Seizure Detection

Rohit Bandaru
Cornell University
Ithaca, NY
rb696@cornell.edu

Aasta Gandhi
Cornell University
Ithaca, NY
apg67@cornell.edu

***Abstract*—Seizure detection using electroencephalography is an important problem in treating epilepsy. We present a machine learning approach to epileptic seizure detection. Through analysis of a EEG data, we can classify one second clips as ictal or non-ictal. This approach can be applied in a patient specific setting, given sufficient training data for each patient. Our models are quick to train and deploy, without requiring expensive computational resources. We used time series data, and wavelet transform coefficients with dimensionality reduction on support vector machine and random forest machine learning models. We were able to achieve over 80% area under ROC curve performance.**

***Index Terms*—seizure detection, machine learning, eeg**

I. Introduction

Epileptic seizures affect approximately 1% of the population [1]. They occur due to abnormal neuronal firings caused by involuntary alterations in activities such as movement and behavior [2]. Drugs have proven to be ineffective for 20% to 40% individuals with epilepsy, and patients continue to suffer seizures and adverse side effects post-surgery [3].

In such cases, seizure detection systems can be useful in detecting, monitoring and responding to seizure activity in real time. And, detection in responsive neurostimulation devices is often done with classification techniques using machine learning on electroencephalography (EEG) data collected in real time [4]. Generalized classification in software [5] [6] and hardware [7] have shown high sensitivity and accuracy rates, but tend to lose information on variability between patients [2].

Consequently, to capture differences in patients we developed a patient specific model to detect ictal (seizure) and non-ictal (non-seizure) events in seven distinct patients through various binary classification techniques. Data was pre-processed, channels were selected and used for feature extraction. Classifiers used included support vector machines, bagging and random forests. An overview of our methodology is provided in Figure 1.

II. Preprocessing

A. Dataset

The EEG data we used had seven patients, each with hundreds or thousands of labeled ictal and non-ictal EEG data. Some patients had data with 5000 Hz sampling frequency, others with 500 Hz sampling frequency. The number of channels also varied between patients. This variance in data between patients, and the fundamental differences in epilepsy between patients made a universal trained classifier infeasible. We extracted features and trained a machine learning model independently for each patient.

The EEG data was too high dimensional to train a machine learning classifier effectively. Therefore, we had to pre-process and extract specific features to obtain specific information to use in classification.

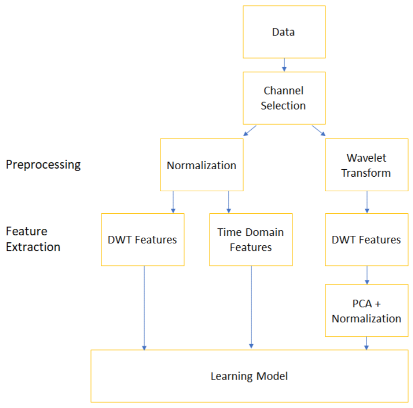

Fig. 1. Overview of methods.

Patient	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Channels	96	56	16	88	47	88	96
F_s	5000	500	500	500	5000	500	500
Ictal	218	191	296	424	180	313	307
Non-Ictal	600	900	900	1200	1800	2100	2400
Test	932	1243	2401	2185	3058	4117	4524

TABLE I
Overview of patient information.

B. Channel Selection

Looking at the EEG data, it became clear that only a few channels have useful information for seizure detection. Rather than extracting features from all channels, we found it more efficient and effective to select a few of the more useful channels for analysis. Additionally, this decreased computation time of the model.

We chose to select the channels with the highest variance. The intuition behind this was that channels with more variance have more information than a channel that is relatively flat.

Channel selection was done by randomly sampling 200 ictal and non-ictal points, finding the n channels with highest variance, and then taking the n channels that appear most often in the selected data points.

