BE 521: Homework 4

HFOs

Spring 2015

58 points

Due: 2/24/2015

Objective: HFO detection and cross-validation

Homework Policy

1. Piazza should be used for peer discussion for all questions related to course material. Please also use Piazza to contact teaching staff for all questions. TA's will be available to help during office hours and occasionally on Piazza.

2. Submit LaTeX write-up (pdf) and Matlab code to Canvas as pennkey_hwx.pdf, .m before listed deadline.

3. Assignments will be returned electronically on Canvas.

4. Collaboration is encouraged but individual write-ups are required. Please list any collaborators. Honor code will be strictly enforced. Note: submitted code is routinely passed through a plagiarism checker.

5. Late Policy: 5% per day. **No homework is accepted after the 5th late day.** (e.g. If originally due Tuesday, 11:59PM, last day to turn in is Sunday, 11:59 PM).

HFO Dataset

High frequency oscillations (HFOs) are quasi-periodic intracranial EEG transients with durations on the order of tens of milliseconds and peak frequencies in the range of 80 to 500 Hz. There has been considerable interest among the epilepsy research community in the potential of these signals as biomarkers for epileptogenic networks.

In this homework exercise, you will explore a dataset of candidate HFOs detected using the algorithm of Staba et al. (see article on Canvas). The raw recordings from which this dataset arises come from a human subject with mesial temporal lobe epilepsy and were contributed by the laboratory of Dr. Greg Worrell at the Mayo Clinic in Rochester, MN.

The dataset `I521_A0004_D001` contains raw HFO clips that are normalized to zero mean and unit standard deviation but are otherwise unprocessed. The raw dataset contain two channels of data: `Test_raw_norm` and `Train_raw_norm`, storing raw testing and training sets of HFO clips respectively. The raw dataset also contains two annotation layers: `Testing windows` and

`Training windows`, storing HFO clip start and stop times (in microseconds) for each of the two channels above.

Annotations contain the classification by an "expert" reviewer (i.e., a doctor) of each candidate HFO as either an HFO (2) or an artifact (1).

After loading the dataset in to a `session` variable as in prior assignments You will want to familiarize yourself with the `IEEGAnnotationLayer` class in the IEEGToolbox documentation to retrieve the time window and expert classification for each candidate HFO. Keep in mind that every HFO has a unique duration.

1 Simulating the Staba Detector (12 pts)

Candidate HFO clips were detected with the Staba et al. algorithm and subsequently validated by an expert as a true HFO or not. In this first section, we will use the original iEEG clips containing HFOs and re-simulate a portion of the Staba detection.

1. How many samples exist for each class (HFO vs artifact) in the training set? (Show code to support your answer) (1 pt)

2. Using the training set, find the first occurrence of the first valid HFO and the first artifact. Using `subplot` with 2 plots, plot the valid HFO's (left) and artifact's (right) waveforms. Since the units are normalized, there's no need for a y-axis, so remove it with the command `set(gca,'YTick',[])`. (2 pts)

3. Using the `fdatool` in MATLAB, build an FIR bandpass filter of the equiripple type of order 100. Use the Staba et al. (2002) article to guide your choice of passband and stopband frequency. Once set, go to `File -> Export`, and export "Coefficients" as a MAT-File. Attach a screenshot of your filter's magnitude response. (N.B., We will be flexible with the choice of frequency parameters within reason.) (3 pts)

4. Using the forward-backward filter function (`filtfilt`) and the numerator coefficients saved above, filter the valid HFO and artifact clips obtained earlier. You will need to make a decision about the input argument `a` in the `filtfilt` function. Plot these two filtered clips overlayed on their original signal in a two plot `subplot` as before. Remember to remove the y-axis. (3 pts)

5. Speculate how processing the data using Staba's method may have erroneously led to a false HFO detection (3 pts)

2 Defining Features for HFOs (9 pts)

In this section we will be defining a feature space for the iEEG containing HFOs and artifacts. These features will describe certain attributes about the waveforms upon which a variety of classification tools will be applied to better segregate HFOs and artifacts

1. Create two new matrices, `trainFeats` and `testFeats`, such that the number of rows correspond to observations (i.e. number of training and testing clips) and the number of columns is two. Extract the line-length and area features (seen previously in lecture and Homework

3) from the normalized raw signals. Store the line-length value in the first column and area value for each sample in the second column of your features matrices. Make a scatter plot of the training data in the 2-dimensional feature space, coloring the valid detections blue and the artifacts red. (N.B., Since we only want one value for each feature of each clip, you will effectively treat the entire clip as the one and only "window".) (4 pts)

2. Feature normalization is often important. One simple normalization method is to subtract each feature by its mean and then divide by its standard deviation (creating features with zero mean and unit variance). Using the means and standard deviations calculated in your *training* set features, normalize both the training and testing sets.

