REI602M Machine Learning

Neonatal Seizure Detection in Newborn Children

Andri Freyr Viðarsson, Haraldur Bjarni Davíðsson

Instructor: Steinn Guðmundsson

22. apríl 2020

1 Introduction and data description

EEGs (Electroencephalograms) are recordings that monitor electrical activity in the brain. The EEGs are obtained by placing electrodes evenly distributed across the scalp and monitoring the voltage difference between the electrode pairs (see figure 1). The objective of this project is to construct a classifier for neonatal EEG recordings in newborn children and use the classifier to detect seizure segments.(NHS, 2022 [Online])

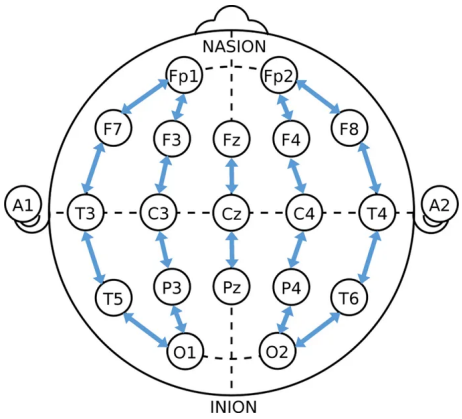

Figure 1: The bipolar EEG montage used by reviewers to annotate the presence of seizures. Source: Nature, A dataset of neonatal EEG recordings with seizure annotations.

The data consists of 79 different EEG recordings, each with different amounts of seizure segments. Some recordings have many segments, some a few, and others none. The declared minimum duration of seizure was 10 seconds. The length of each recording is roughly 74 minutes, each containing 256 samples per second. Each recording has measurements from 21 different channels, where each channel represented measures from separate electrodes (see figure 1), a respiratory system monitor and a heart monitor. Each second of the recordings was individually labeled by three experts, 1 if the expert believed the segment to be a seizure, 0 if he did not.

The data is quite sensitive as 39 of the recordings did not contain any seizure segments and seizure segments from the same individual are often very much alike so one must be quite proficient when choosing the training, testing and validation data when working with this dataset. Another thing to keep in mind when the data is being trained is the balancing of the data, since the data consists of way more none-seizure segments than seizure segments and choosing a random x/y testing/training split would lead to immense over-fitting. After we consider the training and test split, we need to explore which parameters our model should use and build the model around that data. The original voltage difference presented by the raw measurements of the EEGs did not seem very intuitive but did however display some properties that appeared to differentiate in seizure segments and non-seizure segments. The parameters used for the models were based on our observations and research (see figure 2).