III. Feature Extraction

A. Frequency Domain

For the frequency domain, we chose to apply the Discrete Wavelet Transform (DWT) to the selected channels because DWT provides optimal time-frequency resolution across a large frequency range and is now frequently used in seizure detection classification [8]. A wavelet is an oscillation localized in frequency and time that vanishes quickly. Discrete wavelets are formed by discretizing the scale (*a*) and translation (*b*) parameters of the continuous wavelet:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi(\frac{t-b}{a}), a, b \in \Re \tag{1}$$

The parameters are commonly based on powers of two where $a = 2^j$ and $b = k2^j$ where $j, k \in Z$. The full form of DWT can be written as:

$$d_{j,k} = \int_{-\infty}^{\infty} s(t) 2^{-j/2} \psi(2^{-j}t - k) dt \tag{2}$$

$$d_{j,k} = \langle s(t), \psi_{j,k}(t) \rangle \tag{3}$$

Here, $d_{j,k}$ are the wavelet coefficients which will be used to form our feature vectors. *j* corresponds to the level of the wavelet, and *k* corresponds to the location. When processing with DWT, different "mother wavelets" produce different coefficients. The Daubechie family was what we used in this model because it was the most common and relevant feature in literature for seizure detection [2] [8]. The decomposition level of the DWT corresponds to specific frequency bands where greater levels provide greater signal resolution. Frequently, the range of levels used is 2-5, and We chose a decomposition level of 5 whose frequency range corresponds to 1.25-16 Hz and corresponds to theta, delta, alpha and beta brain waves.

We also generated the absolute value, standard deviation, energy and variance from the DWT coefficients [8], but found that these features did not make a difference on performance. To avoid overfitting on features, we chose to just use the coeffcients themselves.

B. Time Domain

For the time domain, we chose a few features which were shown to have a sensitivity of higher than 80% in a study that evaluated discriminative performance of 65 previously reported, distinct features [9]. This study evaluated sensitivity, specificity and area under the sensitivity-specificity curve which we wanted to maximize. We initially chose 9 out of the 15 optimal time domain features from the study including line length, energy/power, maximum, minimum, mean, variance, skew, kurtosis and entropy. However, we narrowed down to 6 features (described below) that produced the higher accuracy on our model.

1) Line Length: Line length was defined as the sum of the distances between a point and its previous point:

$$ll = \sum_{n=1}^{ndatapoints} |x_n x_{n-1}| \tag{4}$$

2) Energy: Energy was defined as the sum of the square of each time point:

$$en = \sum_{n=1}^{ndatapoints} x_n^2 \tag{5}$$

3) Variance: Variance describes how far a set of points are spread out from their mean. It is defined as:

$$va = \mathbb{E}[(X - \mu)^2] \tag{6}$$

4) Skew: We used the skew function in the SciPy Statistics library. Skewness describes how symmetrical the distribution is. Skewness is defined as:

$$sk = \mathbb{E}[(\frac{X - \mu}{\sigma})^3] \tag{7}$$

5) Kurtosis: We used the kurtosis function in the SciPy Statistics library. Kurtosis describes the peakedness or flatness, and measures the probability in the tails of the distribution. When this was added as a feature, we noticed the accuracy of our model improved suggesting significant peak or tail differences between ictal and non-ictal events. Kurtosis is defined as:

$$ku = \mathbb{E}[(\frac{X - \mu}{\sigma})^4] \tag{8}$$

6) Power: To calculate power, we used Welchs method which estimates the power spectral density by dividing the data into overlapping segments using the welch function with default parameters in SciPys Signal library. Then, we identified frequency bands where we thought we might see peaks (12-30 Hz or 100-600 Hz ranges), and calculated total power for those corresponding frequencies.

IV. Machine Learning

A. Dimensionality Reduction

We applied dimensionality reduction in the form of principal component analysis (PCA). We used the Scikit-Learn implementation. This dimensionality reduction was applied to the wavelet coefficients. Each channel yielded about five hundred coefficients. For multiple channels this yields very high dimensional data. Using data of this high dimension in a classifier would result in over-fitting. PCA would lower the dimension, normalize the data, and reduce the noise [10].

We applied PCA to the wavelet coefficients and concatenated them with the time series feature vector. The number of components in the PCA is a hyper-parameter to tune. We found that we were able to get high performance with a low number of components (5). This shows that the wavelet coefficients lie in a small subspace.

B. Support Vector Machines

The first classifier we used with strong performance was Support Vector Machines (SVM) with a RBF kernel. We chose this model because SVMs perform well on datasets with imbalanced class labels.

This requires tuning gamma and C as hyper-parameters. Through validation we found hyper-parameter values that are most effective for each patient.

