U Penn & Mayo Clinic's Seizure Detection Challenge

Ishan Talukdar, Nathan Moore, and Alexander Sood

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

1. Summary

We outline here the general procedure used by our team *cdipsters,* which placed 3rd in this Kaggle contest, and document the model code used for this. The predictive solution was generated with an ensemble of extra-trees classifiers. A set of common features were gathered from the training sets of each patient. These features were used to train two sets of extra-trees[1] classifiers (each consisting of 1000 trees), one for distinguishing the seizure from non-seizure cases, and the other to identify an early seizure.

2. Features Selection

We started with a basic set of global features formed by calculating a quantity of interest for each electrode and then taking either the maximum, mean, or variance of that quantity over all electrodes. The quantities used included the maximum reading over the 1 second span, the mean reading, the variance of the readings, and similar quantities for the power spectrum. Before selecting these features, clips were down sampled to decrease noise. A frequency of 100 Hz was found to be optimal for noise reduction without loss of meaningful information. This simple set of features was used to train two classifiers on the entirety of the training data, reaching AUC scores up to 0.84.

However, it was clear from visual inspection of the data that different groups of electrodes produced vastly different signals, so that simply averaging over them or taking the maximum value failed to capture the whole picture. Presumably, this is due to the placement of electrodes with respect to the origin of the seizure. Quantities for individual channels were added as features, and we went from two classifiers to two classifiers per patient. Having features for the individual electrodes increased the AUC score for the model from 0.84 to 0.9.

Features were also added for the maximum, mean, and variance of the first and second time derivatives of the readings in each electrode. These proved to be particular important for identifying early-seizure clips. To take advantage of correlations between electrodes, a feature was added for the mean of the covariance between two channels, taken over all possible combinations of two electrodes. Along with an adjustment in training of the early vs. non-early

classifiers (discussed in Section 3), these new features raised the model's AUC score from 0.9 to 0.96 (on the 15% of test data for which the score was public).

3. Modeling Techniques and Training

Our model is based on the extra-trees 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 by randomly selecting from the list of all features a subset 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 patient, the features described in Section 2 were generated from patient data and used to train two extra-trees classifiers, 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. Other parameters for the classifiers (how many features to select, method for evaluating cut perform, maximum tree depth, etc.) were left at the default values for the python ExtraTreesClassifier[2] class.

One key realization that helped in the identification of early-seizure clips is that the 15 second cutoff does not seem to correspond with a qualitative change in the data. Therefore, it stands to reason that using a different cutoff point during training may increase the efficiency of identifying early-seizure clips, albeit at the cost of an increased false positive rate. Several cutoff points were tested, and the optimal choice was found to be 19 seconds. This change in the training method increased the AUC score for the model from 0.94 to 0.96.

Several other classifiers, including gradient boost, adaboost, and embedded trees, were tested as alternatives to the extra-trees approach. Despite giving good results on our cross-validations sets, they ultimately seemed to overfit and did not achieve AUC scores beyond 0.9 in 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*, *matplotlib*, and *scikit-learn*. The code was tested a Mac and a Windows 8 pc. Approximately 90% of the run-time spent on the feature calculation of the training and test datasets. Distributing the training among two cores by setting the extra-trees parameter *n_jobs* = *2* led to a moderate speed up.

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 down sample each clip to. The data selector is a list of lists in which each sub-list specifies a patient and what fraction of that patient 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 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 extra-trees classifier is done in the **trainExtraTrees** 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." Similar functions exist for other classifiers, all with the same input and output format.

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 dataframe, 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 patients. For each patient, it loads the patient training data, trains an extra-trees classifier, loads the test data, and stores the classifier 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
- 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 patient data files in variable *dataDirectory* in **utils.py**.
2. Include the list of patients to train and predict in variable *dataSelector* in **test.py**.
3. Run **test.py**. This will train each of the specified patients 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 patient 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.

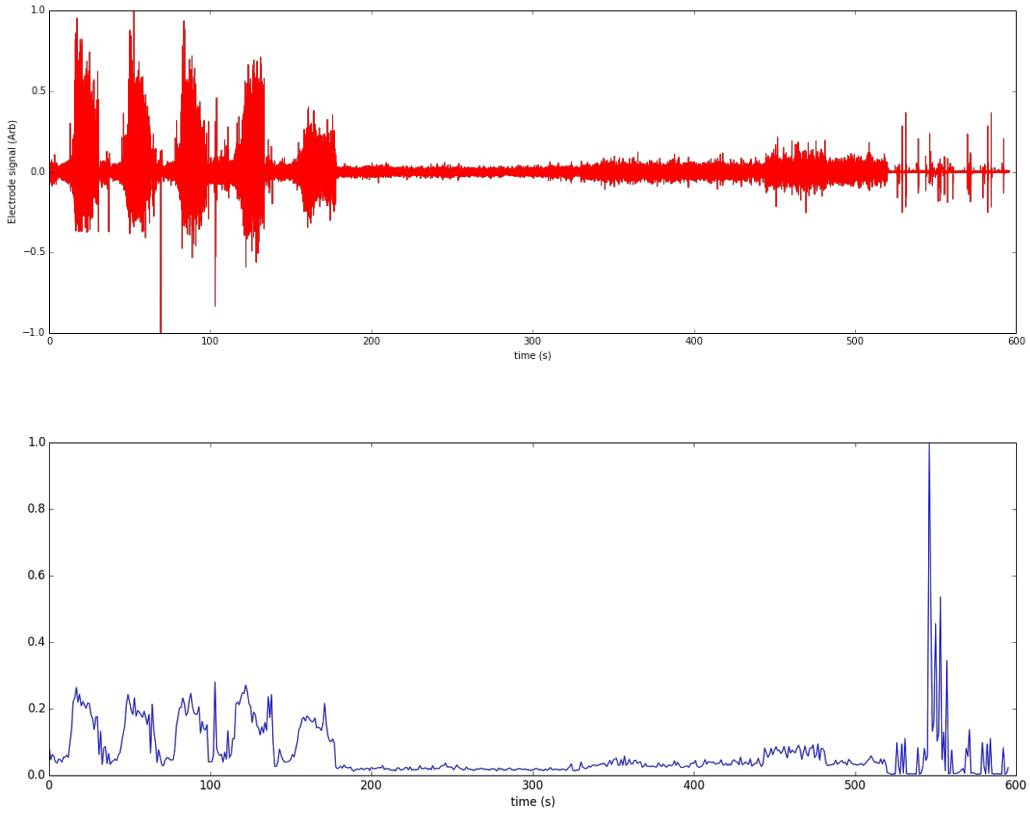

Fig. 1. (a) Signal collected by electrode 14 of Dog 1 showing ictal segments followed by interictal segments. (b) Standard deviation of the amplitudes for this electrode with the clips arranged in the same order as in (a).

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.