BE 521: Homework 4

HFOs

Spring 2021

58 points

Due: Tuesday, 2/23/2021 10:00pm

Objective: HFO detection and cross-validation

Saif Khawaja
Collaborators: Raveen K

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 `sesh` 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)

```
sesh = IEEGSession('I521_A0004_D001', 'saifkhawaja', '/Users/saif/Documents/GitHub/braincomputerinterfaces/H

nr = ceil((sesh.data.rawChannels(1).get_tsdetails.getEndTime)/1e6*sesh.data.sampleRate);
test_data = sesh.data.getvalues(1:nr,1);
sesh.data;

nr = ceil((sesh.data.rawChannels(2).get_tsdetails.getEndTime)/1e6*sesh.data.sampleRate);
tdt = sesh.data.getvalues(1:nr,2);
sesh.data;

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

hfos = 0;
artfx = 0;

descr = zeros(1,length(allEvents));

for i = 1:length(allEvents)
    sd = allEvents(i).description;
    sed = str2num(sd);
    descr(i) = sed;
end

for j = 1:length(descr)
    if descr(j) == 1
        artfx = artfx + 1;
    else
        hfos = hfos + 1;
    end
end

artfx
hfos

% HFO: 101; Artifacts: 99.
```

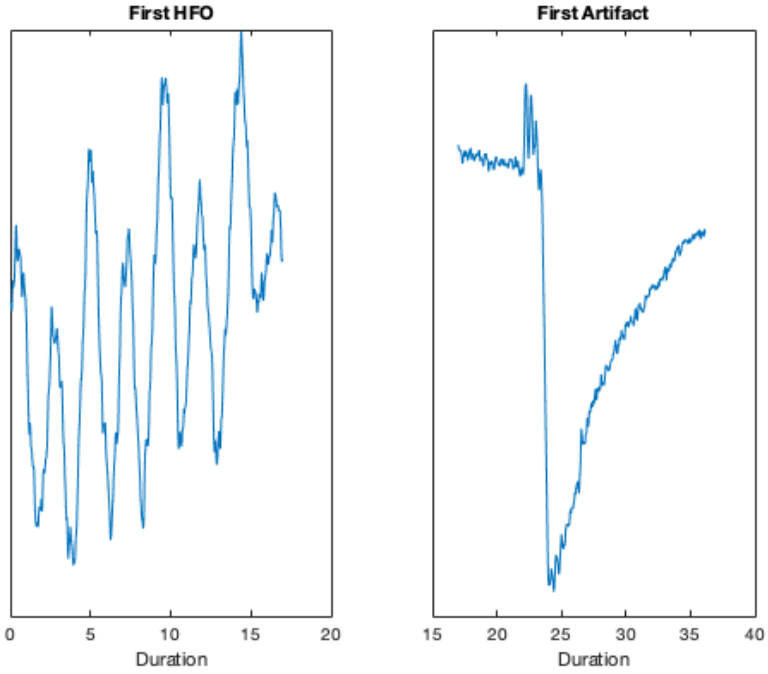
```
Warning: Objects of edu/upenn/cis/db/mefview/services/TimeSeriesDetails class
exist - not clearing java
Warning: Objects of edu/upenn/cis/db/mefview/services/TimeSeriesInterface class
exist - not clearing java
IEEGSETUP: Found log4j on Java classpath.
URL: https://www.ieeg.org/services
Client user: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

artfx =

    99


hfos =

   101
```

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)