C. Ensemble Methods

1) Bootstrap Aggregation: Bootstrap Aggregation, or bagging is used to create multiple models from a single training set. A random number of sub-samples from the training set are created with replacement, and then each sub-sample is trained on a pre-defined weak classifier (e.g decision trees). Then, the average prediction from each trained model is calculated. We initially chose this method because a lot of variance exists within specific channels of patients, and bagging handles this by training on high variance algorithms on training data that is constantly changing.

We created our model using the BaggingClassifier in the SciKit Learn library. The base estimator used was the decision tree because it is a weak classifier with high variance. We calculated the number of estimators to use for each patient by looping over a range of 10 to 150, and found that for all patients, the highest accuracy was obtained at 10 estimators. For patients with a larger dataset size (i.e patients 5-7), 30 estimators also produced over 97% training accuracy. The model was also bootstrapped so that samples were always drawn with replacement.

2) Random Forests: Random Forests are used on larger datasets and are an improvement to bagging because random forests decrease correlation between subtrees. Essentially, when decision trees are being split, the algorithm looks at all possible variables to find an optimal split.

We created our model using the RandomForestClassifier in the SciKit Learn library. The base estimator was a decision tree with a maximum depth of 3, minimum number of samples required to split was 2 and the maximum number of features used for each tree was 20. The number of estimators for all patients was 30.

For this classifier, we also weighted our samples since all patients had a greater number of non-ictal to ictal events. We determined weights by calculating the ratio of ictal to non-ictal events, and assigning the greater weight to the ictal events. We found that most of the patients had approximately 1 ictal event for every 3 non-ictal events (25% of data was ictal), and for patients with larger datasets (patients 6-7), the ratio was even smaller. In the latter case, we default assigned weights of 0.75 for ictal and 0.25 for non-ictal as this gave the highest training and validation accuracy.

D. Scoring

We used the area under (AUC) the ROC (receiver operating characteristic curve) curve as the metric to evaluate our models. The data is highly imbalanced, with far more negative points than positive. This makes classification accuracy highly flawed. The ROC AUC score is a better metric for our purpose.

Model	**Freq. Domain Features**	**Time Domain Features**	**Parameters**	**Score**
SVM	DWT Coeff. w/ PCA	line length, energy, power, variance, skew, kurtosis	C = 100, gamma = 1	0.834
Random Forest	DWT Coeff. w/ PCA	line length, energy, power, variance, skew, kurtosis	200 Decision Trees, depth of 2, 20 max features	0.849

TABLE II
BEST PERFORMING MODELS.

E. Validation

The machine algorithms we used had tunable parameters that affect performance. Random forest is quite resilient to different hyper-parameter settings, but careful tuning is crucial for SVMs.

We used two methods of validation. We began by splitting the training data into train and validation sets, with 80% being in training. After training, the model was evaluated and scored on the validation set. We used the telescopic search method in order to find the hyper-parameters that would maximize validation performance. However, we refrained from being too specific in this tuning to avoid over-fitting on the validation set.

We also used K-fold cross validation (k=10) using the SciKit Learn library. This method is slower, but a stronger validation scheme that mitigates over-fitting.

V. Results

Our final two models were the Support Vector Machine and Random Forests models which achieved over 80% ROC AUC on Kaggle.

Random forests with 200 decision tree estimators of depth 2, and a maximum features of 20 was one of the best performing models with an ROC AUC of 0.849. The feature vectors used were DWT coefficients reduced with PCA to 250 components, and the 6 time domain features described in the methods. Overall, other variations of estimators and features resulted in an average ROC AUC of approximately 0.75 (Table II).

We used SVMs with the same feature vectors and were able to get similar performance on the test set. The best performing model had a score of 0.835 (Table II).

Bagging models were also attempted, but the average kaggle score was consistently less than 0.75 so we chose not to tune the model further, and switched to random forests.

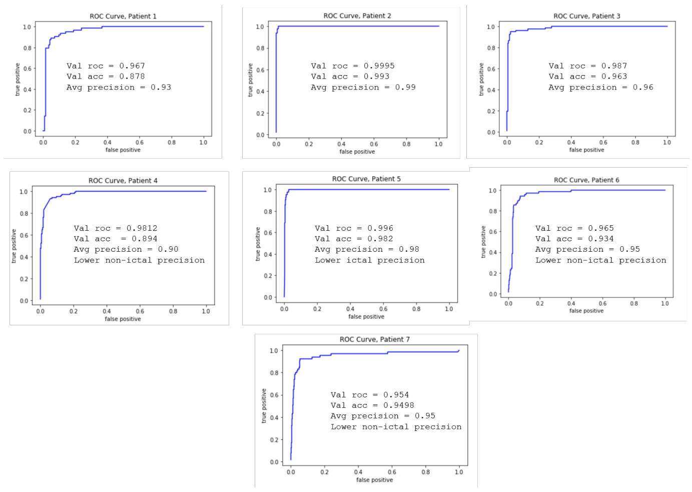

Fig. 2. ROC curves for patients 2, 3 and 5 trained on SVM model.