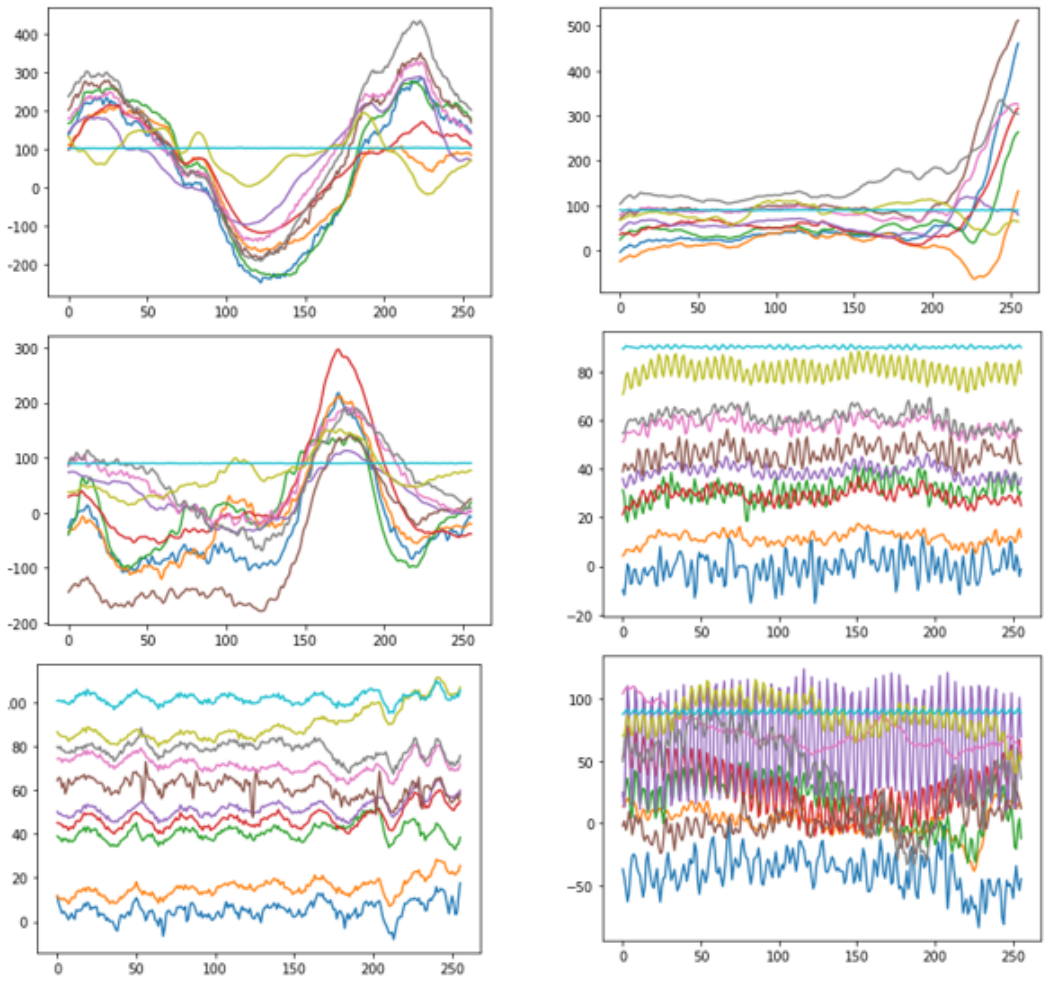

Figure 2: Three random one second seizure (to the left) and non-seizure (to the right) segments of 10 different channels.

2 Data processing and feature extraction

2.1 Data processing

First, we needed to decide the interval on which the data was going to be examined and which of the 21 channels should be used when building the model. Since there were 256 recordings each second there were many different possibilities for viewing the data. Should it be viewed in compliance with the experts' labels, as one second blocks or should they be extended or possibly shortened? Working with shorter than one second data blocks turned out to be a hassle and using larger than two or three second blocks seemed to provide too few samples. Thus, a one second interval was deemed to be appropriate. We also decided on not using all the channels since computations were slow and a reduced number of channels did not seem to affect the performance

of the model at all. Thus, 10 channels were deemed appropriate, 5 on the left side and 5 on the right, where each channel corresponded to another one on the opposite side (see figure 1). For each of the 79 children participating, there was a corresponding .edf format file containing recordings of EEG samples for all 21 channels and an annotation file containing experts' scoring. Categorization of seizure segments was determined by a majority vote, meaning that at least two out of the three experts needed to score segment as a seizure for it to be labeled as one.

We opted for a variant version of "leave one out" testing approach. Recordings from 8 children were entirely left out of the training/testing data and the recordings from other children were stacked, keeping track of seizure segment indices. The training/testing data was constructed using a $\frac{1}{3}$, $\frac{2}{3}$ split of random seizure and non-seizure segments from the stacked data. Therefore, there are seizure segments from the same child in both the training and test data. As seizure segments from the same individual have similar features, the testing prediction did not give a clear measure of accuracy for the classifiers. We tried to minimize the overlap by generating the training/testing data from separate subsets of children but that did not produce promising results either.

2.2 Feature selection

The first factors we analyzed were the Hjorth parameters. The Hjorth parameters are indicators of statistical properties used in signal processing. The Hjorth parameter we were most interested in was the Hjorth Complexity, which represents the change in frequency as the frequency of the signals seemed to change more rapidly during seizure segments. The Hjorth Complexity is defined as:

$$\text{Complexity} = \frac{\text{Mobility}\left(\frac{dy(t)}{dt}\right)}{\text{Mobility}(y(t))}$$

Where the Hjorth mobility parameter represents the mean frequency or the proportion of standard deviation of the power spectrum:

$$\text{Mobility} = \sqrt{\frac{\text{var}\left(\frac{dy(t)}{dt}\right)}{\text{var}(y(t))}}$$

(Hjorth, 1970)

The second factor examined was the variance of the derivative. The variance itself as a factor seemed to rise somewhat during some seizure segments but was not consistent throughout the recordings. We did, however, notice that the variance seemed to flicker a bit more during the seizure segments, so we decided to inspect the variance of the derivative instead. Alas, the variance of the derivative seemed to give a much better picture of the seizure segments than the variance as a stand alone factor by itself and ultimately proved to be a very important feature.

