BE 521: Homework 4

HFOs

Spring 2019

58 points

Due: Tuesday, 2/19/2019 11:59pm

Objective: HFO detection and cross-validation

John Talley
Collaborators: Joseph Iwasyk, John Bellwoar, Andrew Clark

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). On ieeg.org and upon downloading the annotations, You can view this in the "description" field.

After loading the dataset in to a `session` variable as in prior assignments you will want to familiarize yourself with the `IEEGAnnotationLayer` class. Use the provided "getAnnotations.m" function to get all the annotations from a given dataset. The first output will be an array of annotation objects, which you will see also has multiple fields including a description field as well as start and stop times. Use You can use the information outputted by getAnnotations to pull each HFO clip.

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)

```
dataset_ID = 'I521_A0004_D001';
ieeg_id = 'jtalley';
ieeg_pw = 'jta_ieeglogin.bin';

session = IEEGSession(dataset_ID,ieeg_id,ieeg_pw);

durInUSec1 = session.data.rawChannels(1).get_tsdetails.getDuration;
durInSec1 = durInUSec1/1e6;
durInUSec2 = session.data.rawChannels(2).get_tsdetails.getDuration;
durInSec2 = durInUSec2/1e6;

Fs  = session.data.sampleRate;

Test_raw_norm = getvalues(session.data,1:durInSec1*Fs,1);
Train_raw_norm = getvalues(session.data,1:durInSec2*Fs,2);

% train_annotations = getAnnotations(session.data, session.data.annLayer(1).name);
% test_annotations = getAnnotations(session.data, session.data.annLayer(2).name);

[allTrainEvents, train_timesUSec, train_channels] = getAnnotations(session.data, session.data.annLayer(1).na
[allTestEvents, test_timesUSec, test_channels] = getAnnotations(session.data, session.data.annLayer(2).name)

train_artifact_samples = 0;
train_artifact_idx = [];
train_hfo_samples = 0;
train_hfo_idx = [];


for i = 1:length(allTrainEvents)
    if allTrainEvents(i).description == '1'
        train_artifact_samples = train_artifact_samples + 1; % 101 samples
        train_artifact_idx(length(train_artifact_idx) + 1) = i;
    else
        train_hfo_samples = train_hfo_samples + 1; % 99 samples
        train_hfo_idx(length(train_hfo_idx) + 1) = i;
    end
end


test_artifact_samples = 0;
test_artifact_idx = [];
test_hfo_samples = 0;
test_hfo_idx = [];

for j = 1:length(allTestEvents)
    if allTestEvents(j).description == '1'
        test_artifact_samples = test_artifact_samples + 1;
        test_artifact_idx(length(test_artifact_idx) + 1) = j;
    else
        test_hfo_samples = test_hfo_samples + 1;
        test_hfo_idx(length(test_hfo_idx) + 1) = j;
    end
end

% There are 101 HFO samples and 99 artifacts in the training set.
```

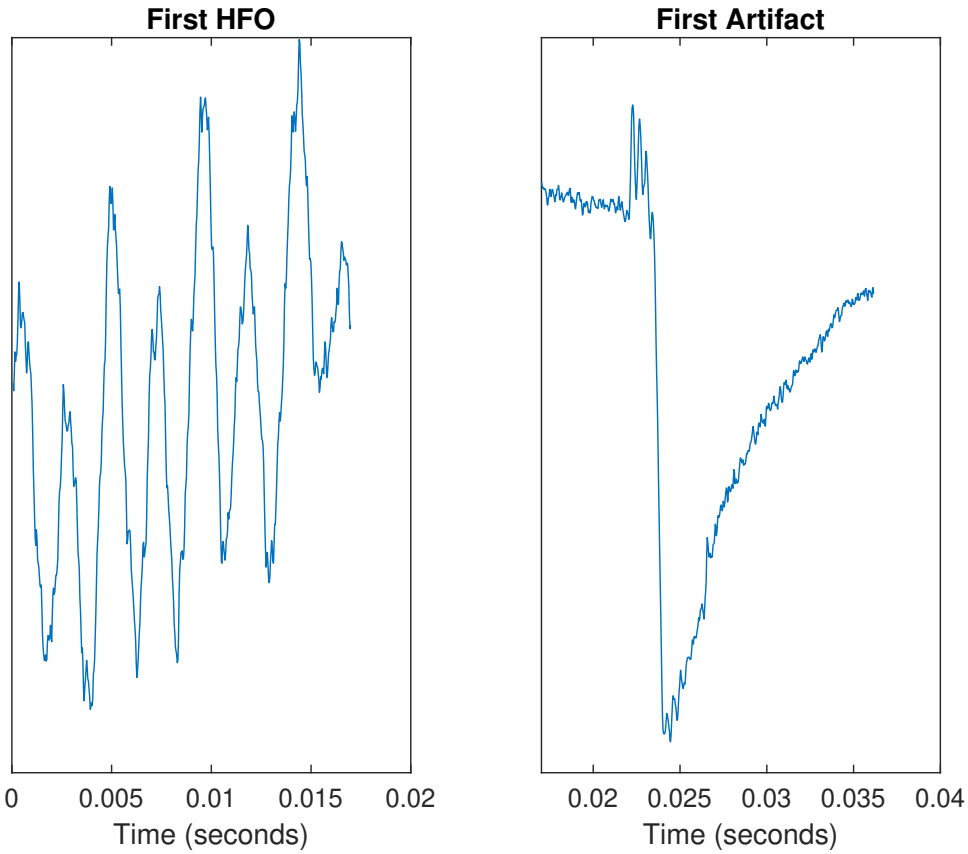
There are 101 HFO samples and 99 artifacts in the training set.

```
Warning: Objects of edu/upenn/cis/db/mefview/services/TimeSeriesDetails class
```

```
exist — not clearing java
Warning: Objects of edu/upenn/cis/db/mefview/services/TimeSeriesInterface class
exist — not clearing java
IEEGSETUP: Found log4j on Java classpath.
URL: https://www.ieeg.org/services
Client user: jtalley
Client password: ****
```