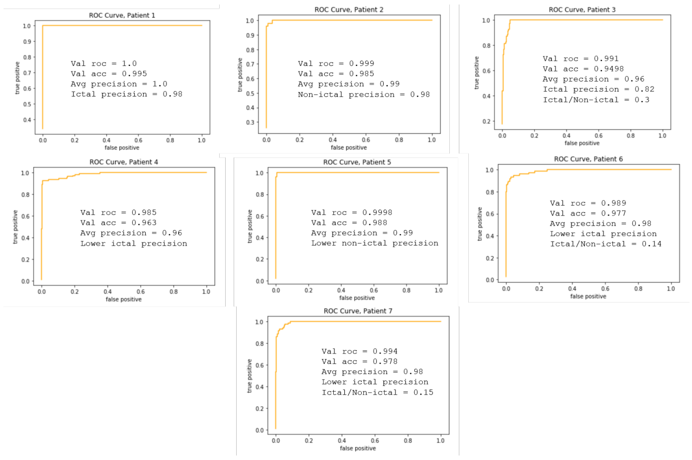

Fig. 3. ROC curves for all patients trained on random forests model.

VI. Discussion and Conclusions

We found that effective seizure detection can be accomplished with a small number of features. Although there may be dozens of EEG channels, only 3-5 need to be used. Given the channels, a small number of features can be used. Although we used many time domain and frequency domain features, decent classification performance can be accomplished with fewer.

For each model, we also generated ROC AUC curves, and evaluated the precision and recall for each patient. We looked at recall because the ROC curve plots the true positive rate (recall) against the false positive rate, and we looked at precision to determine how relevant all the data really was. As we tried to maximize precision, we saw that recall decreased since they are proportional.

For the SVM model, both precision and recall were above 0.9 for all the patients for ictal events. But, as the number of patient data points increased (patients 6, 7), we see that precision decreased for non-ictal events. A likely explanation for this is that we did not use any weighting on our data so the model was able to capture irrelevant non-ictal events. This could cause the non-ictal precision to decrease and the recall to increase. Overall, we saw that as the number of data increased and the amount of ictal events decreased, the AUC score patient wise also decreased (Figure 2).

The AUC curves for the Random Forests model showed similar trends. However, the validation ROC and accuracy scores were consistently over 0.95. For patients such as 3, 4, 6 and 7, we see that the precision for ictal events was lower. This suggests that the recall would be higher and thus, the ROC AUC should also be greater in comparison to the SVM which is observed (Figure 3). Both quantitatively and qualitatively, the random forests model performed slightly better, and was able to handle weighting and changing data size better as well.

Major limitations in our models to account for in the future include the types and number of features, amount of data, skewed data and channel selection. Additional features that we could use include channel correlation matrix, and more advanced time domain features such as entropy. To handle the number of features, it would be necessary to test our model on variations of different groups of features to see how relevant each feature is. To handle skewed data, we would need to look into more complex weighting techniques rather than thresholding over the ratio of ictal to non-ictal events.

Additionally, we generalized our models for all the patients, and tuned the overall model patient wise. Here, we could also try designing unique models for each patient and be strictly patient specific. However, our method also worked sufficiently as we saw significant improvements patient wise between the SVM and Random Forest models.

We could also try other machine learning techniques including gradient boosting and finely tuned neural networks for each patient. We could also try using more channels and more expressive features such as processing over DWT coefficients at the expense of computational efficiency.

Overall, we observed that both Random Forests and Support Vector Machines are strong classifiers for skewed seizure detection data, and we achieved over 80% accuracy on both the models.

Acknowledgments

We would like to thank Professor Mahsa Shoaran and Hanchen Jin for their guidance on the project.