The third and final factors we inspected were different power spectrums of the band power.

Which frequency would provide us with the best model was a mystery to us and appeared to be random for different seizure segments. Many an hour went into inspecting and researching different power spectrums and in the end, we decided to use four different spectrums (4-8 Hz, 8-16 Hz, 16-32 Hz and 32-64 Hz). The reason for this selection was based on research we gathered, stating that it was quite common for EEG experiments to examine spectrums like these, larger spectrums for higher frequencies and smaller for lower. For the SVC model the data was scaled by dividing each feature by the mean across all of the training data.

On top of this, we also experimented with image processing using convolutional neural networks by producing images of 1 second intervals of several channels, and then converting the images to NumPy arrays to use as the training and testing data. This approach, however, proved to be a bit too ambitious as the data processing soaked up an immense amount of time and resources and the results ultimately proved inadequate. Therefore, the final verdict was to use the parameters and methods listed above to build our model and hopefully we would be able to create a classifier that would to some extent be able to detect seizures from non-seizures.

2.3 Post-Processing

When estimating the classes for whole segments from one individual, the data prediction was processed to resemble the structure of the annotations. Knowing beforehand that each seizure was at least 10 second long, the prediction was filtered so that if two elements in the prediction list were predicted as seizures and the distance between them is 13 seconds or less, all of the elements in between them were also marked as seizures. After this step, all seizure segments lasting for 2 seconds or less were re-marked as non-seizures.

3 Modeling

3.1 Methods

After we had finished all the pre-processing, the next step was to choose the base classifier itself. This step in the process is always crucial when conducting experiments such as these. However, with modern methods in this field, this step is not as difficult as it may appear to be. One can simply test the data on various classifiers and roughly determine which one performs the best, for which classifier to use for a dataset is more than often not too obvious. In this case, we ruled out simple linear classifiers due to the complexity of the data. The parameters for each classifier can be grid-searched using small subsamples of the data and then the optimal parameters from the grid-search can be used to train a larger dataset. In our case, the grid-search was conducted using the scikit-learn GridSearchCV and optimized for maximum AUC (Area Under ROC Curve).

When working with binary labeled data, the accuracy is not always the best measurement for determining the performance of the classifier. If the test data consists of 85% of one class, a

90% test accuracy is not very intuitive and it is unclear in what aspect the classifier is underperforming. A good way to get a better insight into how the classifier is labeling the data, is to compute a confusion matrix. The confusion matrix is defined as:

True Negatives	*False Positives*
False Negatives	*True Positives*

Where true positives display the number of times the classifier predicted 1 when the correct label was 1, false positives display the number of times the classifier predicted 1 when the correct label was 0 and so forth. The factors of the confusion matrix we then want to maximize are the true positive and true negative rates, also known as the sensitivity and the specificity:

$$\text{Sensitivity} = \frac{TP}{TP + FP}$$

$$\text{Specificity} = \frac{TN}{TN + FN}$$

The sensitivity tells us the ratio of correctly predicted 1s and specificity tells of the ratio of correctly predicted 0s. When we eventually tested the models for the data, we used the confusion matrix and the metrics listed above to determine whether a model's performance was satisfactory. The base classifiers we used on the test data were the following: a Random Forest Classifier, a RBF kernel Support Vector Machine, a Gradient Boosting Classifier and a simple fully connected Neural Network. The models were implemented using NumPy, scikit-learn, LightGBM and tensorflow.

3.2 Models

Support Vector Machines

Support Vector Machines are classification algorithms that seek to maximize the distance between boundaries of different classes. When first introduced, support vector machines could only be used on linear classification problems but by using a kernel trick the methods can be expanded on a non-linear separation boundary. When implementing a non-linear support vector machine it is common to present a dual problem of a soft margin SVM and solve it using kernel function. A kernel function measures the similarity of two samples u and v, the kernel function used in this project is the Gaussian Rbf kernel:

$$\exp(-\gamma \|u - v\|_2^2)$$

where $\gamma > 0$ is a parameter subject to tuning(Guðmundsson, 2020).