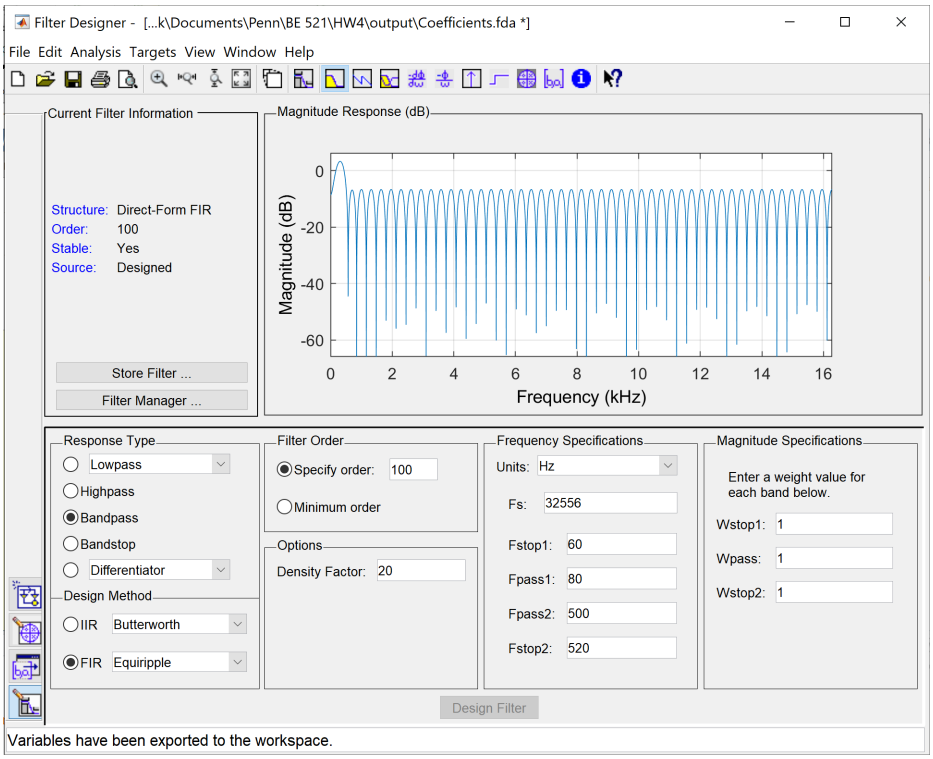
2. Using the training set, find the first occurrence of the first valid HFO and the first artifact. Using **subplot** with 2 plots, plot the valid HFO's (left) and artifact's (right) waveforms. Since the units are normalized, there's no need for a y-axis, so remove it with the command **set(gca,'YTick',[])**.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

```
train_artifact_start_us  = train_timesUSec(train_artifact_idx,1); % microseconds
train_artifact_start = ceil(train_artifact_start_us/1e6*Fs); % samples
train_artifact_stop_us  = train_timesUSec(train_artifact_idx,2);
train_artifact_stop = ceil(train_artifact_stop_us/1e6*Fs);
first_artifact = Train_raw_norm(train_artifact_start(1):train_artifact_stop(1));
t_art1 = train_artifact_start(1)/Fs:1/Fs:train_artifact_stop(1)/Fs;

train_hfo_start_us  = train_timesUSec(train_hfo_idx,1); % microseconds
train_hfo_start = ceil(train_hfo_start_us/1e6*Fs) + 1; % samples
train_hfo_stop_us  = train_timesUSec(train_hfo_idx,2);
train_hfo_stop = ceil(train_hfo_stop_us/1e6*Fs);
first_hfo = Train_raw_norm(train_hfo_start(1):train_hfo_stop(1));
t_hfo1 = train_hfo_start(1)/Fs:1/Fs:train_hfo_stop(1)/Fs;

figure;
subplot(1,2,1);
plot(t_hfo1, first_hfo);
xlabel('Time (seconds)');
title('First HFO');
set(gca,'YTick',[]);
hold on;
subplot(1,2,2);
plot(t_art1, first_artifact);
xlabel('Time (seconds)');
title('First Artifact');
set(gca,'YTick',[]);
```



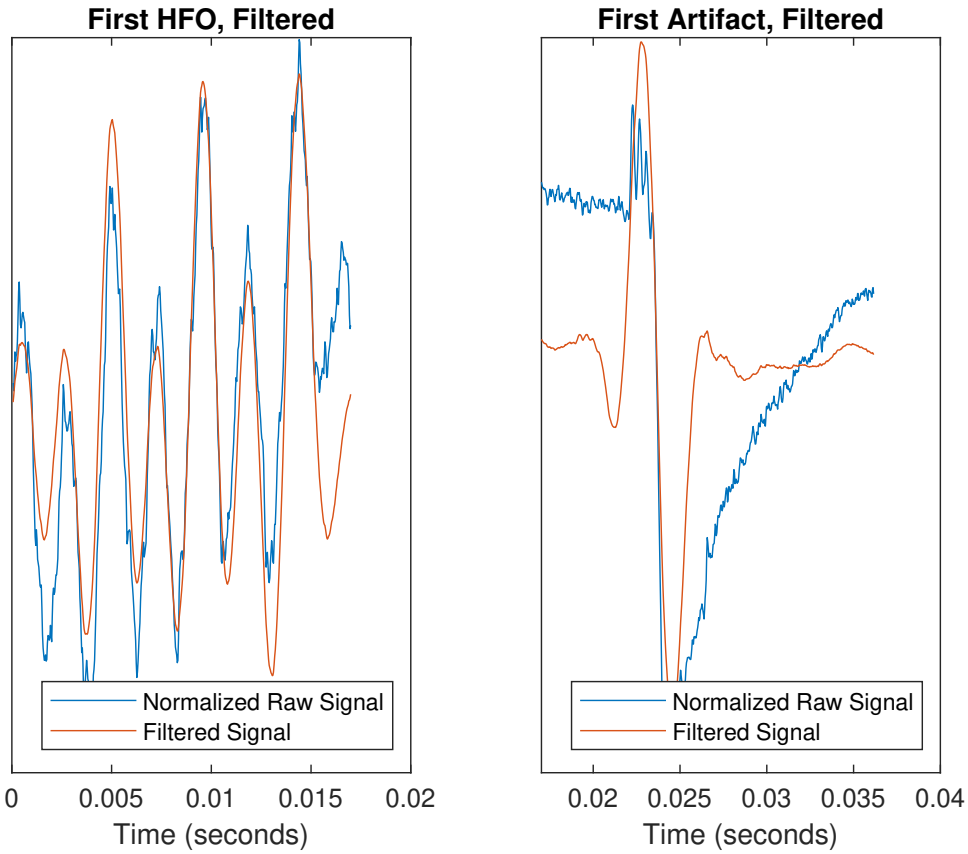
3. Using the `fdatool` in MATLAB, build an FIR bandpass filter of the equiripple type of order 100. Use the Staba et al. (2002) article to guide your choice of passband and stopband frequency. Once set, go to `File -> Export`, and export "Coefficients" as a MAT-File. Attach a screenshot of your filter's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

```
load('Coefficients.mat'); % from fdatool command

% Bandpass filter with range 80–500 Hz
```



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)

```
filtered_hfo1 = filtfilt(Num, 1, first_hfo);
filtered_artifact1 = filtfilt(Num, 1, first_artifact);

figure;
subplot(1,2,1);
plot(t_hfo1, first_hfo);
hold on;
plot(t_hfo1, filtered_hfo1);
xlabel('Time (seconds)');
title('First HFO, Filtered');
legend('Normalized Raw Signal', 'Filtered Signal','Location','southeast');
set(gca,'YTick',[]);
hold on;
subplot(1,2,2);
plot(t_art1, first_artifact);
hold on;
plot(t_art1, filtered_artifact1);
xlabel('Time (seconds)');
title('First Artifact, Filtered');
legend('Normalized Raw Signal', 'Filtered Signal','Location','southeast');
set(gca,'YTick',[]);
```



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

   Using Staba's method and a bandpass filter of 80-500 Hz, we may encounter a false HFO detection due to the removal of the low and high freqeuncy components from the clip. As pictured in the plot above, removing these components make the artifact look similar to the true HFO on the left, which could lead to a false positive HFO detection based on this waveform.