   (a) What is the statistical term for the normalized value, which you have just computed? (1 pt)

   (b) Explain why such a feature normalization might be critical to the performance of a k-NN classifier. (2 pts)

   (c) Explain why (philosophically) you use the training feature means and standard deviations to normalize the testing set. (2 pts)

3 Comparing Classifiers (20 pts)

In this section, you will explore how well a few standard classifiers perform on this dataset. Note, the logistic regression and k-NN classifiers are functions built into some of Matlabs statistics packages. If you don't have these (i.e., Matlab doesn't recognize the functions), we've provided them, along with the LIBSVM mex files, in the `lib.zip` file. To use these, unzip the folder and add it to your Matlab path with the command `addpath lib`. If any of these functions don't work, please let us know.

1. Using Matlab's logistic regression classifier function, (`mnrfit`), and its default parameters, train a model on the training set. Using Matlab's `mnrval` function, calculate the training error (as a percentage) on the data. For extracting labels from the matrix of class probabilities, you may find the command `[~,Ypred] = max(X,[],2)` useful[1], which gets the column-index of the maximum value in each row (i.e., the class with the highest predicted probability). (3 pts)

2. Using the model trained on the training data, predict the labels of the test samples and calculate the testing error. Is the testing error larger or smaller than the training error? Give one sentence explaining why this might be so. (2 pts)

3. (a) Use Matlab's k-nearest neighbors function, `knnclassify`, and its default parameters (k = 1, among other things), calculate the training and testing errors. (3 pts)

   (b) Why is the training error zero? (2 pts)

4. In this question you will use the LIBSVM implementation of a support vector machine (SVM). LIBSVM is written in C, but we'll use the Matlab executable versions (with *.mex* file

[1] N.B., some earlier versions of Matlab don't like the ~, which discards an argument, so just use something like `[trash,Ypred] = max(X,[],2)` instead.

extensions). Type `svmtrain` and `svmpredict` to see how the functions are used[2]. Report the training and testing errors on an SVM model with default parameters. (3 pts)

5. It is sometimes useful to visualize the decision boundary of a classifier. To do this, we'll plot the classifier's prediction value at every point in the "decision" space. Use the `meshgrid` function to generate points in the line-length and area 2D feature space and a scatter plot (with the `'.'` point marker) to visualize the classifier decisions at each point (use yellow and cyan for your colors). In the same plot, show the training samples (plotted with the '*' marker to make them more visible) as well. As before use blue for the valid detections and red for the artifacts. Use ranges of the features that encompass all the training points and a density that yields that is sufficiently high to make the decision boundaries clear. Make such a plot for the logistic regression, k-NN, and SVM classifiers. (4 pts)

6. In a few sentences, report some observations about the three plots, especially similarities and differences between them. Which of these has overfit the data the most? Which has underfit the data the most? (3 pts)

4 Cross-Validation (17 pts)

In this section, you will investigate the importance of cross-validation, which is essential for choosing the tunable parameters of a model (as opposed to the internal parameters the the classifier "learns" by itself on the training data).

1. Since you cannot do any validation on the testing set, you'll have to split up the training set. One way of doing this is to randomly split it into k unique "folds," with roughly the same number of samples (n/k for n total training samples) in each fold, training on $k-1$ of the folds and doing predictions on the remaining one.

   In this section, you will do 10-fold cross-validation, so create a cell array[3] `folds` that contains 10 elements, each of which is itself an array of the indices of training samples in that fold. You may find the `randperm` function useful for this.

   Using the command `length(unique([folds{:}]))`, show that you have 200 unique sample indices (i.e. there are no repeats between folds). (2 pts)

2. Train a new k-NN model (still using the default parameters) on the folds you just created. Predict the labels for each fold using a classifier trained on all the other folds. After running through all the folds, you will have label predictions for each training sample.

   (a) Compute the error (called the validation error) of these predictions. (3 pts)

   (b) How does this error compare (lower, higher, the same?) to the error you found in question 2.3? Does it make sense? (2 pts)

[2]Matlab has its own analogous functions, `svmtrain` and `svmclassify`, so make sure that the LIBSVM files have been added to your path (and thus will superceed the default Matlab functions).

[3]A cell array is slightly different from a normal Matlab numeric array in that it can hold elements of variable size, for example `folds{1} = [1 3 6]; folds{2} = [2 5]; folds{3} = [4 7];`.

3. Create a parameter space for your k-NN model by setting a vector of possible k values from 1 to 30. For each values of k, calculate the validation error and average training error over the 10 folds.

   (a) Plot the training and validation error values over the values of k, using the formatting string `'b-o'` for the validation error and `'r-o'` for the training error. (4 pts)

   (b) What is the optimal k value and its error? (1 pts)

   (c) Explain why k-NN generally overfits less with higher values of k. (2 pts)

4. (a) Using your optimal value for k from CV, calculate the k-NN model's testing error. (1 pts)

   (b) How does this cross-validated model's testing error compare to the k-NN model you trained in question 3.3? Is it the best of the three models you trained in Section 3? (2 pts)