BE 521: Homework 4

HFOs

Spring 2019

58 points

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

Objective: HFO detection and cross-validation

Andrew Clark
Collaborators: Vishal Tien, Joe Iwaysk

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)

   Start the session

```
session = IEEGSession('I521_A0004_D001', 'andrewc', '/Users/andrewclark/Downloads/ieeg_password.bin' );
```

```
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: andrewc
Client password: ****
```

```
%read in testing data
nr = ceil((session.data.rawChannels(1).get_tsdetails.getEndTime)/1e6*session.data.sampleRate);
test_data = session.data.getvalues(1:nr,1);
session.data;
```

```
%read in training data
nr = ceil((session.data.rawChannels(2).get_tsdetails.getEndTime)/1e6*session.data.sampleRate);
train_data = session.data.getvalues(1:nr,2);
session.data;
```

```
%Use get Annotations function

[allEvents, timesUSec, channels]=getAnnotations(session.data,'Training windows');
```

```
%Pull all the descriptions from the output of getAnnotations
descriptions=zeros(1,length(allEvents));
for i=1:length(allEvents)
    d=allEvents(i).description;
    des=str2num(d);
    descriptions(i)=des;
    %descriptions(i)=allEvents(i).description
    %descriptions(i)=desc
end


%count the number of hfos and the number of artifacts
num_Hfos=0;
num_artifacts=0;

for j=1:length(descriptions)
    if descriptions(j)==1
        num_artifacts=num_artifacts+1;
    else
        num_Hfos=num_Hfos+1;
    end
end

num_Hfos
```

```
num_artifacts
```

```
num_Hfos =

   101


num_artifacts =

    99
```

As calculated by the code shown above, the number of HFO samples is 101 and the number of artifact samples is 99.

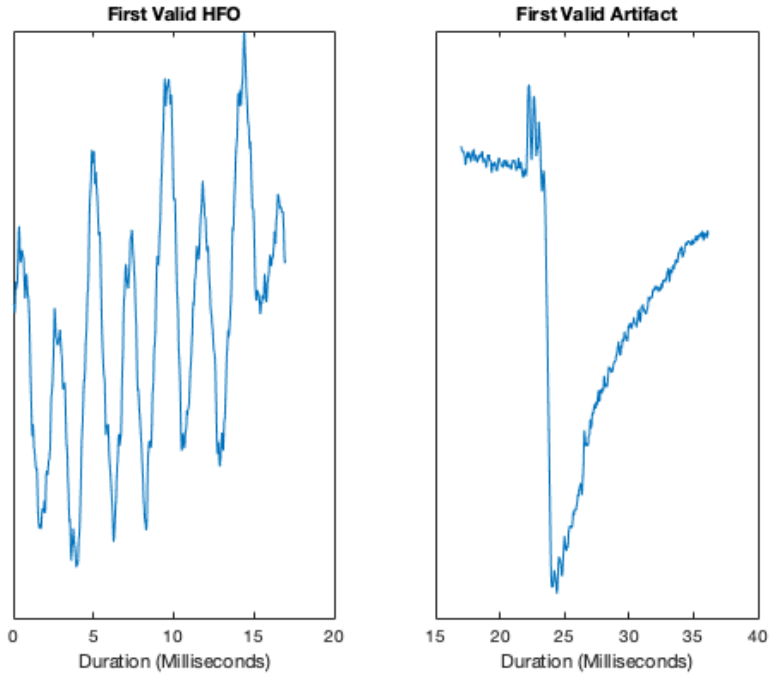
2. Using the training set, find the first occurrence of the first valid HFO and the first artifact. Using **subplot** with 2 plots, plot the valid HFO's (left) and artifact's (right) waveforms. Since the units are normalized, there's no need for a y-axis, so remove it with the command **set(gca,'YTick',[])**.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

```
%find first occurence of first valid HFO and first artifact

%sample rate
sample_rate=32556;

%First valid HFO
first_HFO=allEvents(1);
first_HFO
first_HFO_data=train_data(ceil(allEvents(1).start*1e-6*sample_rate):ceil(allEvents(1).stop*1e-6*sample_rate)

%623
%First valid artifact
first_artifact=allEvents(2);
first_artifact
first_artifact_data=train_data(ceil(allEvents(2).start*1e-6*sample_rate):ceil(allEvents(2).stop*1e-6*sample_



%create time vectors


time_HFO=linspace(1/sample_rate,allEvents(1).stop*1e-6,length(first_HFO_data));


%time_HFO = linspace(1/sample_rate,(length(first_HFO_data)/sample_rate),length(first_HFO_data));
%time_first_artifact = linspace(1/sample_rate,(length(first_artifact_data)/sample_rate),length(first_artifac
%time_first_art=linspace((1/allEvents(2).start),(length(allEvents(2).stop-allEvents(2).start)),length(first_
%time_1_art=allEvents(2).start/sample_rate:1/sample_rate:allEvents(2).stop/sample_rate;

time_1_art=linspace(allEvents(2).start*1e-6,allEvents(2).stop*1e-6,length(first_artifact_data));
```