2 Defining Features for HFOs (9 pts)

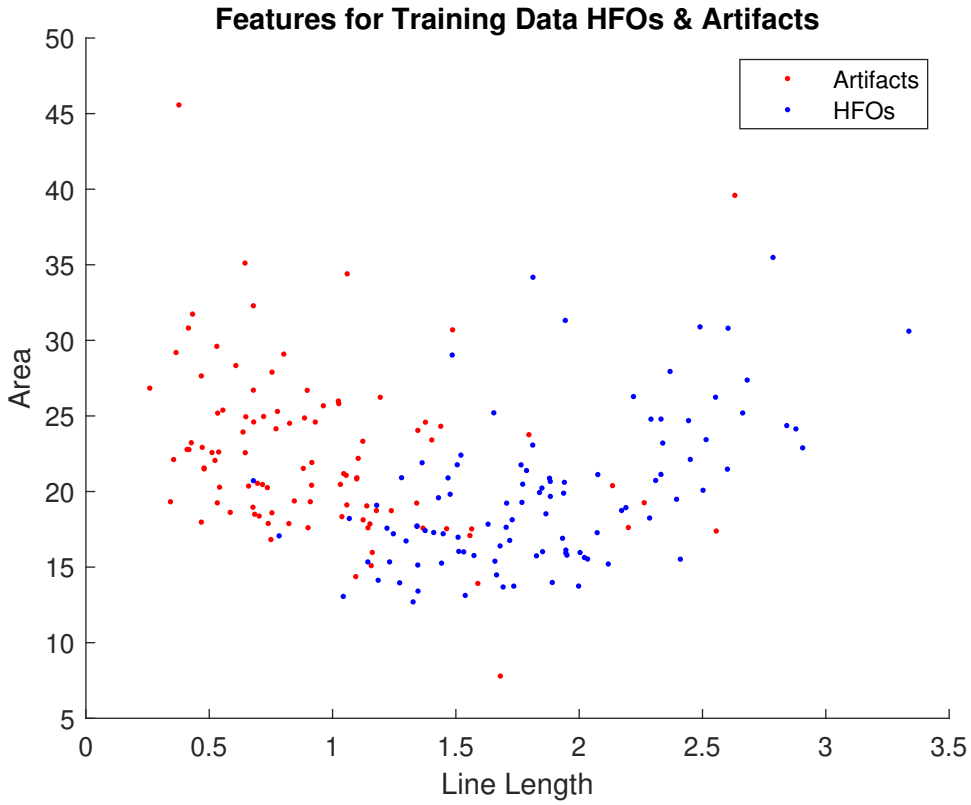
In this section we will be defining a feature space for the iEEG containing HFOs and artifacts. These features will describe certain attributes about the waveforms upon which a variety of classification tools will be applied to better segregate HFOs and artifacts

