U Penn & Mayo Clinic's Seizure Detection Challenge

Ishan Talukdar, Nathan Moore, and Alexander Sood

Location: Berkeley, CA
Email: italukd@berkeley.edu, nmoore@berkeley.edu, asood@berkeley.edu

1. Summary

We outline here the general procedure used by our team *cdipsters,* which placed 3rd in this Kaggle contest. The predictive solution was generated using the Extremely Randomized Trees algorithm developed by Geurts, Ernst, and Wehenkel [1], and implemented in python using scikit-learn's ExtraTreesClassifier [2]. For each of the 12 subjects our algorithm generated between 426 and 1770 features from each clip (depending on the number of channels) and used these features to train two ExtraTreesClassifier forests consisting of 1000 trees each. The first forest was used to distinguish between seizure and not-seizure clips, while the second was used to distinguish between early-seizure and not-early-seizure clips.

2. Features Selection

Initially we started with a basic set of global features formed by calculating a quantity of interest averaged over all electrodes, and then estimating statistical properties of that quantity. These features included properties such as the maximum, mean and variance of the readings over the 1 second span, and similar quantities for the power spectrum. Thus, we were only using global features over the entire clip, averaged over all electrodes. Before selecting these features, clips were down sampled to decrease noise. A frequency of 100 Hz found to be optimal for noise reduction without loss of meaningful information. This set of 12 global features was used to train two classifiers on the entirety of the training data, and were able to achieve an AUC score of 0.84 on the Kaggle leaderboard.

We then realized that it would make more sense to run subject specific classifiers, as in a real world application a machine learning algorithm would most likely be trained on individual subjects so as to maximize the effectiveness of detecting early seizures in that subject. Thus we decided to run our algorithm over each subject's training and test data individually, which boosted our score up to 0.855 (the same exact algorithm was used for each subject).

It soon became clear from visual inspection of the data that different groups of electrodes produced vastly different signals, presumably due to placement of the electrodes with respect to the origin of the seizure. Features thus calculated by simply averaging over all the electrode signals failed to capture the whole picture. We therefore used the individual channels of each

subject's data clips to generate features. This effectively multiplied the number of features per clip by the number of channels and boosted our score up to 0.9.

After considering the importance of time dependence of a seizure, especially at the onset, we realized that it would be a good idea to also generate features from the first and second derivatives of the time series data. This effectively tripled our feature size and improved our score to 0.94. Additionally, to take advantage of correlations between electrodes, a feature was added for the mean of the covariance between channels, taken over all possible combinations of two electrodes. This improved the AUC score to 0.95. At this point we were using 24 channel features and 42 global features per clip, giving us a total of between 426 and 1770 features, depending on the number of channels.

Lastly, we discovered that our algorithm seemed to perform best, in terms of identifying early seizure clips, when we raised the early cut-off from 15 to 19 seconds. This change in labeling of the training data increased the AUC score for our model to 0.96.

3. Modeling Techniques and Training

Our model is based on the ExtraTreesClassifier algorithm, which, like many tree-based algorithms, produces a result by averaging over the outputs of an ensemble of decision trees. In this algorithm, trees are grown from a randomly selected subset of the training data (of size square root the number of training samples) by randomly selecting from the list of all features a subset (of size square root of the number of features) to consider for splitting the current node. For each of these features, a cut point is chosen at random, and the cut that performs the best is used to split the node. For each subject, the features described in Section 2 were generated from the subject training data and used to train two ExtraTreesClassifiers, one to discriminate between ictal and interictal clips and the other to separate early-seizure clips from everything else (late-seizure + non-seizure). The number of trees used in each classifier was set to 1000, and the minimum number of samples required to split a node was set to 1.

Several other classifiers, including gradient boosted trees and embedded trees were tested as alternatives to the ExtraTreesClassifier approach. Despite giving good results on our cross-validations sets, they ultimately seemed to overfit the training data and did not achieve AUC scores beyond 0.9 on the test data.

4. Code Description

The code used in this contest is available at https://github.com/asood314/SeizureDetection. It is written in python 2.7 and uses python libraries *numpy, scipy, pandas, matplotlib, os, itertools, and scikit-learn*. The code was tested on a Mac running OS X 10.9.4 and a PC running Windows 8.1. The entire code takes approximately 12 hours to run on a 2009 iMac 3.06 GHz Intel Core 2 Duo with 16GB of RAM and around 10 hours on an i5 PC with 4 GB of RAM.