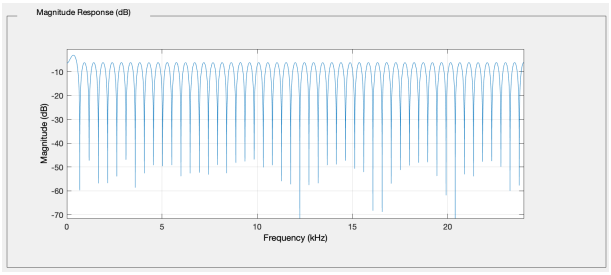
```
first_HFO =

  <a href="matlab:help('IEEGAnnotation')">IEEGAnnotation</a>:

          type: 'trainWin'
   description: '2'
      isGlobal: True
   hasDuration: True
```

```
          start: 1
           stop: 16956
       channels: [1x1 IEEGTimeseries]

  <a href="matlab:methods(IEEGAnnotation)">Methods</a>, <a href="matlab:IEEGObject.openPortalSite()">main.ie


first_artifact =

  <a href="matlab:help('IEEGAnnotation')">IEEGAnnotation</a>:

           type: 'trainWin'
    description: '1'
       isGlobal: True
    hasDuration: True
          start: 16987
           stop: 36123
       channels: [1x1 IEEGTimeseries]

  <a href="matlab:methods(IEEGAnnotation)">Methods</a>, <a href="matlab:IEEGObject.openPortalSite()">main.ie
```

```
%create plot
figure
subplot(1,2,1)
plot(time_HFO*1e3,first_HFO_data)
xlabel('Duration (Milliseconds)')
set(gca,'Ytick',[])
title('First Valid HFO')
subplot(1,2,2)
plot(time_1_art*1e3,first_artifact_data)
title('First Valid Artifact')
xlabel('Duration (Milliseconds)')
set(gca,'Ytick',[])
```



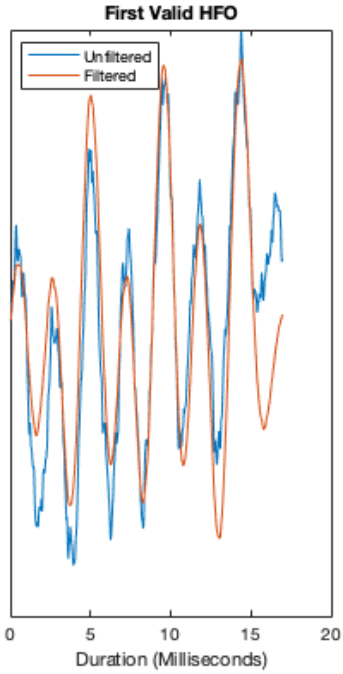
3. Using the `fdatool` in MATLAB, build an FIR bandpass filter of the equiripple type of order 100. Use the Staba et al. (2002) article to guide your choice of passband and stopband frequency. Once set, go to `File -> Export`, and export "Coefficients" as a MAT-File. Attach a screenshot of your filter's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

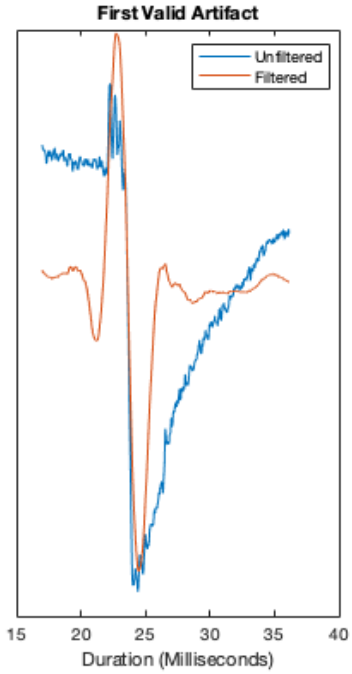


4. Using the forward-backward filter function (`filtfilt`) and the numerator coefficients saved above, filter the valid HFO and artifact clips obtained earlier. You will need to make a decision about the input argument `a` in the `filtfilt` function. Plot these two filtered clips overlayed on their original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

   use filt filt command

```
filtered_HFO=filtfilt(coeff_1,1,first_HFO_data);
filtered_art=filtfilt(coeff_1,1,first_artifact_data);


figure
subplot(1,2,1)
plot(time_HFO*1e3,first_HFO_data)
hold on
plot(time_HFO*1e3,filtered_HFO)
xlabel('Duration (Milliseconds)')
set(gca,'Ytick',[])
title('First Valid HFO')
legend('Unfiltered', 'Filtered','Location','northwest')

subplot(1,2,2)
plot(time_1_art*1e3,first_artifact_data)
hold on
plot(time_1_art*1e3,filtered_art)
xlabel('Duration (Milliseconds)')
title('First Valid Artifact')
legend('Unfiltered', 'Filtered')
set(gca,'Ytick',[])
```





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

   Processing the data using Staba's method may erroneously lead to a false HFO detection because Staba's method invovles filtering the signal to remove frequency components above 500 Hz and below 80 Hz, which may make what is truly artifact appear to look like an HFO. In essence the bandpass filtering that is done in Saba's method will bring the two different types of waves (HFO and Artifacts) closer together in terms of frequency components, which may lead to a false HFO detection.