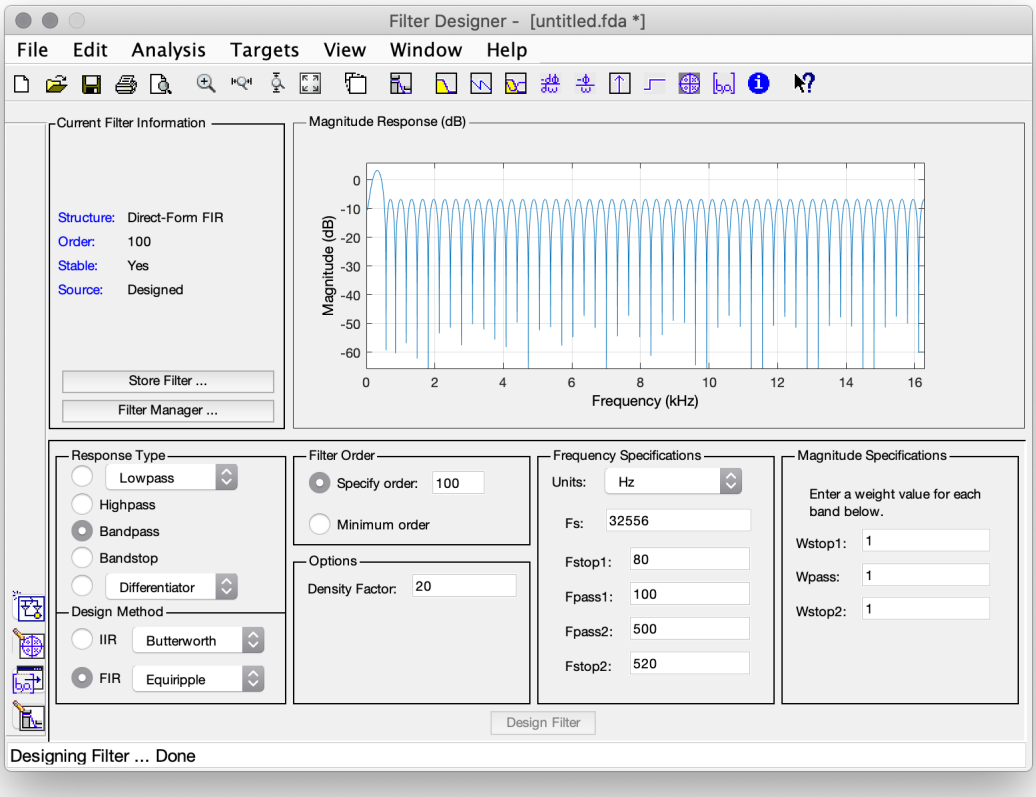
```
allEvents(1:5).description % for first data points for 1st HFO and artifact

sr = 32556;

fhfo = allEvents(1);
fhfod = tdt(ceil(allEvents(1).start * 1e-6 * sr):ceil(allEvents(1).stop * 1e-6 * sr));

fa = allEvents(2).start;
fa = allEvents(2).stop;
fad = tdt(ceil(allEvents(2).start * 1e-6 * sr):ceil(allEvents(2).stop * 1e-6 * sr));

t1a = linspace(allEvents(2).start * 1e-6, allEvents(2).stop * 1e-6, length(fad));
thfo = linspace(1/sr,allEvents(1).stop*1e-6,length(fhfod));

figure
subplot(1,2,2)
plot(t1a * 1e3, fad)
title('First Artifact')
xlabel('Duration')
set(gca,'Ytick',[])
subplot(1,2,1)
plot(thfo * 1e3, fhfod)
xlabel('Duration')
set(gca,'Ytick',[])
title('First HFO')
```

```
ans =

    '2'


ans =

    '1'


ans =

    '1'


ans =

    '1'


ans =

    '2'
```



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)

```
% fdatool
```



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)

```
%{

Collaborator and I were unable to get this working because of filtfilt.

%}
```

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

```
% We may incorrectly detect HFOs because of the removal of frequency by the
% filter with a band of 80 and 500 HZ - this makes an artifact a false
% positive of an HFO. The method collects both wave types with regards to
% frequency and leads to incorrect or misrepresented HFOs.
```