Random Forest

A Random Forest is a decision tree based ensemble classification method that uses bootstrap aggregation and stochastic feature selection. The forest consists of many decision trees that implement the CART algorithm. The prediction of each sample is then based on a majority vote amongst the trees in the forest. The main parameters in a Random Forest model are the number of trees and the number of features to consider when looking for the best split in the CART algorithm. For this model we did not gridsearch for optimal parameters and instead used the default parameters in the scikit implementation. That is the number of trees = 100 and the number of features = square root of total number of available features.(Guðmundsson, 2020)

Gradient Boosting

Boosting is an ensemble method that uses decision trees or similar models for prediction. Unlike Random Forest, the trees in boosting methods are built sequentially, meaning that each tree is grown using information from previously grown trees. By doing that the algorithm aims to correct wrong predictions from previous trees. This is done by fitting decision trees to the current residual state of the model in each iteration rather than the outcome. Gradient boosting is more volatile to overfitting than a random forest model, thus the main parameters in a boosting model need to be tuned carefully. Those parameters are number of trees and the learning rate.(Gareth James, 2013)

Neural Network

Neural Networks are prediction models that are inspired by biological neural networks that constitute animal brains. A neural network consists of nodes and vertices, each vertex in the network has an assigned weight and in each node there is an activation function that passes data forward through the network. Generally, the weights of a network are tuned using batch gradient descent to minimize a given loss function. In our case we have a binary classification problem, thus the binary cross-entropy loss function is appropriate. We constructed a simple neural network with one hidden layer of 512 nodes and an output layer with a sigmoid activation.(Guðmundsson, 2020)

4 Results

4.1 Support Vector Machines

After running a cross-validated grid-search the chosen parameters for the SVC model were: $C = 100, \gamma = 0.01$, where C is the regularization strength. With those parameters the confusion matrix for the original testing data was:

18485	1396
4712	5107

The table below shows the confusion matrices for each of the left out samples.

Recording ID													
67		35		45		55		2		34		37	
3275	122	3391	376	3652	170	4873	288	3620	141	5816	223	4562	16
1424	79	0	0	0	0	0	0	0	0	226	226	0	0

As can be seen from the first confusion matrix this model does not even produce great results on the original test data that contains similar segments as the training data. Predictions on the left-out segments that contain seizures are not acceptable. Post-processing the predictions on the left out data did not improve the accuracy.

4.2 Random Forest

No grid-searching was done for this model (see chapter 3.2). The confusion matrix when tested on the original data:

18832	1049
2028	7791

Recording ID													
67		35		45		55		2		34		37	
3394	3	3743	24	3792	30	5131	30	3757	4	6030	9	4578	0
1503	0	0	0	0	0	0	0	0	0	419	33	0	0

The sensitivity and specificity for the Random Forest Classifier were significantly larger than the ones of the SVC model, but this model appears to perform worse for the left out segments. The post-processing did not improve these predictions either.

4.3 Gradient Boosting

After running a cross-validated grid-search the Gradient Boosting Classifier, the learning rate of the model was 0.05 and the number of estimators was 1000. The confusion matrix for the original test data was:

18628	1253
1778	8041

Recording ID													
67		35		45		55		2		34		37	
3394	3	3663	104	3784	38	5103	58	3755	6	5992	47	4578	0
1497	6	0	0	0	0	0	0	0	0	314	111	0	0

The sensitivity and specificity for the Gradient Boosting Classifier on the original data tests were very similar to the ones of the Random Forest Model. However, the matrices for the left-out segments look slightly better for this model. The post-processing did seem to improve the results a bit for some of the id's, but decrease for others. The post-processing matrices were:

Recording ID													
67		35		45		55		2		34		37	
3395	2	3285	482	3727	95	5018	143	3759	2	5974	65	4578	0
1485	18	0	0	0	0	0	0	0	0	109	343	0	0

The post-processing seems to work wonders for id no. 34 and improve the accuracy for id no. 67 ever so slightly. The accuracy decreased for the other ones, which were all seizure-free. This tells us that the post-processing algorithm should not be run on seizure-free segments on the data labeled by this classifier.