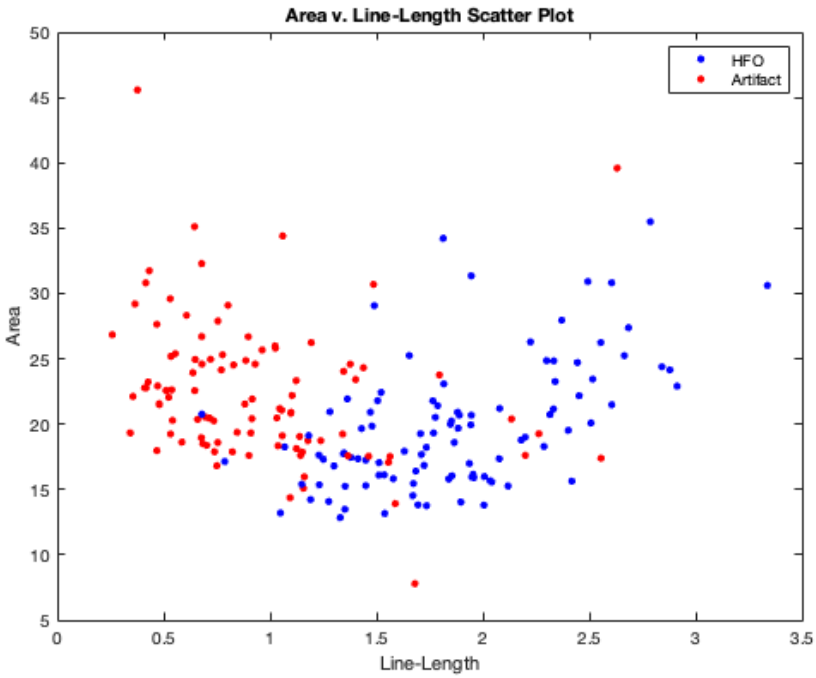
2 Defining Features for HFOs (9 pts)

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)

```
[allEvents_train, timesUSec_train, channels_train]=getAnnotations(session.data,'Training windows');
%run get annotations on the testing data
[allEvents_test, timesUSec_test, channels_test]=getAnnotations(session.data,'Testing windows');
```

```
%420 clips for testing
%200 clips for training

%create trainFeats matrix
trainFeats=zeros(length(allEvents_train),3);% first col is line-length, second is area, third is HFO or arti

%create testFeats matrix
testFeats=zeros(length(allEvents_test),3);

%define line length function
LLfn = @(x) sum(abs(diff(x)));

%define Area feature function
A = @(x) sum(abs(x));

%compute values for trainFeats matrix
for i=1:length(allEvents_train)
    trainFeats(i,1)=LLfn(train_data(ceil(allEvents_train(i).start*1e-6*sample_rate):ceil(allEvents_train(i).
    trainFeats(i,2)=A(train_data(ceil(allEvents_train(i).start*1e-6*sample_rate):ceil(allEvents_train(i).sto
    %characterize HFOs and Artifacts
    if str2num(allEvents_train(i).description) == 1
        trainFeats(i,3)=1; % 1 for artifact
    elseif str2num(allEvents_train(i).description) == 2
         trainFeats(i,3)=2; % 2 for HFO
    end


end


%compute values for testFeats matrix
for i=1:length(allEvents_test)
    testFeats(i,1)=LLfn(test_data(ceil(allEvents_test(i).start*1e-6*sample_rate):ceil(allEvents_test(i).stop
    testFeats(i,2)=A(test_data(ceil(allEvents_test(i).start*1e-6*sample_rate):ceil(allEvents_test(i).stop*1e
    %characterize HFOs and Artifacts
    if str2num(allEvents_test(i).description) == 1
        testFeats(i,3)=1; % 1 for artifact
    elseif str2num(allEvents_test(i).description) == 2
        testFeats(i,3)=2; % 2 for HFO
    end
end

%create scatter plot of the training
figure
for i=1:length(trainFeats)
    if trainFeats(i,3) == 1
        plot(trainFeats(i,1), trainFeats(i,2),'.r','MarkerSize',10)
        hold on
    else
        plot(trainFeats(i,1),trainFeats(i,2),'.b','MarkerSize',10)
        hold on
    end
end
hold off
xlabel('Line-Length')
ylabel('Area')
title('Area v. Line-Length Scatter Plot')
legend('HFO','Artifact')
```