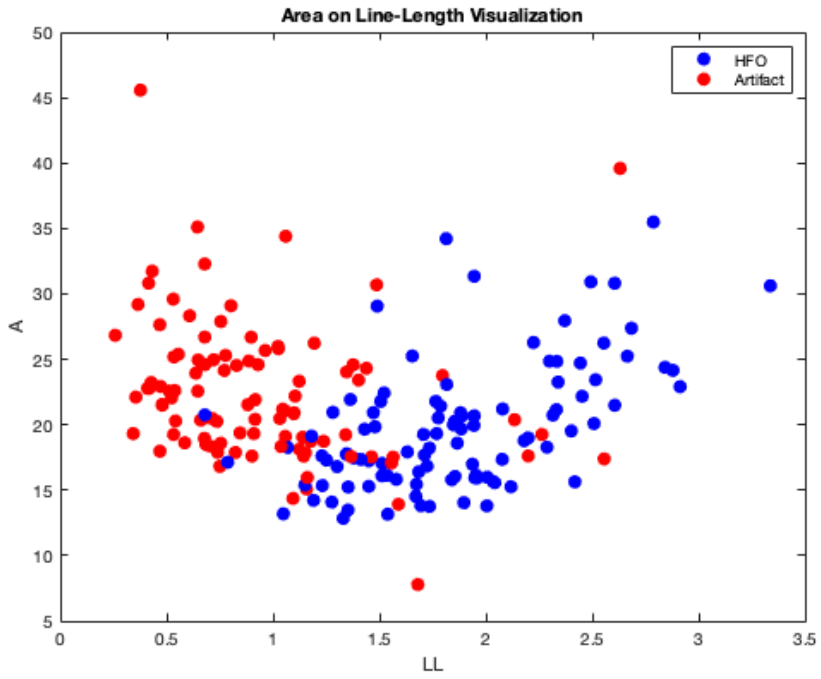
2 Defining Features for HFOs (9 pts)

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, `feats` 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)

```
sr = 32556;

a = @(x) sum(abs(x));

Llf = @(x) sum(abs(diff(x)));


[allEvents_train, timesUSec_train, channels_train]=getAnnotations(sesh.data,'Training windows');

[allEvents_test, timesUSec_test, channels_test]=getAnnotations(sesh.data,'Testing windows');


feats = zeros(length(allEvents_train), 3);

tfeats = zeros(length(allEvents_test), 3);


for i = 1:length(allEvents_train)
    feats(i,1) = Llf(tdt(ceil(allEvents_train(i).start*1e-6*sr):ceil(allEvents_train(i).stop*1e-6*sr)));
    feats(i,2) = a(tdt(ceil(allEvents_train(i).start*1e-6*sr):ceil(allEvents_train(i).stop*1e-6*sr)));
    if str2num(allEvents_train(i).description) == 1
        feats(i,3)=1;
    elseif str2num(allEvents_train(i).description) == 2
         feats(i,3)=2;
    end
end

figure

for i=1:length(feats)
    if feats(i,3) == 1
        plot(feats(i,1), feats(i,2), '.r', 'MarkerSize', 20)
        hold on
    else
        plot(feats(i,1), feats(i,2), '.b', 'MarkerSize', 20)
        hold on
    end
end

for i=1:length(allEvents_test)
    tfeats(i,1)=Llf(test_data(ceil(allEvents_test(i).start*1e-6*sr):ceil(allEvents_test(i).stop*1e-6*sr)));
    tfeats(i,2)=a(test_data(ceil(allEvents_test(i).start*1e-6*sr):ceil(allEvents_test(i).stop*1e-6*sr)));

    if str2num(allEvents_test(i).description) == 1
        tfeats(i,3)=1;
    elseif str2num(allEvents_test(i).description) == 2
```

```
        tfeats(i,3)=2;
    end
end

hold off
xlabel('LL')
ylabel('A')
title('Area on Line-Length Visualization')
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)
   Computing Z Scores (statistical term):

```
z_train = zeros(length(feats),2);

z_test = zeros(length(tfeats),2);

for i=1:length(z_test)
    z_test(i, 1) = (tfeats(i,1) - mean(feats(:,1))) / std(feats(:,1));
    z_test(i, 2) = (tfeats(i,2) - mean(feats(:,2))) / std(feats(:,2));
end


for i=1:length(z_train)
    z_train(i, 1)=(feats(i, 1) - mean(feats(:,1))) / std(feats(:,1));
    z_train(i, 2)=(feats(i, 2) - mean(feats(:,2))) / std(feats(:,2));
```

```
end
```

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

```
% We must normalize features because the k-NN classifier, which uses
% distance across the feature space for classification, needs to have a
% normalized distance length for correct classification. The changes in
% features should all be weighted appropriately, as, if not, the classifier
% would falsely allocate weight to particular features that are larger,
% when they may have no effect on the results of the model when
% inappropriately smaller ones do. The comparisons are not on the same
% scale and therefore woudl not create realistic accuracy or a high performing
% model.
```

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