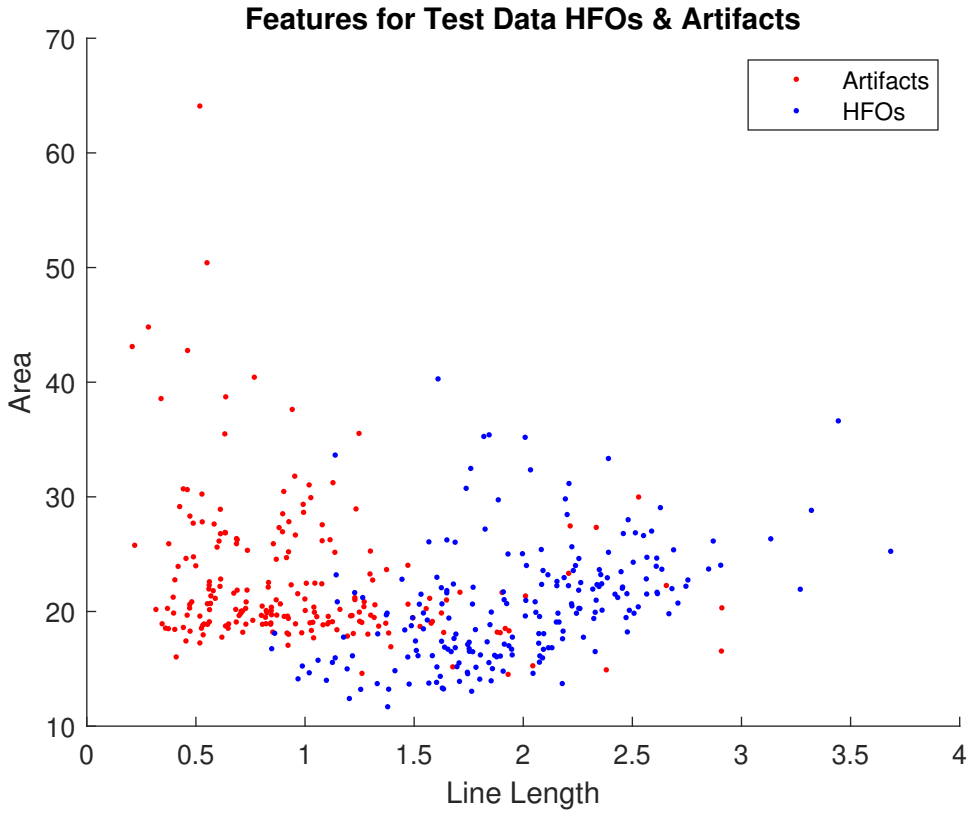
1. Create two new matrices, `trainFeats` and `testFeats`, such that the number of rows correspond to observations (i.e. number of training and testing clips) and the number of columns is two. Extract the line-length and area features (seen previously in lecture and Homework 3) from the normalized raw signals (note: use the raw signal from ieeg.org, do not filter the signal). 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. (Note: 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)

```
trainFeats = [];
```

```
% train_artifact_startTimes = [];
% train_artifact_stopTimes = [];
% train_hfo_startTimes = [];
% train_hfo_stopTimes = [];

testFeats = [];
% test_artifact_startTimes = [];
% test_artifact_stopTimes = [];
% test_hfo_startTimes = [];
% test_hfo_stopTimes = [];

% for i = 1:length(train_annotations)
%     if ismember(i,train_artifact_idx)
%         train_artifact_startTimes(length(train_artifact_startTimes) + 1) = train_annotations(i).start;
%         train_artifact_stopTimes(length(train_artifact_stopTimes) + 1) = train_annotations(i).stop;
%     else
%         train_hfo_startTimes(length(train_hfo_startTimes) + 1) = train_annotations(i).start;
%         train_hfo_stopTimes(length(train_hfo_stopTimes) + 1) = train_annotations(i).stop;
%     end
% end

% train_artifact_startTimes = ceil(train_artifact_startTimes./1e6*Fs);
% train_artifact_stopTimes = ceil(train_artifact_stopTimes./1e6*Fs);
% train_hfo_startTimes = ceil(train_hfo_startTimes./1e6*Fs);
% train_hfo_startTimes(1) = 1;
% train_hfo_stopTimes = ceil(train_hfo_stopTimes./1e6*Fs);
%
% for j = 1:length(test_annotations)
%     if ismember(j,test_artifact_idx)
%         test_artifact_startTimes(length(test_artifact_startTimes) + 1) = test_annotations(j).start;
%         test_artifact_stopTimes(length(test_artifact_stopTimes) + 1) = test_annotations(j).stop;
%     else
%         test_hfo_startTimes(length(test_hfo_startTimes) + 1) = test_annotations(j).start;
%         test_hfo_stopTimes(length(test_hfo_stopTimes) + 1) = test_annotations(j).stop;
%     end
% end

% test_artifact_startTimes = ceil(test_artifact_startTimes./1e6*Fs);
% test_artifact_stopTimes = ceil(test_artifact_stopTimes./1e6*Fs);
% test_hfo_startTimes = ceil(test_hfo_startTimes./1e6*Fs);
% test_hfo_startTimes(1) = 1;
% test_hfo_stopTimes = ceil(test_hfo_stopTimes./1e6*Fs);

% Test session start and stop times
test_artifact_start_us  = test_timesUSec(test_artifact_idx,1); % microseconds
test_artifact_start = ceil(test_artifact_start_us/1e6*Fs); % samples
test_artifact_stop_us  = test_timesUSec(test_artifact_idx,2);
test_artifact_stop = ceil(test_artifact_stop_us/1e6*Fs);

test_hfo_start_us  = test_timesUSec(test_hfo_idx,1); % microseconds
test_hfo_start = ceil(test_hfo_start_us/1e6*Fs) + 1; % samples
test_hfo_stop_us  = test_timesUSec(test_hfo_idx,2);
test_hfo_stop = ceil(test_hfo_stop_us/1e6*Fs);

LLFn = @(x) sum(abs(diff(x)));
Area = @(x) sum(abs(x));

train_artLengths = [];
train_artAreas = [];

train_hfoLengths = [];
train_hfoAreas = [];

for n = 1:length(train_artifact_start)-1
    train_artLengths(n) = LLFn(Train_raw_norm(train_artifact_start(n):train_artifact_stop(n)));
```

```
    train_artAreas(n) = Area(Train_raw_norm(train_artifact_start(n):train_artifact_stop(n)));
end
train_artLengths(99) = LLFn(Train_raw_norm(train_artifact_start(99):train_artifact_stop(99)-2));
train_artAreas(99) = Area(Train_raw_norm(train_artifact_start(99):train_artifact_stop(99)-2));

for m = 1:length(train_hfo_start)
    train_hfoLengths(m) = LLFn(Train_raw_norm(train_hfo_start(m):train_hfo_stop(m)));
    train_hfoAreas(m) = Area(Train_raw_norm(train_hfo_start(m):train_hfo_stop(m)));
end

test_artLengths = [];
test_artAreas = [];

test_hfoLengths = [];
test_hfoAreas = [];

for n = 1:length(test_artifact_start)
    test_artLengths(n) = LLFn(Test_raw_norm(test_artifact_start(n):test_artifact_stop(n)));
    test_artAreas(n) = Area(Test_raw_norm(test_artifact_start(n):test_artifact_stop(n)));
end

for m = 1:length(test_hfo_start)-1
    test_hfoLengths(m) = LLFn(Test_raw_norm(test_hfo_start(m):test_hfo_stop(m)));
    test_hfoAreas(m) = Area(Test_raw_norm(test_hfo_start(m):test_hfo_stop(m)));
end
test_hfoLengths(209) = LLFn(Test_raw_norm(test_hfo_start(209):test_hfo_stop(209)-2));
test_hfoAreas(209) = Area(Test_raw_norm(test_hfo_start(209):test_hfo_stop(209)-2));

trainFeats = zeros(length(allTrainEvents),2);
trainFeats(train_artifact_idx,1) = train_artLengths;
trainFeats(train_artifact_idx,2) = train_artAreas;
trainFeats(train_hfo_idx,1) = train_hfoLengths;
trainFeats(train_hfo_idx,2) = train_hfoAreas;
figure;
scatter(trainFeats(train_artifact_idx,1),trainFeats(train_artifact_idx,2),'.r');
hold on;
scatter(trainFeats(train_hfo_idx,1),trainFeats(train_hfo_idx,2),'.b');
xlabel('Line Length');
ylabel('Area');
title('Features for Training Data HFOs & Artifacts');
legend('Artifacts','HFOs');

testFeats = zeros(length(allTestEvents),2);
testFeats(test_artifact_idx,1) = test_artLengths;
testFeats(test_artifact_idx,2) = test_artAreas;
testFeats(test_hfo_idx,1) = test_hfoLengths;
testFeats(test_hfo_idx,2) = test_hfoAreas;
figure;
scatter(testFeats(test_artifact_idx,1),testFeats(test_artifact_idx,2),'.r');
hold on;
scatter(testFeats(test_hfo_idx,1),testFeats(test_hfo_idx,2),'.b');
xlabel('Line Length');
ylabel('Area');
title('Features for Test Data HFOs & Artifacts');
legend('Artifacts','HFOs');
```


Features for Training Data HFOs & Artifacts
Artifacts
HFOs
Area
Line Length
50
45
40
35
30
25
20
15
10
5
0
0.5
1
1.5
2
2.5
3
3.5


Features for Test Data HFOs & Artifacts
Artifacts
HFOs
Area
Line Length
70
60
50
40
30
20
10
0
0.5
1
1.5
2
2.5
3
3.5
4

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. You should use these normalized features for the remainder of the assignment.