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)

```
%take mean of each column, do element in column minus mean and divide by
%standard deviation

train_z_scores=zeros(length(trainFeats),2);

test_z_scores=zeros(length(testFeats),2);

%compute z scores for train data
for i=1:length(train_z_scores)
    train_z_scores(i,1)=(trainFeats(i,1)-mean(trainFeats(:,1)))/std(trainFeats(:,1)); %compute train li
    train_z_scores(i,2)=(trainFeats(i,2)-mean(trainFeats(:,2)))/std(trainFeats(:,2)); %comput train are
end


%compute z scores for test data
for i=1:length(test_z_scores)
    test_z_scores(i,1)=(testFeats(i,1)-mean(trainFeats(:,1)))/std(trainFeats(:,1)); %compute test line
    test_z_scores(i,2)=(testFeats(i,2)-mean(trainFeats(:,2)))/std(trainFeats(:,2)); %comput test area z
end
```

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

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

This feature normalization we have just completed will be critical to the performance of a KNN classifier because it ensures that all features used to train your KNN classifier on on the same scale. You want to ensure that changes in features used to train the KNN classifier are all equally weighted, this is important because the KNN classifier measures distances between points in the feature space to classify them. So, if these distances are on different scales then more weight would be erroneously given to features that are on bigger scales, even if they don't influence what the model is trying to predict even as much as features on smaller scales. This can be thought of as analagous to not adding two quantities together that have different units, if you did that your answer would make no physical sense.

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

You use the training feature means and standard deviations to normalize the testing set because (philosophically) you do not have access to the testing dataset when you are creating a machine learning classifier, you only have access to the training dataset. For example if you are trying to actually predict a future outcome your testing dataset could be a dataset from the future (which doesn't exist yet and you don't have access to) so you need to train your model on the present data available. Thus, you would normalize the testing set to the training feature means and standard deviations, because those are the only means and standard deviations you have access to. Additionally, we can think of the classifier as classifying one testing observation at a time, so we can't normalize a single point and must use the training feature means and standard deviations.

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)

```
%run mnr fit model
[B,dev,stats]=mnrfit((train_z_scores(:,1:2)),trainFeats(:,3))

pihat=mnrval(B,(train_z_scores(:,1:2)));

pihat_percentage=pihat*100;

[~, Ypred]=max(pihat_percentage, [], 2); %gets the column index of the maximumum of each row

%first column is 1 (artifact) second column is 2 (HFO)

%calculate training error

number_wrong_train=0;
for i=1:length(Ypred)
```

[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.

```
    if Ypred(i) ~= trainFeats(i,3)
        number_wrong_train=number_wrong_train+1;
    end

end

%calculate training error as a percent
training_error=(number_wrong_train/length(trainFeats))*100
```

```
B =

    0.0499
   -2.4386
    0.8806


dev =

  144.2538


stats =

  struct with fields:

         beta: [3×1 double]
          dfe: 197
         sfit: 1.2636
            s: 1
      estdisp: 0
         covb: [3×3 double]
    coeffcorr: [3×3 double]
           se: [3×1 double]
            t: [3×1 double]
            p: [3×1 double]
        resid: [200×1 double]
       residp: [200×1 double]
       residd: [200×1 double]


training_error =

   12.5000
```

The training error is 12.5 percent.

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)

   Use the model trained on training data to predict labels of the test samples

```
pihat_test=mnrval(B,(test_z_scores(:,1:2)));

pihat_percentage_test=pihat_test*100;

[~, Ypred_test]=max(pihat_percentage_test, [], 2);

%calculate the testing error
number_wrong_test=0;
for i=1:length(Ypred_test)
```

```
    if Ypred_test(i) ~= testFeats(i,3)
        number_wrong_test=number_wrong_test+1;
    end

end

%calculate training error as a percent
testing_error=(number_wrong_test/length(testFeats))*100
```

```
testing_error =

   13.5714
```

The testing error is 13.5714 percent while the training error is 12.5 percent. So, the testing error is slightly larger than the training error. The testing error is slightly larger because the model was trained on the training data, so inherently the model is biased towards the training data and better predicting the data it was trained on. When presented with a novel testing dataset, the model has to use the relationships it derived from the training set and apply it to the testing set. Since the testing set will not be identical to the training set, the testing error will be higher.