References

[1] Leppik I. Contemporary diagnosis and management of the patient with epilepsy. Newtown, PA: *Handbooks in Health Care* 2000.
[2] Shoeb A, Edwards H, Connolly J, Bourgeois B, Treves ST, Guttag J. Patient-specific seizure onset detection. *Epilepsy & Behavior* 2004;5:483-498.
[3] Jette N, Reid AY, Wiebe S. Surgical management of epilepsy. *CMAJ: Canadian Medical Association Journal.* 2014;186(13):997-1004.
[4] Ramgopal S, Thome-Souza S, Jackson M, Kadish NE, Fernandex IS, Klehm J, Bosl W, Reinsberger C, Schachter S, Loddenkemper T. *Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy* 2014;37:291-307.
[5] Gotman J. Automatic recognition of epileptic seizures in the EEG. *Electroencephalogr Clin Neurophysiol* 1982;54:53040.

[6] Webber WRS, Lesser RP, Richardson RT, Wilson K. An approach to seizure detection using an artificial neural network. *Electroencephalogr Clin Neurophysiol* 1996;98:25072.

[7] Raghunathan S, Gupta SK, Markandeya HS, Roy K, Irazoqui PP. A hardware-algorithm co-design approach to optimize seizure detection algorithms for implantable applications. *Journal of Neuroscience Methods* 2010; 193:106-117.

[8] Chen D, Wan S, Xiang J, Bao FS. A high-performance seizure detection algorithm based on Discrete Wavelet Transform (DWT) and EEG. *PLoS ONE* 2017;12(3): e0173138.

[9] Logesparan L, Casson AJ, Rodriguez-Villegas E. Optimal features for online seizure detection. *Med Biol Eng Comput* 2012;50:659-669.

[10] Subasi A, Gursoy MI. EEG signal classification using PCA, ICA, LDA and support vector machines. *Expert Systems with Applications* 2010;37:8659-8666.

In [ ]:
```
import numpy as np
import pywt
from pywt import wavedec
import scipy.io as spio
from scipy import signal
import scipy.signal as sg
import scipy.stats as stats
'''
data: (t,c)
'''

def extract_feature(data):
    n_t, n_c = data.shape

    feature = []

    ll = line_length(data).flatten()
    en = energy(data).flatten()
    va = variance(data).flatten()
    po = power(data).flatten()
    ku = kurtosis(data).flatten()
    sk = skew(data).flatten()
    feature = np.append(feature, [ll,en,va,po,ku,sk])
    return feature

def extract_feature_coeff(data):
    n_t, n_c = data.shape

    feature = []

    for channel_number in range(n_c):
        coeffs = wavedec(data = data[:, channel_number], wavelet='db4', 
mode = 'symmetric', level = 5)
        cA5, cD5, cD4, cD3, cD2, cD1 = coeffs
        for c in coeffs:
            feature = np.append(feature,c)
    return feature

# not axis function

def abs_val_coeff(data):
    return np.mean(np.abs(data))

def stdev_coeff(data):
    return np.std(data)

def energy_coeff(data):
    #m, n = data.shape
    energy = np.sum(np.square(np.abs(data)))
    return energy

def variance_coeff(data):
    variance = np.sum(np.square(data - np.mean(data)))
    return variance
```

```
# axis functions
def kurtosis(data):
    return stats.kurtosis(data, axis=0, bias=False)

def skew(data):
    return stats.skew(data, axis=0, bias=True)

def stdev(data):
    return np.std(data, axis = 0)

def line_length(data):
    # subtract previous data point from every point to get the line leng
th
    data[1:,:] = np.abs(data[1:,:] - data[0:-1,:])
    line_length  = np.sum(data, axis = 0)
    return line_length

def energy(data):
    energy = np.sum(np.square(np.abs(data)), axis = 0)
    return energy

def variance(data):
    variance = np.sum(np.square(data - np.mean(data, axis = 0)), axis =
0)
    return variance

def power(data):
    m, n = data.shape
    power = np.zeros(n)
    for j in range(n):
        # f contains the frequencies that correspond to the powers in Px
x_den
        f, Pxx_den = signal.welch(data, m)
        # sum up the powers in the frequency bands
        band_f = np.where(np.logical_or(np.logical_and(f>=12, f<=30), np
.logical_and(f>=100, f<=600)))
        # add the power to the channel
        power[j] = np.sum(Pxx_den[band_f])

    return power
```