4.4 Neural Network

No grid searching was done on the network. The confusion matrix for the original test data was:

17853	2538
2028	7281

Recording ID													
67		35		45		55		2		34		37	
3232	165	3162	605	1169	2598	4787	374	3653	108	5451	588	4578	5
1446	57	0	0	0	0	0	0	0	0	326	126	0	0

The sensitivity and specificity for this classifier when tested on the original data showed some positive results when compared with previous results of the Random Forest and Gradient Boosting Classifiers. The classifier performed very poorly on the left-out segments, however. Post-processing did not improve the performance on this model. This was a simple Neural Network with a single dense layer and not much time was spent on improving this. A more complex Convolutional Neural Network would most likely have achieved better results.

5 Conclusion

The end results were disappointing. Despite the classifier displaying great results on the test data, it performed very poorly when tested on the left-out segments. This is, as was stated in the second chapter, most likely due to the fact that the test set and the training set both contained segments from the same children and the classifier became too optimistic. The classifier that appeared to perform the best based on the results from the left-out segments was the gradient boosting classifier. The enormous discrepancy between the accuracy of the test set and the left-out children does, however, somewhat attest to there being a bug in the code, but given the time that was spent on debugging, this possibility seems highly unlikely. Perhaps we should have been more patient with the image processing approach, but given the time constraint, that was a risk we were not willing to take, especially when we had already constructed a concrete model that appeared to be somewhat functional. Perhaps our leave-one-out testing approach was flawed and a different approach should have been taken instead since the results seemed unclear and inconsistent. Another thing to keep in mind was that there were only 80 samples and seizure segments seemed to differentiate between samples quite a bit, as is displayed by the low accuracy of sample id no. 67. Human error is also a big factor to be considered as there were only three experts and their labeling was not quite as consistent as we would have liked. Perhaps with more samples and more consistent labeling, we could have constructed a more accurate model.

With a project of this scale and severity, to create a model that would be able to detect the seizures almost flawlessly, a model that would be useful in solving real world problems, would take a lot more time and resources than we had to work with. Despite that fact, we were still not satisfied with our results and we think it is safe to say that our ambitions got the better of us. We feel that we are not walking out of this experience empty handed though, it is our belief that we learned a great deal. We, for instance, got more familiar with the methodology of the binary classifiers and even discovered a new classifier that had, up until this point, been completely unknown to us. We also drew some lessons from the complications of conducting a large machine learning project and working with intricate data of this scale.

References

Gareth James, T. H. R. T., Daniela Witten. (2013). *An Introduction to Statistical Learning: with Applications in R.* Springer.

Guðmundsson, S. (2020). *Lecture Notes (REI 602M) - Spring 2020.* University of Iceland.

Hjorth, B. (1970). Eeg analysis based on time domain properties. *Electroencephalography and Clinical Neurophysiology*, *29*(3), 306-310. Retrieved from `https://www.sciencedirect.com/science/article/pii/0013469470901434` doi: https://doi.org/10.1016/0013-4694(70)90143-4

NHS. (2022 [Online], Jan. 5,). *Electroencephalogram (eeg).* Retrieved from `https://www.nhs.uk/conditions/electroencephalogram/`

Appendix 1 - Contribution of each group member

There was a lot of cooperation within the group, both members of the group took active part in all phases of the project.

Appendix 2 - Code

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from itertools import chain
from utils.edf import read_edf
np.set_printoptions(4)

# Data Processing

np.random.seed(35429671) # force random vals
leave_out_ids = np.random.choice(np.arange(1,80), size = 8, replace = False)

# y - list of children, could later on be joined and combined
# y majority vote labeling
y_labels = []
for recording_id in range(1, 80):
  if recording_id not in leave_out_ids:
    y_iter = [] # list of labels 0-1
    annot = np.genfromtxt('annotation/id{:d}.txt'.format(recording_id),\
                      delimiter='\t',skip_header=False)
    for i in annot:
      if sum(i) >1:
        y_iter.append(1)
      else:
        y_iter.append(0)
    y_labels.append(y_iter)

  y = np.array(y_labels)

left_idx = [0, 2, 4, 6, 8] # index of channels in left brain
right_idx = [1, 3, 5, 7, 9] # index of channels in right brain