Program description:

features.py: contains functions to calculate features from time series data. Each function

takes as input a pandas data frame corresponding to a single clip with one column for the readings from each electrode and one column giving the time relative to the beginning of the clip for each set of readings. It is assumed that the first column of the data frame is the time column. The output is a data frame with columns for each feature calculated. A dictionary associates feature functions with names, allowing the user to choose which features to include.

The flexibility of this approach is useful for quickly examining new features and relationships between small groups of features, but it adds significant overhead to generating the full set of features used in this solution. When training the classifiers on the whole dataset, this approach is effectively abandoned by using a single function to calculate all of the features. This single function is still far from optimal but was sufficiently fast to meet our needs, so no further effort was made to speed up the calculation.

utils.py: contains functions to load the data, train and test algorithms, and make submission files. Training samples are loaded with **loadTrainAndValidationSamples**. This function takes three arguments: a data selector, a list of functions to use for calculating features, and a common frequency to which each clip is down sampled. The data selector is a list of lists in which each sub-list specifies a subject and what fraction of that subject ictal and interictal data to use for validation. For instance, an input of "[['Patient_1,0.5,0.2'],['Dog_3',0,0]]" would load all training data for Patient 1 and Dog 3, but half of the ictal clips and 20% of the interictal clips for Patient 1 (selected at random) would be set aside for validation. The common frequency argument is optional, and none of the feature calculation is dependent on receiving input at a particular sampling rate. However, as previously mentioned, we found that the best results were achieved with a rate of 100 Hz.

The output of **loadTrainAndValidationSamples** is a dictionary with the keys "train" and "validation" corresponding to data frames for the training and validation samples. The data frames are organized such that each row corresponds to one clip with one column for each feature. Three extra columns are tacked on after the feature columns to specify the latency, whether or not the clip is from a seizure, and whether the clip is occurring early in the seizure.

Test samples are loaded with **loadIndivTestSamples**, which works the same way as **loadTrainAndValidationSamples** with a few small differences. Although the arguments taken are same, there is no splitting of the test sample. Any numbers supplied for splitting the sample will be ignored. Columns giving the target data are naturally not included since this is not known for the test data. Instead, there is one extra column providing the filename of each test clip for use in writing the submission file. Finally, there is no need to put the output in a dictionary since there is only one data frame to return.

The functions **loadData**, **downSample**, and **convertToFeatureSeries** are used by **loadTrainAndValidationSamples** for loading time series data from .mat files into a dataframe, reduce the sampling rate, and convert from time series data to feature data. **loadData** takes a filename and returns a data frame with the appropriate format to be used as input to the functions in features.py. The latency of the clip is stored by using it as the starting point for the time column. There is also an optional argument for specifying the latency that is only used for the purpose of plotting interictal data over a period spanning multiple clips.

The arguments to **downSample** are a time series data frame and a reduction factor specifying the number of adjacent readings that should be averaged into a single measurement. The returned data frame has the same format as the input but with the reduced sampling rate.

convertToFeatureSeries takes a time series data frame, a list of functions for calculating the features, a boolean specifying whether the clip is a seizure clip, an integer specifying the latency, a boolean specifying whether the clip is a test clip, and the corresponding file name of the test clip. It returns a data frame of the format output by **loadTrainAndValidationSamples** or **loadIndivTestSamples** as appropriate but with just one row corresponding to the given clip. The data frames for each clip are concatenated afterward.

Training of the ExtraTreesClassifier is done in the **trainDoubleForest** function. It takes the "train" data frame from **loadTrainAndValidationSamples** as input. Two ExtraTreesClassifiers are initialized and trained to determine seizure/non-seizure and early/non-early, respectively. The two trained classifiers are returned in a dictionary labeling them as "seizure" and "early."

Predictions are generated with **testProbs**. The inputs are a list of trained classifiers and the output of **loadIndivTestSamples**. It returns a two-dimensional array of predicted probabilities for seizure and early seizure.