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)

```
%create KNN model
Mdl=fitcknn((train_z_scores(:,1:2)),trainFeats(:,3),'NumNeighbors',1);
```

```
%make predictions for training data

label_train=predict(Mdl,(train_z_scores(:,1:2)));


%calculate the training error
number_wrong_test=0;
for i=1:length(label_train)
    if label_train(i) ~= trainFeats(i,3)
        number_wrong_test=number_wrong_test+1;
    end

end

%calculate training error as a percent
training_error=(number_wrong_test/length(trainFeats))*100


label_test=predict(Mdl,(test_z_scores(:,1:2)));

%calculate the testing error
number_wrong_test=0;
for i=1:length(label_test)
    if label_test(i) ~= testFeats(i,3)
        number_wrong_test=number_wrong_test+1;
    end

end

%19

%calculate training error as a percent
```

```
testing_error=(number_wrong_test/length(testFeats))*100
```

```
training_error =

     0


testing_error =

   17.3810
```

The training error is 0 percent. The testing error is 17.3810 percent.

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

The training error is zero because the number of neighbors (k) is specified to be one in the default parameters of the KNN classifer model. Since k=1, the KNN classifier is simply memorizing the training dataset and thus it will always accurately predicted the training data. Each datapoint in the training set will just be classified as the label it has, since it is its only and nearest neighbor.

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)

Run svm model on train data

```
svm_mdl=fitcsvm((train_z_scores(:,1:2)),trainFeats(:,3),'KernelFunction','RBF');


%generate training svm predictions
svm_predictions_train=predict(svm_mdl,(train_z_scores(:,1:2)));
```

```
%calculate the training error
number_wrong=0;
for i=1:length(svm_predictions_train)
    if svm_predictions_train(i) ~= trainFeats(i,3)
        number_wrong=number_wrong+1;
    end

end

%calculate training error as a percent
training_error=(number_wrong/length(trainFeats))*100
```

```
training_error =

    10
```

```
%generate svm testing predictions
svm_predictions_test=predict(svm_mdl,(test_z_scores(:,1:2)));
```

[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).

```
%calculate test error

%calculate the testing error
number_wrong=0;
for i=1:length(svm_predictions_test)
    if svm_predictions_test(i) ~= testFeats(i,3)
        number_wrong=number_wrong+1;
    end

end

%calculate training error as a percent
testing_error=(number_wrong/length(testFeats))*100
```

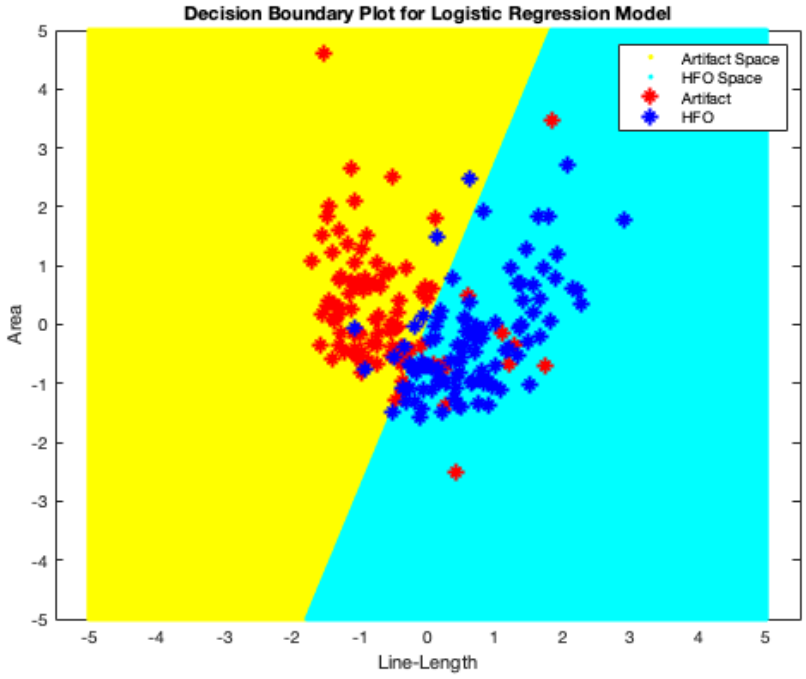
```
testing_error =

   11.6667
```

The SVM classifier training error is 10 percent and the SVM classifier testing error is 11.6667 percent.