In [ ]:
```
import feature_extractor as fe
from os import listdir
import scipy.io as spio
import numpy as np

'''
data: (m,n)
'''
def directory_data(path, channels):
    files = listdir(path)
    print(str(len(files))+" number of points")
    data = None

    for i, file in enumerate(files):
        print(str(i+1)+"/"+str(len(files))+": extracting "+file)
        file_data = spio.loadmat(path+file)["data"]
        vec = fe.extract_feature(file_data[:,channels])
        if data is None:
            data = np.zeros((len(files),vec.shape[0]))
        data[i,:] = vec
    print("directory data shape", data.shape)
    return data

def get_data(patient_number, channels):
    ictal_train = directory_data("data/patient_"+str(patient_number)+"/i
ctal train/", channels)
    non_ictal_train = directory_data("data/patient_"+str(patient_number)
+"/non-ictal train/", channels)

    m_ictal = ictal_train.shape[0]
    m_non_ictal = non_ictal_train.shape[0]
    data = data = np.vstack([ictal_train, non_ictal_train])
    labels = np.hstack([np.ones((m_ictal)),np.zeros((m_non_ictal))])
    for i in range(int(m_non_ictal/m_ictal) -1):
        data = np.vstack([ictal_train, data])
        labels = np.hstack([np.ones((m_ictal)),labels])
    return data, labels

def directory_data_coeff(path, channels):
    files = listdir(path)
    print(str(len(files))+" number of points")
    data = None

    for i, file in enumerate(files):
        print(str(i+1)+"/"+str(len(files))+": extracting "+file)
        file_data = spio.loadmat(path+file)["data"]
        vec = fe.extract_feature_coeff(file_data[:,channels])
        if data is None:
            data = np.zeros((len(files),vec.shape[0]))
        data[i,:] = vec
    print("directory data shape", data.shape)
    return data

def get_data_coeff(patient_number, channels):
        #dir_name = "C:/Users/Aasta/Documents/CU SP 18/ECE 5040/"
    ictal_train = directory_data_coeff("data/patient_"+str(patient_numbe
r)+"/ictal train/", channels)
    non_ictal_train = directory_data_coeff("data/patient_"+str(patient_n
umber)+"/non-ictal train/", channels)

    m_ictal = ictal_train.shape[0]
    m_non_ictal = non_ictal_train.shape[0]
    data = np.vstack([ictal_train, non_ictal_train])
    labels = np.hstack([np.ones((m_ictal)),np.zeros((m_non_ictal))])
    for i in range(int(m_non_ictal/m_ictal) -1):
        data = np.vstack([ictal_train, data])
        labels = np.hstack([np.ones((m_ictal)),labels])
    return data, labels

# sample datapoints to get the most useful channels for a particular pat
ient
def directory_data_sample(path, channels_num, sample):
    import random
    files = listdir(path)
    random.shuffle(files)
    data = None

    for i, file in enumerate(files):
        if(i>sample):
            break
        file_data = spio.loadmat(path+file)["data"]
        vec = np.argsort(-np.var(file_data, axis=0))[0:channels_num]
        if data is None:
            data = np.zeros((len(files),vec.shape[0]))
        data[i,:] = vec
    return data

def get_channels(patient_number, channels_num, sample):
    ictal_train = directory_data_sample("data/patient_"+str(patient_numb
er)+"/ictal train/", channels_num, sample)
    non_ictal_train = directory_data_sample("data/patient_"+str(patient_
number)+"/non-ictal train/", channels_num, sample)
    data = np.vstack([ictal_train, non_ictal_train])
    return np.argsort(np.bincount(data.astype(int).flatten()))[-channels
_num:]

# split data set and labels randomly on the split, which is the proporti
on of points that will go in the training set
def train_val_split(data,labels,split):
    # shuffle data
    rand_idx = np.random.permutation(labels.shape[0])

    labels = labels[rand_idx]
    data  = data[rand_idx,:]

    m = data.shape[0]
    train_m = int(m*split)
    val_m = m - train_m

    train_data = data[:train_m,:]
    train_labels = labels[:train_m]

    val_data = data[train_m:,:]
    val_labels = labels[train_m:]

    return train_data, train_labels, val_data, val_labels
```

In [ ]:
```
import feature_extractor as fe
from os import listdir
import scipy.io as spio
import numpy as np
import dataset
```