chan_idx = left_idx + right_idx
n_channels = len(left_idx)+ len(right_idx)
```

```
sampling_freq = 256

x_data = np.empty((71, n_channels),dtype = object)
row_idx = 0
for recording_id in range(1, 80):
  if recording_id not in leave_out_ids:
    file_name = './eeg/eeg{}.edf'.format(recording_id)
    data = read_edf(file_name, 256)[3]
    n_samples = data.shape[1]
    n_seconds = int(n_samples/sampling_freq)
    for i, chan in enumerate(chan_idx):
      signal = data[chan, :]
      signal_intervals = np.split(signal, n_seconds)
      x_data[row_idx, i] = signal_intervals
    row_idx += 1

y_complete = list(chain(*y_labels)) # list of labels, stacked
X = np.empty((n_channels),dtype= object) # stack on top of each
for i in range(10):
  X[i] = list(chain(*x_data[:, i]))

X_dat = np.empty((len(y_complete), n_channels),dtype= object)
for i in range(10):
  X_dat[:,i] = X[i]

s_index = [index for index, value in enumerate(y_complete) if value == 1]
non_index = [index for index, value in enumerate(y_complete) if value == 0]

# Create more balanced data (2/3) non seizure (1/3) seizure
# get 30.000 seizure segments, 60.000 non seizure segments

s_select = list(np.random.choice(len(s_index)-1, 30000, replace = False))
n_select = list(np.random.choice(len(non_index)-1, 60000, replace = False))

# Functions used to calculate features
from scipy.integrate import simps
from scipy import signal

def band_power(x, fs, low=0.5, high=4, win=1):
    # Calculate the power spectrum of a signal
    # Input: x contains EEG data from a single channel
    #        fs sampling rate
```

```
    #         low, high are the frequency band limits
    #         win is the window length
    # Output: Absolute and relative power in the [low,high] band

    # Code is based on: https://raphaelvallat.com/bandpower.html

    # Welch's averaged periodogram is based on the Fast Fourier Transform (FFT)
    freqs, psd = signal.welch(x, fs, nperseg=win*fs)
    idx_delta = np.logical_and(freqs >= low, freqs <= high)
    freq_res = freqs[1] - freqs[0]

    # Compute the absolute power by approximating the area under the curve
    abs_power = simps(psd[idx_delta], dx=freq_res)
    #print('Absolute delta power: %.3f uV^2' % abs_power)

    # Relative delta power (expressed as a percentage of total power)
    total_power = simps(psd, dx=freq_res)
    rel_power = abs_power / total_power
    #print('Relative delta power: %.3f' % rel_power)
    return abs_power, rel_power

def diff_var(x):
  dx = np.diff(x)
  d_var = np.var(dx)
  return d_var

def hjorth_mob(x):
  dx = np.diff(x)
  mob = np.sqrt(np.var(dx)/np.var(x))
  return mob

def hjorth_comp(x):
  # trouble when all elements in x are the same
  # detect whether that is the case
  if np.isnan(hjorth_mob(x)):
    return 0

  dx = np.diff(x)
  comp = hjorth_mob(dx)/hjorth_mob(x)
  return comp

# generate train/test data
```

```
s_dat = [s_index[i] for i in s_select]
n_dat = [non_index[i] for i in n_select]
X_s = X_dat[s_dat, :] # seizure segments
X_n = X_dat[n_dat, :] # non_seizure segments

# train / test will be shuffled
X = np.vstack((X_s, X_n)) #
y = np.zeros((90000, 1))
y[0:30000] = 1


X_features = np.zeros((X.shape[0], 6*X.shape[1]))
for i in range(X.shape[0]):
  for j in range(X.shape[1]):
    segment_j = X[i, j]
    b_power_1 = band_power(segment_j, sampling_freq, 4, 8)
    b_power_2 = band_power(segment_j, sampling_freq, 8, 16)
    b_power_3 = band_power(segment_j, sampling_freq, 16, 32)
    b_power_4 = band_power(segment_j, sampling_freq, 32, 64)
    h_comp = hjorth_comp(segment_j)
    diff_ = diff_var(segment_j)
    X_features[i, j*6] = b_power_1[0]
    X_features[i, j*6+1] = b_power_2[0]
    X_features[i, j*6+2] = b_power_3[0]
    X_features[i, j*6+3] = b_power_4[0]
    X_features[i, j*6+4] = h_comp
    X_features[i, j*6+5] = diff_