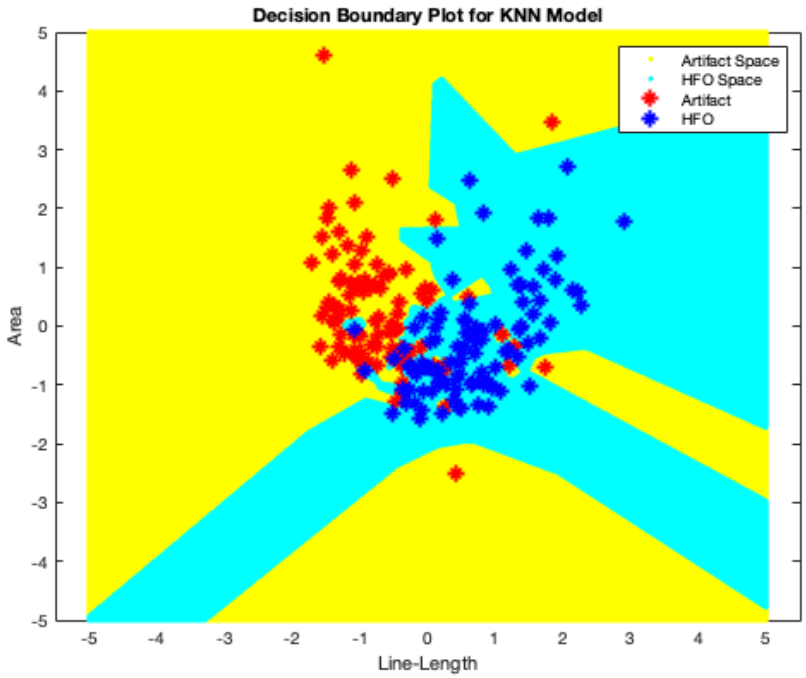
5. It is sometimes useful to visualize the decision boundary of a classifier. To do this, we'll plot the classifier's prediction value at every point in the "decision" space. Use the `meshgrid` function to generate points in the line-length and area 2D feature space and a scatter plot (with the `'.'` 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)

```
% generate points using meshgrid

x_values=-5:0.01:5; %specify x values

y_values=-5:0.01:5; %specify y values

[X,Y]=meshgrid(x_values,y_values);

% run meshgrid through the model, color based on the prediction

%run meshgrid through the mnr fit model
meshgrid_mat=[X(:),Y(:)];
pihat_mesh=mnrval(B,(meshgrid_mat));

pihat_percentage_mesh=pihat_mesh*100;

[~, Ypred_mnr]=max(pihat_percentage_mesh, [], 2); %gets the column index of the maximumum of each row

plot_train_z_scores_X=[];

plot_train_z_scores_Y=[];

plot_groups_train=[];

for i=1:length(train_z_scores)
    plot_train_z_scores_X(i)=train_z_scores(i,1);
    plot_train_z_scores_Y(i)=train_z_scores(i,2);
    plot_groups_train(i)=trainFeats(i,3);
end

%create plot
```

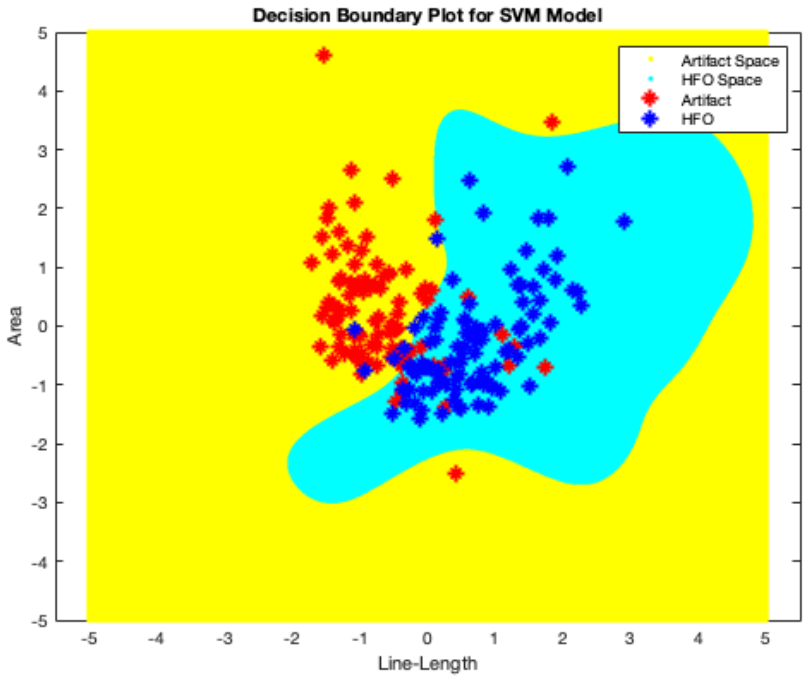
```
figure
gscatter(X(:),Y(:),Ypred_mnr,'yc')
hold on
gscatter(plot_train_z_scores_X(:),plot_train_z_scores_Y(:),plot_groups_train(:),'rb','*')
legend('Artifact Space','HFO Space','Artifact','HFO','Location','northeast')
xlabel('Line-Length')
ylabel('Area')
title('Decision Boundary Plot for Logistic Regression Model')
```




```
%create plot for KNN
label_train_mesh=predict(Mdl,meshgrid_mat);

figure
gscatter(X(:),Y(:),label_train_mesh,'yc')
hold on
gscatter(plot_train_z_scores_X(:),plot_train_z_scores_Y(:),plot_groups_train(:),'rb','*')
legend('Artifact Space','HFO Space','Artifact','HFO','Location','northeast')
xlabel('Line-Length')
ylabel('Area')
title('Decision Boundary Plot for KNN Model')
```




```
%create plot for SVM
svm_predictions_test_mesh=predict(svm_mdl,meshgrid_mat);

figure
gscatter(X(:),Y(:),svm_predictions_test_mesh,'yc')
hold on
gscatter(plot_train_z_scores_X(:),plot_train_z_scores_Y(:),plot_groups_train(:),'rb','*')
legend('Artifact Space','HFO Space','Artifact','HFO','Location','northeast')
xlabel('Line-Length')
ylabel('Area')
title('Decision Boundary Plot for SVM Model')
```