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

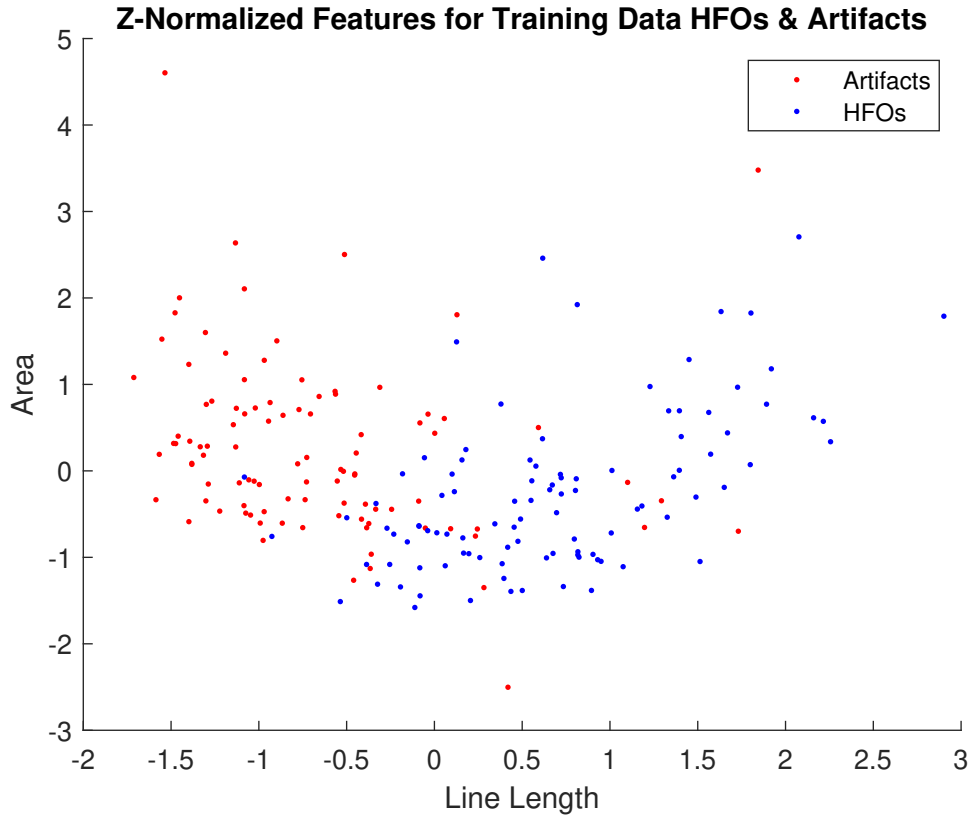
```
trainFeatsZ(:,1) = (trainFeats(:,1) - mean(trainFeats(:,1)))./std(trainFeats(:,1)); % Z-scored lengths
trainFeatsZ(:,2) = (trainFeats(:,2) - mean(trainFeats(:,2)))./std(trainFeats(:,2)); % Z-scored areas

% Now, normalize testing set using mean and stdev from training data

testFeatsZ(:,1) = (testFeats(:,1) - mean(trainFeats(:,1)))./std(trainFeats(:,1)); % Z-scored lengths
testFeatsZ(:,2) = (testFeats(:,2) - mean(trainFeats(:,2)))./std(trainFeats(:,2)); % Z-scored areas

% Z-score.
```

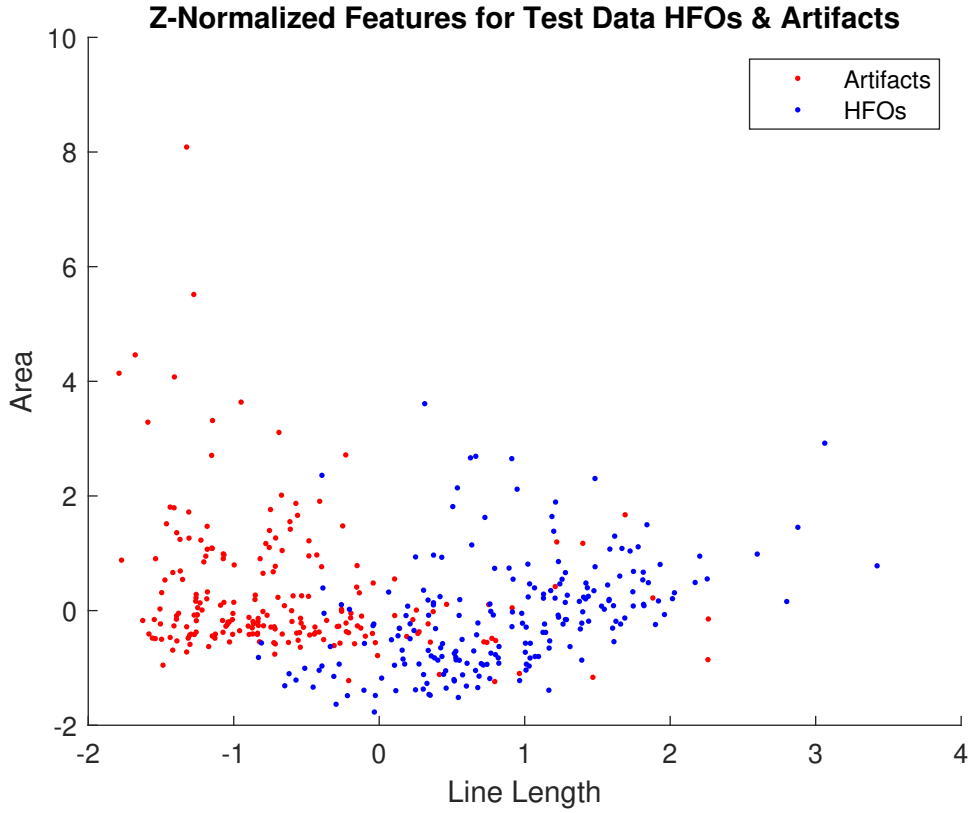
   The statistical term for the normalized value is Z-score.