In [ ]:
```
patient_number = 2
channels = dataset.get_channels(patient_number, 30, 200)
print(channels)
data, labels = dataset.get_data(patient_number, channels)
datac, labelsc = dataset.get_data_coeff(patient_number, channels)

print(data.shape)
print(datac.shape)
```

In [ ]:
```
print(data.shape)
print(labels.shape)
data = np.nan_to_num(data)
std = np.std(data, axis=0)
mean = np.mean(data, axis=0)
#max1 = np.max(data, axis=0)
#data2 = data/max1
data2 = np.nan_to_num((data-mean)/std)

# perform PCA on frequency coefficients
from sklearn.decomposition import FastICA
from sklearn.decomposition import PCA
pca = PCA(n_components=250, whiten = True)
pca.fit(datac)
data2c = pca.transform(datac)

# concatenate time domain and frequency domain data
new_data = np.hstack([data, data2c])

train_data, train_labels, val_data, val_labels = dataset.train_val_split
(np.nan_to_num(new_data.astype(np.float64)), labels, .75)

print(train_data.shape)
print(train_labels.shape)
print(val_data.shape)
print(val_labels.shape)
'''
import matplotlib.pyplot as pyplot
plt.scatter(data[:,0],data[:,1],c=labels)
plt.title("2 component PCA")
plt.show()
'''
```

In [ ]:
```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn import metrics

#### SVM
#clf = SVC(C=100, gamma=1, kernel='rbf',
 #max_iter=10000000, probability=True)
clf = RandomForestClassifier(n_estimators=200, max_depth=2, max_features
=20)
clf.fit(train_data, train_labels)

print("training")
print(metrics.classification_report(train_labels,clf.predict(train_data
)))
print(metrics.roc_auc_score(train_labels, clf.predict_proba(train_data)
[:,1]))
print("\nvalidation")
print(metrics.classification_report(val_labels,clf.predict(val_data)))
print(metrics.roc_auc_score(val_labels,clf.predict_proba(val_data)[:,1
]))
print(metrics.accuracy_score(val_labels,clf.predict(val_data)))
print("\n")

fpr, tpr, thresholds = metrics.roc_curve(val_labels, clf.predict_proba(v
al_data)[:,1], pos_label=1)
import matplotlib.pyplot as plt
plt.plot(fpr,tpr)
#print(clf.feature_importances_[:48])
```

In [ ]:
```
path = "data/patient_"+str(patient_number)+"/test/"
files = listdir(path)
print(str(len(files))+" number of points")

import os
import csv
csv_path = 'patient_roc' + str(patient_number)+'.csv'
f = open(csv_path, "w")
f.truncate()
f.close()
with open(csv_path, 'a') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoti
ng=csv.QUOTE_MINIMAL)
    for i, file in enumerate(files):
        # weird MACOS file to ignore
        if file == ".DS_Store":
            continue
        print(str(i+1)+"/"+str(len(files))+": extracting "+file)
        file_data = spio.loadmat(path+file)["data"]
        vec = np.nan_to_num(fe.extract_feature2(file_data[:,channels]))
        vec = np.nan_to_num((vec-mean)/std)

        vec2 = np.nan_to_num(fe.extract_feature_coeff(file_data[:,channe
ls]))
        vec2 = np.nan_to_num(pca.transform(vec2.reshape(1,vec2.shape[0
])).astype(np.float32))

        vec = np.append(vec,vec2)
        vec = vec.reshape(1,vec.shape[0])

        filename = file[:-4].replace("test_", "")
        score = clf.predict_proba(vec).reshape(-1, 1).T[0,1]
        pred = [filename, score]
        print(pred)

        spamwriter.writerow(pred)
```

In [ ]:

```
###concatenate all files
import pandas as pd
files = ["patient_roc1.csv","patient_roc2.csv","patient_roc3.csv",
         "patient_roc4.csv","patient_roc5.csv","patient_roc6.csv","patie
nt_roc7.csv"]

out = open("patient.csv", "w")
out.truncate()
out.close()

with open("patient.csv", 'a') as csvfile:
    writer = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=c
sv.QUOTE_MINIMAL)
    writer.writerow(["id","prediction"])
    for file in files:
        with open(file, 'r') as csvfile:
            reader = csv.reader(csvfile, delimiter=',', quotechar='|')
            for row in reader:
                writer.writerow(row)
```