The function **makeSubmit** writes the submission file, using as input the array of predicted probabilities and the test sample data frame, which contains the filenames of the clips. Nothing is returned, but a file called "submission.csv" is written containing the predictions for each clip.

test.py: the "main()" function. This script loops through the list of subjects. For each subject, it loads the subject training data, trains two ExtraTreesClassifiers, loads the test data, and stores the classifiers output. Once finished, the predictions are written to a .csv file for submission.

5. Dependencies

The code uses methods from the following python libraries:

- numpy
- scipy
- pandas
- os
- itertools
- scikit-learn
- matplotlib

6. How To Generate the Solution (aka README file)

3rd place submission for the U Penn & Mayo Clinic's Seizure Detection Challenge

1. Specify the location of subject data files in the variable *dataDirectory* in **utils.py**.
2. Include the list of subjects to train and predict in the variable *dataSelector* in **test.py**.
3. Run **test.py**. This will train each of the specified subjects and save the predictions of the test sample in a csv file 'submission.csv'.

7. Figures

Initially we started by stacking together the time clips of the ictal and interictal datasets of each subject next to each other, to get an idea of these signals. Plotting the features calculated for such time series then helped us get insights into how important they would be. An example is provided in the figures below for the case of Dog 1.

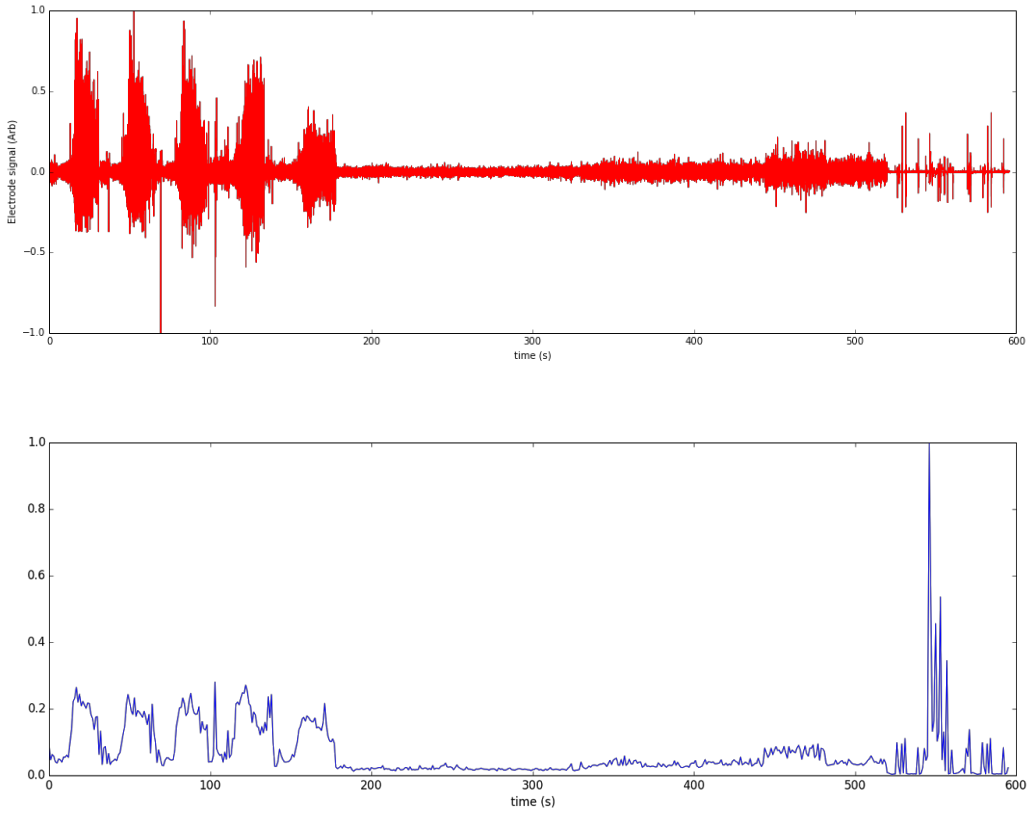

Fig. 1. (a) Signal collected by electrode 14 of Dog 1 showing ictal segments followed by interictal segments. (b) Standard deviation of the amplitudes for this electrode with the clips arranged in the same order as in (a).

References

1. P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. Machine Learning, 36(1):3-42, 2006.
2. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html