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

   Plotting normalized, z-scored data from above to visually support the answer to this question

```
figure;
scatter(trainFeatsZ(train_artifact_idx,1),trainFeatsZ(train_artifact_idx,2),'.r');
hold on;
scatter(trainFeatsZ(train_hfo_idx,1),trainFeatsZ(train_hfo_idx,2),'.b');
xlabel('Line Length');
ylabel('Area');
title('Z-Normalized Features for Training Data HFOs & Artifacts');
legend('Artifacts','HFOs');

figure;
scatter(testFeatsZ(test_artifact_idx,1),testFeatsZ(test_artifact_idx,2),'.r');
hold on;
scatter(testFeatsZ(test_hfo_idx,1),testFeatsZ(test_hfo_idx,2),'.b');
xlabel('Line Length');
ylabel('Area');
title('Z-Normalized Features for Test Data HFOs & Artifacts');
legend('Artifacts','HFOs');
```


Z-Normalized Features for Training Data HFOs & Artifacts
Artifacts
HFOs
Area
Line Length
5
4
3
2
1
0
-1
-2
-3
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


Z-Normalized Features for Test Data HFOs & Artifacts
Artifacts
HFOs
Area
Line Length
10
8
6
4
2
0
-2
-2
-1
0
1
2
3
4

Z-score normalization (or a similar method) is critical to k-NN classification in order to ensure that higher scale features do not dominate in the Euclidean distance calculation that this classification uses. When the data are normalized to be centered at zero and have unit variance, the features have a mean centered at 0 and a unit variance.

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

The training data provides a set of known HFOs and artifacts that can be used to build a model that will be applied to the testing set of unknown classifications. From these known values, we can calculate an expected mean and standard deviation. After deriving these values from the training data set, the mean and standard deviation can be generalized to z-score normalize the testing dataset. Models such as k-NN, logistic regression, and support vector machine (SVM) can then easily use these normalized values to classify features from the testing set after being trained on the training set.

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)

```
% logistic regression classifier function
train_class = zeros(200,1);
train_class(train_artifact_idx) = 1;
train_class(train_hfo_idx) = 2;
B = mnrfit(trainFeatsZ,train_class); % coefficients
% training error (percentage);
probs = mnrval(B,trainFeatsZ);

[~,Ytrain] = max(probs,[],2);
training_error = 100*((length(train_class) - sum(Ytrain == train_class))/length(train_class));

% 12.50% training error, MATLAB logistic regression
```

12.50 percent training error, MATLAB logistic regression

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)

```
test_class = zeros(420,1);
test_class(test_artifact_idx) = 1;
test_class(test_hfo_idx) = 2;
```

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

```
% testing error (percentage);
probabilities = mnrval(B,testFeatsZ);

[~,Ytest] = max(probabilities,[],2);
test_error = 100*((length(test_class) - sum(Ytest == test_class))/length(test_class));

% 13.5714% testing error, MATLAB logistic regression
```

13.5714 percent testing error, MATLAB logistic regression

The testing error is slightly larger than the training error. This makes sense, because the trained model should have a lower error rate on the data used to derive the coefficients of the model than the error rate it will exhibit in predictions on a foreign dataset that it has not trained on, such as this testing set.

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

```
knn_mdl = fitcknn(trainFeatsZ,train_class);
knn_train = predict(knn_mdl,trainFeatsZ);
knn_train_error = 100*((length(train_class) - sum(knn_train == train_class))/length(train_class));
% 0% training error from knn method

knn_test = predict(knn_mdl,testFeatsZ);
knn_test_error = 100*((length(test_class) - sum(knn_test == test_class))/length(test_class));
% 17.3810% testing error from knn method
```

0 percent training error from k-NN method, 17.3810 percent testing error from k-NN method.

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

Because a k-NN classifier with a default k-value of 1 compares a feature to its nearest feature in terms of distance to categorize, it makes sense that the training error of a k-NN classifier will be 0. The model was trained with the true training classification data and the normalized training features data, so when the k-NN classifier is applied to the training set used to build the model, it looks at the nearest neighbor which is already of known classification. When this model is applied to the whole training dataset that was already used for the initial training, it will correctly classify each feature and the error rate will be 0.

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 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)

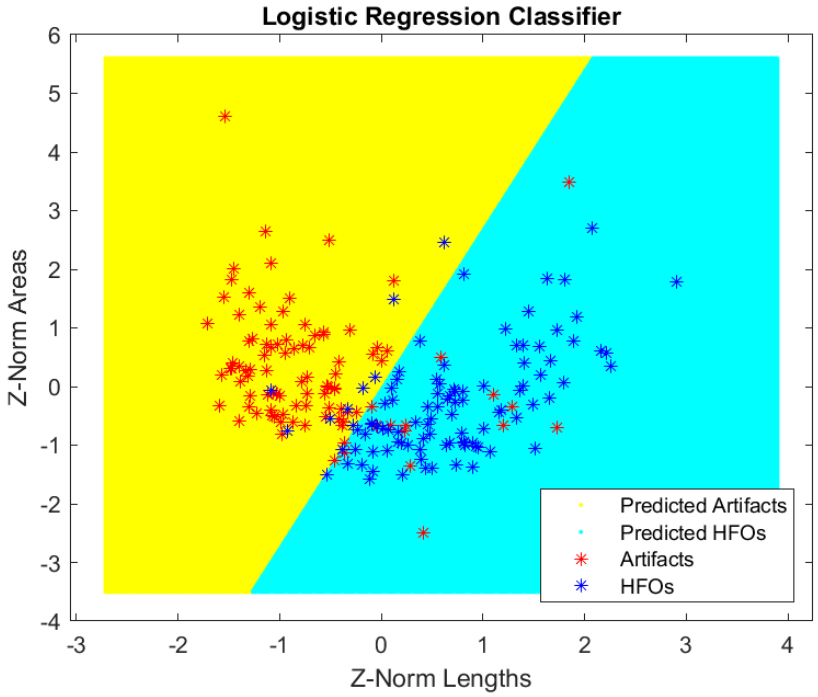
```
SVM_model = fitcsvm(trainFeatsZ,train_class,'KernelFunction','RBF');
SVM_train = predict(SVM_model, trainFeatsZ);
SVM_train_error = 100*((length(train_class) - sum(SVM_train == train_class))/length(train_class));
% 10.00% SVM training error

SVM_test = predict(SVM_model, testFeatsZ);
SVM_test_error = 100*((length(test_class) - sum(SVM_test == test_class))/length(test_class));
% 11.67% SVM testing error
```

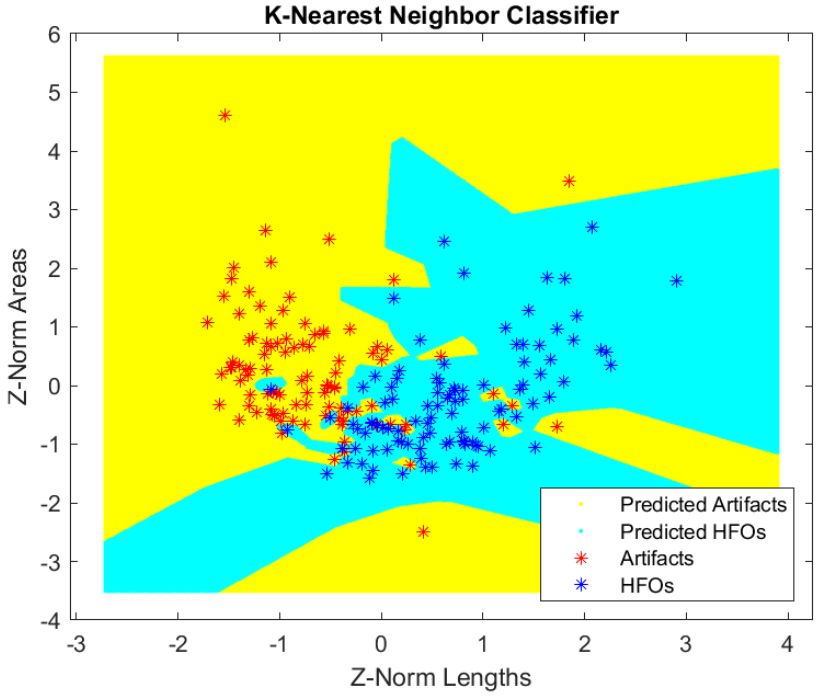
10.00 percent SVM training error, 11.67 percent SVM testing error.