```
% The mean and standard deviation are used because one cannot use the
% testing dataset for the classifier - we can only use the separated
% training dataset. Potential predictions are only based off of incumbent
% data and not future-generated data. The testing set therefore has to
% normalize on the average and spread of its own statistics.
```

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, k-NN, and SVM classifiers are functions built into some of Matlabs statistics packages. If you don't have these (i.e., Matlab doesn't recognize the functions), and you are experiencing any difficulty downloading the necessary packages, 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)

```
[B,dev,stats] = mnrfit((z_train(:,1:2)),feats(:,3))

incorrect = 0;

ph2 = mnrval(B,(z_train(:,1:2)));

php = ph2*100;

[~, Ypred] = max(php, [], 2);

for i=1:length(Ypred)
    if Ypred(i) ~= feats(i,3)
        incorrect=incorrect+1;
    end
end
```

[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.

```
errortraining = (incorrect / length(feats)) * 100

% 12.5% error.
```

```
B =

    0.0499
   -2.4386
    0.8806


dev =

  144.2538


stats =

  struct with fields:

         beta: [U+FFFD]double]
          dfe: 197
         sfit: 1.2636
            s: 1
      estdisp: 0
         covb: [U+FFFD]double]
    coeffcorr: [U+FFFD]double]
           se: [U+FFFD]double]
            t: [U+FFFD]double]
            p: [U+FFFD]double]
        resid: [2[U+FFFD]double]
       residp: [2[U+FFFD]double]
       residd: [2[U+FFFD]double]


errortraining =

   12.5000
```

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)

```
phtest = mnrval(B, (z_test(:,1:2)));

phptest = phtest * 100;

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

incorrect_test=0;
for i=1:length(Ypred_test)
    if Ypred_test(i) ~= tfeats(i,3)
        incorrect_test=incorrect_test+1;
    end

end

errortesting = (incorrect_test / length(tfeats)) * 100
```

```
errortesting =

   13.5714
```

```
% The error of the testing set is slightly larger than the training set (13.5714% vs 12.5%),
% which likely arises due to the model training on the training data as
% opposed to the testing set and therefore parameters and statistics are
% based on a different set. It is intrinsit to any model that it performs
% better on data it has been trained on as opposed to an alternative set -
% the links between sets are inferred and so the error of the testing set
% with the training model 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)

```
incorrect_test = 0;

Mdl = fitcknn((z_train(:,1:2)), feats(:,3), 'NumNeighbors', 1);

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

for i=1:length(label_train)
    if label_train(i) ~= feats(i,3)
        incorrect_test = incorrect_test+1;
    end

end

errortraining = (incorrect_test / length(feats)) * 100


labelt = predict(Mdl,(z_test(:,1:2)));

incorrect_test=0;

for i=1:length(labelt)
    if labelt(i) ~= tfeats(i,3)
        incorrect_test = incorrect_test+1;
    end

end

errortesting = (incorrect_test / length(tfeats)) * 100

% 0% Training Error. 17.3810% Testing Error.
```

```
errortraining =

     0


errortesting =

   17.3810
```

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

```
% The k-NN classifier requires an input number of neighbours (k), and since
% the input is 1, the model memorizes the input dataset and so will not
% make any incorrect classifications. Each point will be identified as its
% label given the algorithm.
```