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)

   These three plots differ greatly in their appearance. The logistic regression plot is divided by the classifer into the HFO and Artifact space by a line which cleanly classifies HFOs and Artifacts into two planes distinct planes (artifact on the left HFO on the right). But, many points are incorrectly classified by the logistic regression. However, the KNN model has many distinct artifact spaces and one distinct HFO space. This is because KNN classifies points based on the points around the point in question. If the K in the KNN classifier was increased (up to a suitable number), more data points would be looked at and there would be less distinct artifact and HFO spaces. Since k=1 in our KNN model here, each point in the training set (shown here) will be correctly classified, however there is alot of overfitting going on. This is because overfitting is when the classifier is too specific to the training data, which means it will break up the 2-D feature space into too many distinct HFO and artifact subspaces. In contrast, the SVM plot has only one distinct HFO space and one distinct artifact space, and these two spaces capture the large majority of the correct training labels. This is a better model than the KNN one shown because there is less overfitting going on and the SVM model will generalize better to testing data as the model is more rigorous. So, the KNN model overfits the data the most and the logistic regression model underfits the data the most. The SVM model produces the least amount of overfitting and underfitting (it is the best model of the three computed).

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)

Create a cell array folds that contains 10 elements, each of which itself is an array of the indices of training samples in that fold

```
permutations=randperm(200,200);

%perm_reshape=reshape(permutations,[

folds=cell(10,1);

folds{1}=permutations(1:20);
folds{2}=permutations(21:40);
folds{3}=permutations(41:60);
folds{4}=permutations(61:80);
folds{5}=permutations(81:100);
folds{6}=permutations(101:120);
folds{7}=permutations(121:140);
folds{8}=permutations(141:160);
folds{9}=permutations(161:180);
folds{10}=permutations(181:200);

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)

```
%We need to iterate through the folds and train on all the folds that aren't the index we are on
shifted_num_array=folds;
%number_wrong=0;
error_list=[];
for i=1:length(folds)
   training_set=[shifted_num_array{1:9}];

    Mdl_folds=fitcknn(train_z_scores(training_set,1:2),trainFeats(training_set,3));
    predicted_fold=predict(Mdl_folds,(train_z_scores(shifted_num_array{10}(1:20),1:2)));
    compare=trainFeats(shifted_num_array{10}(1:20),3);
    error_list(i)=((sum(predicted_fold~=compare))/20)*100;

    shifted_num_array=circshift(shifted_num_array,1);
end

validation_error=mean(error_list)
```

[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];`.

```
validation_error =

   19.5000
```

The validation error for the folds computed on this iteration is 20.5 percent.

(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)

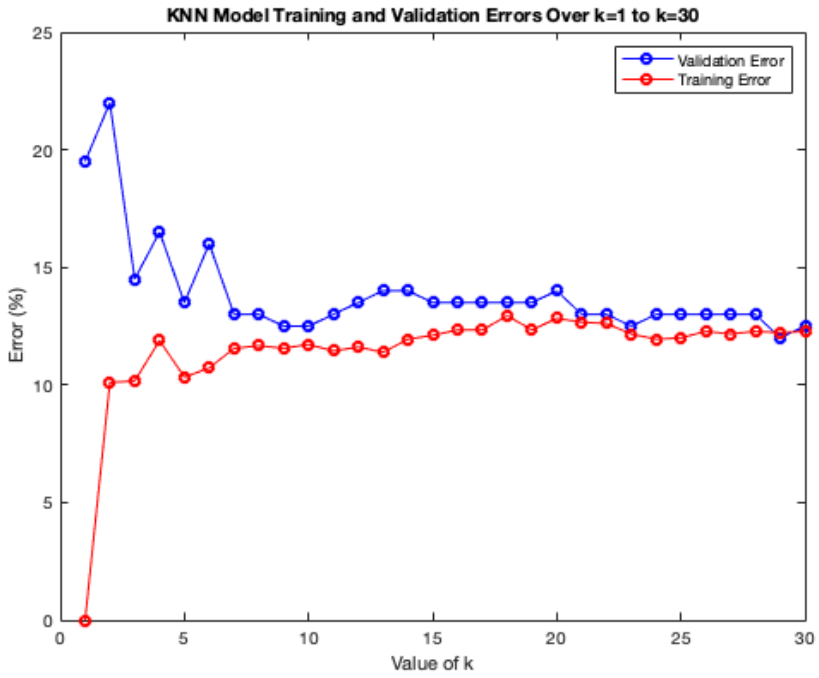
The validation error is 20.5 percent, which is higher than the error found in 3.3 which was 17.3810 percent. This makes sense because you are only using 180 datapoints to train your model with this folds method, as opposed to all 200 datapoints used to train the model in 3.3. A higher number of training points will lead to a lower training error because the classifier will have more data to compare against and can thus create a more rigorous model that will generalize better.

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)

