"C:\Python\python\seizure_detection\CHB-MIT\chb01_18.edf",
             r"C:\Python\python\seizure_detection\CHB-MIT\chb01_21.edf",
             r"C:\Python\python\seizure_detection\CHB-MIT\chb01_26.edf"]
#We know seizure time beforehand
seizure_intervals_list = [[(2996, 3036)],
                          [(1467, 1494)],
                          [(1732, 1772)],
                          [(1015, 1066)],
                          [(1720, 1810)],
                          [(327, 420)],
                          [(1862, 1963)]
                          ]

def segment_signal_custom_logic(data, fs, seizure_intervals, window_size=5000):
    n_channels, n_samples = data.shape
    segments = []
    labels = []

    for ch in range(n_channels):
        signal = data[ch]
        num_windows = n_samples // window_size  # ignore incomplete last window

        for i in range(num_windows):
            start_idx = i * window_size
            end_idx = start_idx + window_size

            # Time of this segment (in seconds)
            segment_start_time = start_idx / fs
            segment_end_time = end_idx / fs

            # Check if no seizure intervals
            if not seizure_intervals:
                label = 0
                segments.append(signal[start_idx:end_idx])
                labels.append(label)
            else:
                # Check if segment is fully inside any seizure interval
                added = False
                for seizure_start, seizure_end in seizure_intervals:
                    if (segment_start_time >= seizure_start) and (segment_end_time <= seizure_end):
                        # fully inside → label=1
                        segments.append(signal[start_idx:end_idx])
                        labels.append(1)
                        added = True
                        break
                    elif (segment_end_time >= seizure_start and segment_start_time < seizure_start) or \
                         (segment_start_time <= seizure_end and segment_end_time > seizure_end) or \
                         (segment_start_time < seizure_start and segment_end_time > seizure_end):
                        # overlaps but not fully inside, skip
                        added = True
                        break
                if not added:
                    # fully outside all seizure intervals → label=0
                    segments.append(signal[start_idx:end_idx])
                    labels.append(0)

    X = np.array(segments)
    y = np.array(labels)
    data_with_labels = np.hstack((X, y[:, None]))
    return data_with_labels

all_data = []

for edf_file, seizure_intervals in zip(edf_files, seizure_intervals_list):
    raw = mne.io.read_raw_edf(edf_file, preload=True)
    raw.filter(1., 40.)
    raw.notch_filter(60.)
    data, times = raw[:]
    fs = int(raw.info['sfreq'])
    dataset = segment_signal_custom_logic(data, fs, seizure_intervals, window_size=5000)
    all_data.append(dataset)

final_dataset = np.vstack(all_data)
print("Final combined dataset shape:", final_dataset.shape)
```

For Feed Forward Neural Nets:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import layers, models

# Split into features (X) and labels (y)
X = all_data[:, :-1]  # EEG data
y = all_data[:, -1]   # Labels
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)

# Build the model
model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X.shape[1],)),
    layers.Dropout(0.3),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=30, batch_size=16, validation_split=0.1)
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy:.4f}")

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.savefig('model_training_results.png', dpi=300)
plt.show()
```

For Logistic Regression:

```
import numpy as np
import pywt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Split features and labels
X_time = all_data[:, :-1]  # EEG time-domain data
y = all_data[:, -1]        # Labels
# Fourier Transform (absolute magnitude)
X_freq = np.abs(np.fft.fft(X_time, axis=1))
X_freq = X_freq[:, :X_freq.shape[1] // 2]
```

```
print("Shape of frequency-domain data:", X_freq.shape)

#Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_freq)
#Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)
#Logistic Regression
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
#Predictions
y_pred = model.predict(X_test)
# Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks([0,1], ['Non-seizure', 'Seizure'])
plt.yticks([0,1], ['Non-seizure', 'Seizure'])

# Annotate counts
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha='center', va='center', color='black')
plt.show()



#Split features and labels
X_time = all_data[:, :-1]  # EEG time-domain data
y = all_data[:, -1]        # Labels
#Wavelet Transform function
def compute_wavelet_features(data, wavelet='db4', level=4):
    features = []
    for signal in data:
        coeffs = pywt.wavedec(signal, wavelet=wavelet, level=level)
        coeff_vector = np.hstack(coeffs)  # Flatten coefficients into one vector
        features.append(coeff_vector)
    return np.array(features)

#Apply wavelet transform
X_wavelet = compute_wavelet_features(X_time)
print("Shape of wavelet-transformed data:", X_wavelet.shape)
#Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_wavelet)
#Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)
#Logistic Regression
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
#Predictions
y_pred = model.predict(X_test)
#Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy: {accuracy:.4f}\n")

print("Classification Report:")
print(classification_report(y_test, y_pred))
#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks([0,1], ['Non-seizure', 'Seizure'])
plt.yticks([0,1], ['Non-seizure', 'Seizure'])

# Annotate counts
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha='center', va='center', color='black')
plt.show()
```