# plot 3 random seizure/ non seizure segments

# seizure
print('...seizure...')
inds = np.random.choice(np.arange(1,10000), size = 3,replace = False)
for i in inds:
  for j in range(10):
    plt.plot(X_s[i, j]+j*10)
  plt.show()

#non-seizure
print('...non-seizure...')
inds = np.random.choice(np.arange(1,10000), size = 3,replace = False)
```

```
for i in inds:
  for j in range(10):
    plt.plot(X_n[i, j]+j*10)
  plt.show()


# generate y_labels and features for individual childre

def gen_y(recording_id):
  y_id = [] # list of labels 0-1
  annot = np.genfromtxt('annotation/id{:d}.txt'.format(recording_id),\
                        delimiter='\t',skip_header=False)
  for i in annot:
    if sum(i) >1:
      y_id.append(1)
    else:
      y_id.append(0)
  return y_id


def calc_features(recording_id):
  file_name = './eeg/eeg{}.edf'.format(recording_id)
  data = read_edf(file_name, 256)[3]
  n_samples = data.shape[1]
  n_seconds = int(n_samples/sampling_freq)
  X_data = np.empty((1, n_channels), dtype = object)

  for i, chan in enumerate(chan_idx):
    signal = data[chan, :]
    X_data[0, i] = signal

  X = np.empty((n_seconds, n_channels),dtype= object) # stack on top of each
  for i in range(10):
    X[:, i] = np.split(X_data[0, i], n_seconds)

  X_features_out = np.zeros((X.shape[0], 6*X.shape[1]))
  for i in range(X.shape[0]):
    for j in range(X.shape[1]):
      segment_j = X[i, j]
      b_power_1 = band_power(segment_j, sampling_freq, 4, 8)
      b_power_2 = band_power(segment_j, sampling_freq, 8, 16)
      b_power_3 = band_power(segment_j, sampling_freq, 16, 32)
```

```
        b_power_4 = band_power(segment_j, sampling_freq, 32, 64)
        h_comp = hjorth_comp(segment_j)
        diff_ = diff_var(segment_j)
        X_features_out[i, j*6] = b_power_1[0]
        X_features_out[i, j*6+1] = b_power_2[0]
        X_features_out[i, j*6+2] = b_power_3[0]
        X_features_out[i, j*6+3] = b_power_4[0]
        X_features_out[i, j*6+4] = h_comp
        X_features_out[i, j*6+5] = diff_
    return X_features_out

# post processing

def convert_to_seizure(first_index, second_index, list):
        list[first_index:second_index] = 1

def convert_from_seizure(first_index, second_index, list):
        list[first_index:second_index] = 0


def filter_pred(y_pred):
        interval = 15
        for index in range(len(y_pred)):
                if y_pred[index] == 1:
                        no_seizure = True
                        for index_2 in range(index+1,index+interval+2):
                                try:
                                        if y_pred[index_2] == 1:
                                                convert_to_seizure(index,index_2+1,y_pred)
                                                index = index_2+1
                                                no_seizure = False
                                except:
                                        continue
        count = 0
        for index in range(len(y_pred)):
                if y_pred[index] == 1:
                        count+=1
                else:
                        if count < 3:
                                y_pred[index-count:index] = 0
                        count = 0
        return
```

```
# base models

#Random forest
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

X_features = np.nan_to_num(X_features)

X_train, X_test, y_train, y_test = train_test_split(X_features, y, test_size=0.33, random_state=42)
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.33, random_state=42)

clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

# left out testing

for recording_id in leave_out_ids:
  print('id:', recording_id)
  y = gen_y(recording_id)
  X = calc_features(recording_id)
  y_pred = clf.predict(X)
  print(confusion_matrix(y, y_pred))

  filter_pred(y_pred)
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)

# Gradient boosting

import lightgbm as lgb
from sklearn.model_selection import GridSearchCV

y = np.zeros((90000, 1))
y[0:30000] = 1

X_train, X_test, y_train, y_test = train_test_split(X_features, y, test_size=0.33, random_state=42)
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.33, random_state=42)
```