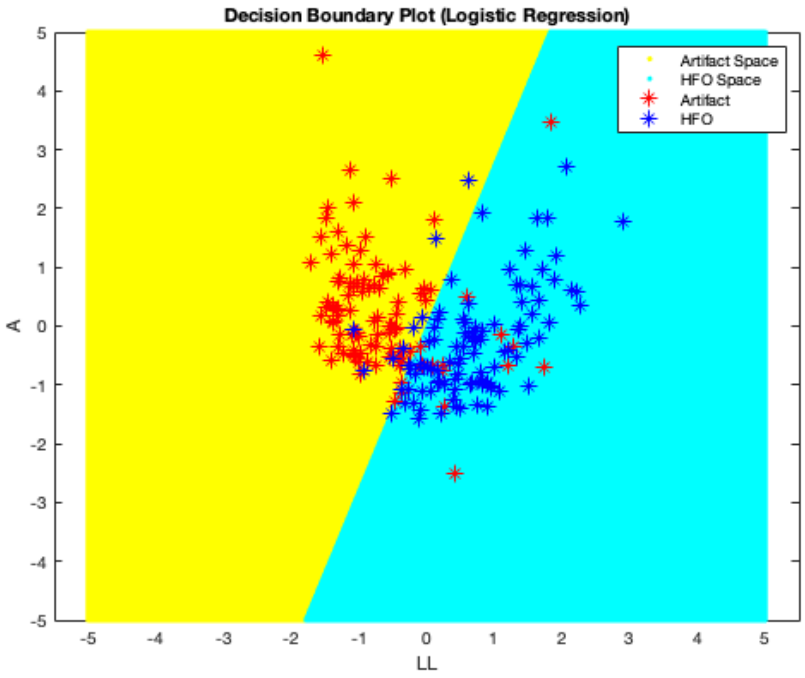
4. Now, train Matlab's implementation of a support vector machine (SVM), `fitcsvm`. Report the training and testing errors for an SVM model with a radial basis function (RBF) kernel function, while keeping other parameters at their default values. (3 pts)

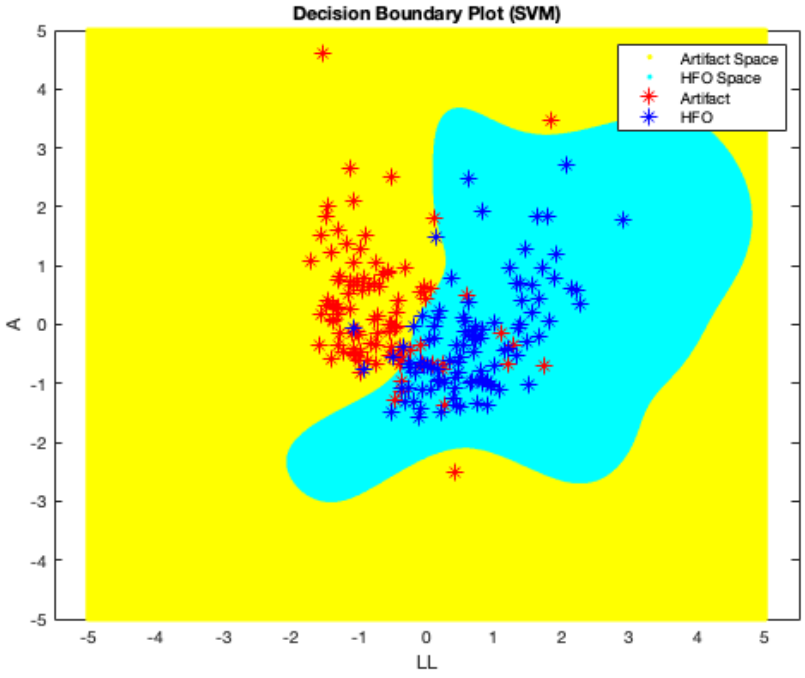
```
incorrect = 0;

svm = fitcsvm((z_train(:,1:2)),feats(:,3), 'KernelFunction', 'RBF');

svm_pdt = predict(svm, (z_train(:,1:2)));

for i=1:length(svm_pdt)
    if svm_pdt(i) ~= feats(i,3)
        incorrect=incorrect+1;
    end

end

errortraining = (incorrect / length(feats)) * 100

svm_predictions_test=predict(svm, (z_test(:,1:2)));

incorrect = 0;

for i=1:length(svm_predictions_test)
    if svm_predictions_test(i) ~= tfeats(i,3)
        incorrect = incorrect+1;
    end

end

errortesting = (incorrect / length(tfeats)) * 100

% Training error: 10%; Testing error is 11.6667%.
```