For Random Forest:

```
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

X_time = all_data[:, :-1]
y = all_data[:, -1]
X_freq = np.abs(np.fft.fft(X_time, axis=1))
X_freq = X_freq[:, :X_freq.shape[1] // 2]
print("Fourier transformed data shape:", X_freq.shape)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_freq)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)

#Random Forest Classifier
model = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)
model.fit(X_train, y_train)

#Predictions
y_pred = model.predict(X_test)
#Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy:  {accuracy:.4f}\n")

print("Classification Report:")
print(classification_report(y_test, y_pred))

#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks([0,1], ['Non-seizure', 'Seizure'])
plt.yticks([0,1], ['Non-seizure', 'Seizure'])
```

```
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha='center', va='center', color='black')

plt.show()

#Feature Importance
importances = model.feature_importances_
plt.figure(figsize=(12, 4))
plt.bar(range(len(importances)), importances)
plt.title('Feature Importance (Frequency Components)')
plt.xlabel('Frequency Index')
plt.ylabel('Importance')
plt.show()


#Using raw EEG data
X_raw = all_data[:, :-1]
y = all_data[:, -1]
print("Raw EEG data shape:", X_raw.shape)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)

model = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy (Raw EEG): {accuracy:.4f}\n")
print("Classification Report (Raw EEG):")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix (Raw EEG)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks([0,1], ['Non-seizure', 'Seizure'])
plt.yticks([0,1], ['Non-seizure', 'Seizure'])
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha='center', va='center', color='black')

plt.show()
importances = model.feature_importances_
plt.figure(figsize=(12, 4))
plt.bar(range(len(importances)), importances)
plt.title('Feature Importance (Raw EEG Timepoints)')
plt.xlabel('Time Index')
plt.ylabel('Importance')
plt.show()
```

Lastly, for random forest using statistical feature:

```
from scipy.stats import kurtosis, skew

def extract_features(signal):
    return [
        np.mean(signal),
        np.std(signal),
        np.min(signal),
        np.max(signal),
        np.sqrt(np.mean(signal ** 2)),  # RMS
        kurtosis(signal),
        skew(signal)
    ]

X_raw = all_data[:, :-1]  # shape: (num_rows, num_samples_per_window)
y = all_data[:, -1]
features_list = []
for row in X_raw:
    features = extract_features(row)
    features_list.append(features)

X_features = np.array(features_list)
X_majority = X_features[y == 0]
y_majority = y[y == 0]
X_minority = X_features[y == 1]
y_minority = y[y == 1]
n_samples_majority = len(y_minority) * 5
```

```
# Downsample majority to 5x minority size
X_majority_downsampled, y_majority_downsampled = resample(
    X_majority, y_majority,
    replace=False,
    n_samples=n_samples_majority,
    random_state=42
)
X_balanced = np.vstack((X_minority, X_majority_downsampled))
y_balanced = np.hstack((y_minority, y_majority_downsampled))
y = y_balanced
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_balanced)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)
model = RandomForestClassifier(
    n_estimators=500,
    max_depth=40,
    n_jobs=-1
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy (Feature-based): {accuracy:.4f}\n")
print("Classification Report (Feature-based):")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix (Feature-based)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks([0,1], ['Non-seizure', 'Seizure'])
plt.yticks([0,1], ['Non-seizure', 'Seizure'])
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha='center', va='center', color='black')

plt.show()
importances = model.feature_importances_
plt.figure(figsize=(12, 4))
plt.bar(range(len(importances)), importances)
plt.title('Feature Importance (Raw EEG Timepoints)')
plt.xlabel('Time Index')
plt.ylabel('Importance')
plt.show()
```

Works Cited:

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation, 101(23), e215–e220. https://doi.org/10.1161/01.CIR.101.23.e215

Guttag, J. (2010). CHB-MIT Scalp EEG database (version 1.0.0) [Dataset]. PhysioNet. https://doi.org/10.13026/C2K01R

Shoeb, A. (2009). Application of machine learning to epileptic seizure onset detection and treatment (Doctoral dissertation, Massachusetts Institute of Technology).

Stevenson, N., Tapani, K., Lauronen, L., Vanhatalo, S., & Metsäranta, M. (2018). A dataset of neonatal EEG recordings with seizure annotations [Dataset]. Zenodo. https://doi.org/10.5281/zenodo.2547147

Andrzejak, R. G., Lehnertz, K., Rieke, C., Mormann, F., David, P., & Elger, C. E. (2001). 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, 64(6), 061907. https://doi.org/10.1103/PhysRevE.64.061907