```
# unlike random trees, boosting can overfit if n_estimators is too large
gbm = lgb.LGBMClassifier(learning_rate = 0.25,
                         n_estimators = 100, num_leaves = 31)

param_grid = {
    'n_estimators': [x for x in range(100,1500,100)],
    'learning_rate': [0.01, 0.05, 0.125, 0.25, 0.5, 1]}

grid_search = GridSearchCV(gbm, param_grid)

grid_search.fit(X_tr, y_tr,
                eval_set = [(X_val, y_val)],
                eval_metric = ['auc'],
                early_stopping_rounds= 10,
                verbose = 0)

param_select = grid_search.best_params_
print(param_select)

learning_rate, n_estimators = param_select['learning_rate'], param_select['n_estimators']
gbm = lgb.LGBMClassifier(learning_rate = learning_rate,
                         n_estimators = n_estimators, num_leaves = 31)

gbm.fit(X_tr, y_tr,
        eval_set=[(X_val, y_val)],
        eval_metric=['auc'],
        verbose = 1,
early_stopping_rounds=10)

y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

# left out testing

for recording_id in leave_out_ids:
  print('id:', recording_id)
  y = gen_y(recording_id)
  X = calc_features(recording_id)
  y_pred = gbm.predict(X)
```

```
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)
  print()
  filter_pred(y_pred)
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)

# SVC
from sklearn.svm import SVC

def data_subsample(X, y, n):
  # Select a random subset of the training data
  perm = np.random.permutation(len(y))
  X_sub=X[perm[0:n],:]
  y_sub=y[perm[0:n]]
  return X_sub, y_sub

# scale data for svm

y_train  =y_train.reshape(-1)
y_test = y_test.reshape(-1)

X_tr_mean = X_train.mean(axis = 0)
X_tr_sd = X_train.std(axis = 0)

X_tr1 = (X_train)/X_tr_mean
X_test1 = (X_test)/X_tr_mean

X_tr_sub, y_tr_sub =  data_subsample(X_tr1, y_train, 5000)
X_val_sub, y_val_sub = data_subsample(X_test1, y_test, 2500)

sv_model = SVC()

param_grid = {
    'C': [0.01, 0.1, 1, 10, 100, 1000],
    'gamma': [0.01, 0.01, 1, 10, 100, 1000, 10000,]}

grid_search = GridSearchCV(sv_model, param_grid, scoring = 'roc_auc')

grid_search.fit(X_tr_sub, y_tr_sub)
param_select = grid_search.best_params_
print(param_select)
```

```
C, gamma = param_select['C'], param_select['gamma']
sv_model = SVC(C = C, gamma = gamma)
sv_model.fit(X_tr1, y_train)

y_pred = sv_model.predict(X_test1)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)

# left out testing
for recording_id in leave_out_ids:
  print('id:', recording_id)
  y = gen_y(recording_id)
  X = calc_features(recording_id)
  X_scaled = X/X_tr_mean
  y_pred = sv_model.predict(X_scaled)
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)
  print()
  filter_pred(y_pred)
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)

# neural network

%tensorflow_version 1.x
import tensorflow as tf
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Activation
from tensorflow.python.keras.optimizers import Adadelta
from tensorflow.python.keras.callbacks import EarlyStopping, ModelCheckpoint

model = Sequential()
model.add(Dense(512, activation='relu',input_dim =  X_tr.shape[1]))
model.add(Dense(1, activation="sigmoid"))
model.summary()

mcp_save = ModelCheckpoint('.mdl_wts.hdf5', save_best_only=True, monitor='val_loss', mode='min')

model.compile(loss="binary_crossentropy",
              optimizer=Adadelta(),
                                metrics = ['accuracy']
             )
```

```
history=model.fit(X_tr, y_tr,
                  batch_size=50,
                  epochs=1000,
                  verbose=1,
                  callbacks = [mcp_save],
                  validation_data=(X_val, y_val))

model.load_weights(filepath='.mdl_wts.hdf5')
y_pred = model.predict_classes(X_test, verbose = 0)
conf_mat = confusion_matrix(y_pred, y_test)
print(conf_mat)

# left out testing
for recording_id in leave_out_ids:
  print('id:', recording_id)
  y = gen_y(recording_id)
  print(len(y))
  X = calc_features(recording_id)
  print(X.shape)
  y_pred = model.predict_classes(X)
  print(len(y_pred))
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)
  print()
  filter_pred(y_pred)
  conf_mat = confusion_matrix(y, y_pred)
  print(conf_mat)
```