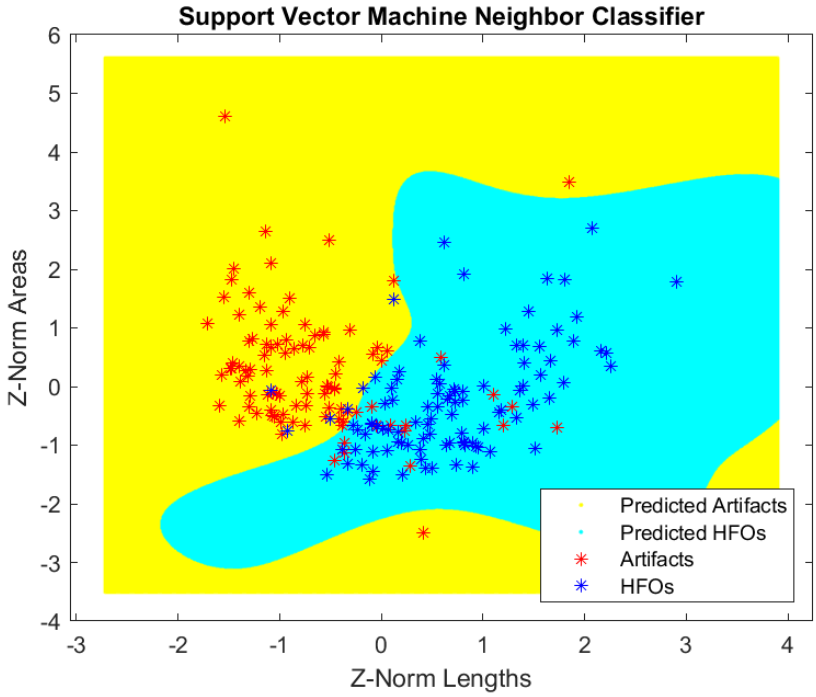
[2]Matlab has its own analogous functions, **svmtrain** and **svmclassify**, so make sure that the LIBSVM files have been added to your path (and thus will superceed the default Matlab functions).

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)

```
% LOGISTIC REGRESSION CLASSIFIER
xspace = min(trainFeatsZ(:,1))-1:0.01:max(trainFeatsZ(:,1))+1;
yspace = min(trainFeatsZ(:,2))-1:0.01:max(trainFeatsZ(:,2))+1;

[Xspace,Yspace] = meshgrid(xspace,yspace);
grid = [Xspace(:),Yspace(:)];

% X = trainFeatsZ(:,1);
% Y = trainFeatsZ(:,2);

B = mnrfit(trainFeatsZ,train_class); % coefficients
% training error (percentage);
pihat = mnrval(B,grid); % good
[~,Ypreds] = max(pihat,[],2);

figure;
gscatter(Xspace(:),Yspace(:),Ypreds,'yc','.');
hold on;
gscatter(trainFeatsZ(:,1),trainFeatsZ(:,2),train_class,'rb','*');
legend('Predicted Artifacts','Predicted HFOs','Artifacts','HFOs','Location','southeast');
xlabel('Z-Norm Lengths');
ylabel('Z-Norm Areas');
title('Logistic Regression Classifier');

% K-NN CLASSIFIER
knn_mdl = fitcknn(trainFeatsZ,train_class);
knn_train = predict(knn_mdl,trainFeatsZ);
knn_grid = predict(knn_mdl,grid);

figure;
gscatter(Xspace(:),Yspace(:),knn_grid,'yc','.');
hold on;
gscatter(trainFeatsZ(:,1),trainFeatsZ(:,2),train_class,'rb','*');
legend('Predicted Artifacts','Predicted HFOs','Artifacts','HFOs','Location','southeast');
xlabel('Z-Norm Lengths');
ylabel('Z-Norm Areas');
title('K-Nearest Neighbor Classifier');

% SUPPORT VECTOR MACHINE CLASSIFIER

SVM_model = fitcsvm(trainFeatsZ,train_class,'KernelFunction','RBF');
SVM_train = predict(SVM_model, trainFeatsZ);
SVM_grid = predict(SVM_model, grid);

figure;
gscatter(Xspace(:),Yspace(:),SVM_grid,'yc','.');
hold on;
gscatter(trainFeatsZ(:,1),trainFeatsZ(:,2),train_class,'rb','*');
legend('Predicted Artifacts','Predicted HFOs','Artifacts','HFOs','Location','southeast');
xlabel('Z-Norm Lengths');
ylabel('Z-Norm Areas');
title('Support Vector Machine Neighbor Classifier');
```


Logistic Regression Classifier
Z-Norm Areas
Z-Norm Lengths
Predicted Artifacts
Predicted HFOs
Artifacts
HFOs


K-Nearest Neighbor Classifier
Z-Norm Areas
Z-Norm Lengths
Predicted Artifacts
Predicted HFOs
Artifacts
HFOs



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)

   MATLAB's logistic regression model provides a linear classifier that "underfits" the training data the most. This is exhibited visually by some HFOs found in the predicted artifacts region, and some artifacts found in the predicted HFO region, and the training error of 13.57 percent. The k-NN classifier (with a k value of 1) has 100 percent accuracy (or 0 percent error) in the training dataset. That is, it accurately classifies all HFOs and artifacts but "overfits" the data, leading to a very irregular and non-generalizable classifier region. This would lead to a high error rate when applied to testing sets. However, this overfitting effect can be reduced by increasing the value of k. The SVM model appears to be the most balanced classifier, with a low training error of 10 percent, and the classifier region is a middle-ground between the logistic regression and k-NN methods based on visual inspection.