```
% %We need to iterate through the folds and train on all the folds that aren't the index we are on
shifted_num_array=folds;
number_wrong=0;
error_list=[];
train_list=[];
final_valid_error_list=[];
training_error_list=[];
for k=1:30
    for i=1:length(folds)
        training_set=[shifted_num_array{1:9}];

        Mdl_folds=fitcknn(train_z_scores(training_set,1:2),trainFeats(training_set,3),'NumNeighbors',k)
        predicted_fold_train=predict(Mdl_folds,(train_z_scores(training_set,1:2)));
        compare_train=(trainFeats(training_set,3));
        train_list(i)=((sum(predicted_fold_train~=compare_train))/180)*100;

        predicted_fold=predict(Mdl_folds,(train_z_scores(shifted_num_array{10}(1:20),1:2)));
        compare=trainFeats(shifted_num_array{10}(1:20),3);
        error_list(i)=((sum(predicted_fold~=compare))/20)*100;

        shifted_num_array=circshift(shifted_num_array,1);
    end
    final_valid_error_list(k)=mean(error_list);
    train_error_list(k)=mean(train_list);
end
```

```
%create plot

k=1:30;

figure
plot(k,final_valid_error_list,'b-o')
hold on
plot(k,train_error_list,'r-o')
xlabel('Value of k')
ylabel('Error (%)')
title('KNN Model Training and Validation Errors Over k=1 to k=30')
legend('Validation Error','Training Error')
```



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

```
optimal_valid_error=final_valid_error_list(30)
optimal_train_error=train_error_list(30)
```

```
optimal_valid_error =

   12.5000


optimal_train_error =

   12.2778
```

The optimal k is k=30. This value was chosen because this value of k gave the lowest validation error when many different sets of permutations were computed. It is possible to get a lower validation error at a lower value of k (I have observed somewhere in the range between k=8 and k =15), but the variation in the value of k makes a choice in this range suboptimal. For example, if you pick k=10 as the optimal value based on k=10 giving the lowest validation error for a certain set of folds, you may actually get a validation error higher than k=30 when the folds are computed again. Therefore, since the difference between the minmimum validation error and the validation error produced by k=30 has been observed to be quite small (approximately less than 3%), k=30 is the optimal value as you can be most sure it will consistently give a validation within approximately 3% of the lowest validation error possible with this model. The validation error for k=30 (with the current folds computed) is 12.5% and the training error for k=30 is 12.28%. It should be noted that, in general, as k increases error does not decrease but since we are only going up to k=30 the argument presented is valid.

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

KNN generally overfits less with higher values of k because as k increases the KNN classifier considers more and more datapoints when it makes a decision on any the classification of any one datapoint. As such, the feature space it is classifying will be broken up into fewer distinct classification spaces so the training data will be "memorized" less and general patterns of datapoints will be deduced more often. We see that when k=1 the KNN only considers the point it is classifying at the time when it makes a classification decision, leading to overfitting as the feature space is paritioned too many times. So, as k increases more information will be taken into account and the general trends inherent in the data will be better incapsulated, creating a more rigorous model. It is important to note that the model does have not decreased error as k continues to increase (there exists an optimal value for k).

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

```
%train on the whole training set with k=30
optimal_k_model=fitcknn(train_z_scores(:,1:2),trainFeats(:,3),'NumNeighbors',30);

%predict on the test data
predicted_optimal_k=predict(optimal_k_model,(test_z_scores(:,1:2)));



%calculate the testing error
number_wrong_test=0;
for i=1:length(predicted_optimal_k)
    if predicted_optimal_k(i) ~= testFeats(i,3)
        number_wrong_test=number_wrong_test+1;
    end

end



%calculate testing error as a percent
testing_error_optimal=(number_wrong_test/length(testFeats))*100
```

```
testing_error_optimal =

   11.4286
```

The testing error with the optimal k (k=30) was calculated to be 11.4286 percent.

(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)

This model's testing error is lower than the model trained in question 3.3, which had a testing error of 17.3810 percent. So, this model is the best of the three models trained in Section 3.3 (in 3.3 SVM had a testing error of 11.667 percent and logistic regression had a testing error of 13.5714 percent, so this model's error is slightly lower than these 3 other models).