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)

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

split into k unique "folds" n/k for n total samples train on k-1 of the folds and predict on the remaining one

```
k = 10;
lenFold = 200/k;
% folds{:} = reshape(randperm(200,200),k,200/k);

idxs = randperm(200,200);
for i = 1:10
    folds{i} = idxs(1+(i-1)*lenFold:i*lenFold);
end

length(unique([folds{:}]))
```

```
ans =

   200
```

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)

```
val_error = [];

for i = 1:10
    folds_train = folds;
    folds_train{i} = []; % make current fold empty, use other 9 to train
    knn_mdl = fitcknn(trainFeatsZ([folds_train{1:10}],:),train_class([folds_train{1:10}]));
    knn_predictions = predict(knn_mdl,trainFeatsZ(folds{i},:));
    classes  = train_class(folds{i});
    val_error(i) = 100*(20- sum(knn_predictions == classes))/20; % 100*(20 predictions - boolean correc
end
% Percent error
disp(val_error); % Validation errors for each fold, trained on the other 9 folds
avg_val_error = mean(val_error); % Average validation error across 10 folds
disp(avg_val_error);

% compare to known train classifications to compute error
```

```
    20
```

   Validation Errors: 25 20 15 10 25 25 30 5 20 25 Average Validation Error: 20

   NOTE: Because the folds are randomized and recalculated every time, the average validation error may vary when published than from when I ran it and answered the questions.

   (b) How does this error compare (lower, higher, the same?) to the error you found in question 3.3? Does it make sense? (2 pts)

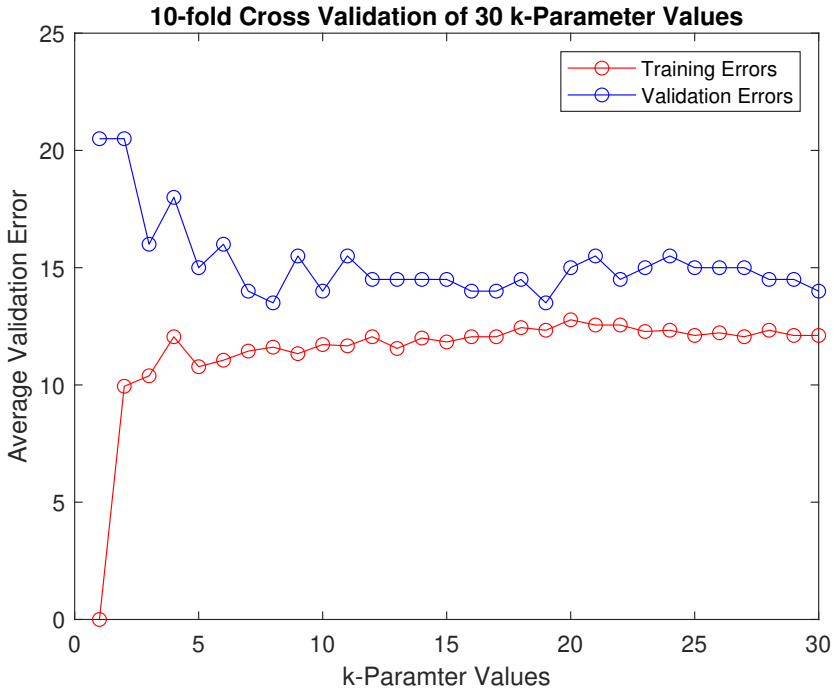
   Although the average validation error for the k-NN classifier model run on the folds fluctuates with regeneration of the 10 folds in a range of ˜18-22, it remains higher than both the k-NN training (0%) and k-NN test error (17.38%) values calculated in portion 3.3. This makes sense, because in each iteration of the loop where k-NN is applied to the folds, it predicts the classifications of the kth fold based on a model trained on k-1 folds, or only 90% of the data. In 3.3, the k-NN model

is trained once on the entire set of training features, allowing it to have a 0% error when applied to the same training set and to maintain a lower prediction error rate when applied to the testing dataset.

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)

```
valErrors = [];
trainErrors = [];

for k = 1:30 % k-parameter space
    for i = 1:10 % folds
        folds_train = folds;
        folds_train{i} = []; % make current fold empty, use other 9 to train

        knn_mdl = fitcknn(trainFeatsZ([folds_train{1:10}],:),train_class([folds_train{1:10}],:),'NumNei

        knn_predictions_train = predict(knn_mdl,trainFeatsZ([folds_train{1:10}],:));
        classes  = train_class([folds_train{1:10}],:);
        trainErrors(i) = 100*(180 - sum(knn_predictions_train == classes))/180;

        knn_predictions_val = predict(knn_mdl,testFeatsZ(folds{i},:));
        classes  = test_class(folds{i});
        valErrors(i) = 100*(20 - sum(knn_predictions_val == classes))/20;
    end
    avgTrainErrors(k) = mean(trainErrors);
    avgValErrors(k) = mean(valErrors);
end

figure;
plot(avgTrainErrors,'r-o');
hold on;
plot(avgValErrors,'b-o');
xlabel('k-Paramter Values');
ylabel('Average Validation Error');
title('10-fold Cross Validation of 30 k-Parameter Values');
legend('Training Errors', 'Validation Errors');
```



(b) What is the optimal k value and its error? (1 pts)
To determine the optimal k-value, we will take the minimum of the array of the average validation errors

```
[k_optimal_error, k_optimal] = min(avgValErrors) % Error reported in %

% k_optimal is also conventionally computed as the square root of n

k_optimal_sqrt = round(sqrt(200));

% Based on this calculation, we get a k_optimal of 14, which is close to
% our empirically derived value of k_optimal = 13 from the training data
```

```
k_optimal_error =

   13.5000


k_optimal =

     8
```

NOTE: Because the folds are randomized and recalculated every time, the optimal k value may vary when published than from when I ran it and answered the questions.

(c) Explain why k-NN generally overfits less with higher values of k. (2 pts)
k-NN generally overfits less with higher values of k because a k-NN model with a higher k compares the feature of interest to more nearest neighbors before making a classification decision prediction. In doing so, the decision classification region becomes more generalizable because the model factors in the classification of more points in the region near the feature of interest, allowing more accurate predictions without the effect of overfitting.

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

```
knn_mdl = fitcknn(trainFeatsZ,train_class,'NumNeighbors',k_optimal);
knn_test = predict(knn_mdl,testFeatsZ);
knn_optimal_test_error = 100*((length(test_class) - sum(knn_test == test_class))/length(test_class));

% k-NN testing error of 11.1905% with k_optimal = 13
% these values vary when you re-generate the folds
```

(b) How does this 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)

Using a k optimal value of 13 , the testing error of 11.1905 percent beats all three prior classifiers, including the default k-NN method. This method proves to have the lowest testing error after optimizing the k-parameter, but the SVM classifier with teh RBF kernel comes close with a testing error of 11.67 percent.