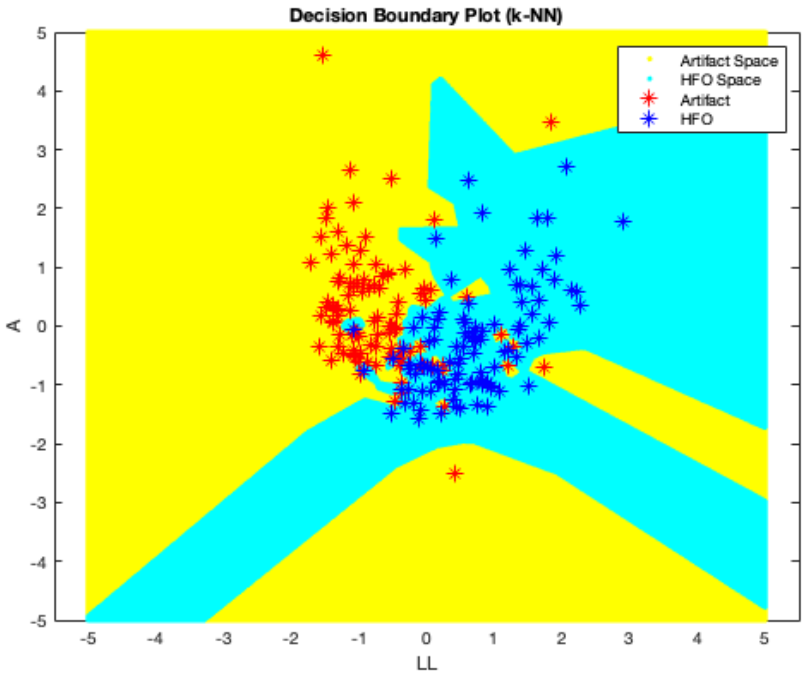
```
errortraining =

    10


errortesting =

   11.6667
```

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)

```
yv = -5:0.01:5;

xv = -5:0.01:5;

[X,Y] = meshgrid(xv, yv);

mmat = [X(:), Y(:)];

ph_mesh = mnrval(B, (mmat));
ph_pmesh = ph_mesh * 100;

[~, Ypred_mnr]=max(ph_pmesh, [], 2);

ptzY = [];
ptzX = [];
ptzt = [];

for i=1:length(z_train)
    ptzY(i) = z_train(i,2);
    ptzX(i) = z_train(i,1);
    ptzt(i) = feats(i,3);
end

figure
gscatter(X(:), Y(:), Ypred_mnr, 'yc')
hold on
gscatter(ptzX(:), ptzY(:), ptzt(:), 'rb', '*')
legend('Artifact Space','HFO Space','Artifact','HFO','Location','northeast')
xlabel('LL')
ylabel('A')
title('Decision Boundary Plot (Logistic Regression)')

svm_ptm = predict(svm, mmat);

figure
gscatter(X(:),Y(:),svm_ptm,'yc')
hold on
gscatter(ptzX(:),ptzY(:),ptzt(:),'rb','*')
legend('Artifact Space','HFO Space','Artifact','HFO','Location','northeast')
xlabel('LL')
ylabel('A')
title('Decision Boundary Plot (SVM)')

labeltm = predict(Mdl, mmat);

figure
gscatter(X(:),Y(:), labeltm,'yc')
hold on
gscatter(ptzX(:),ptzY(:),ptzt(:),'rb','*')
legend('Artifact Space','HFO Space','Artifact','HFO','Location','northeast')
xlabel('LL')
ylabel('A')
title('Decision Boundary Plot (k-NN)')
```


Decision Boundary Plot (Logistic Regression)
Artifact Space
HFO Space
Artifact
HFO
A
LL
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
-4
-3
-2
-1
0
1
2
3
4
5


Decision Boundary Plot (SVM)
Artifact Space
HFO Space
Artifact
HFO
A
LL
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
-4
-3
-2
-1
0
1
2
3
4
5



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)

```
% The plots reflect differentiated feature spaces. The logistic regression
% only divides the HFO and artifact space by a line that gives rise to two
% distinct planes. Many of the classifications are incorrectly identified
% because of that.

% The k-NN model has distinct spaces because of the intrinsic neighbour
% identification (points around the points). k = 1 is overfitting, however,
% and higher k values will give rise to a less specialized model that would
% also work more consistently with non-training data.

% The SVM is the most successful by both reflecting the data labels and prediction
% of data points. There is less overfitting and therefore the model has
% generalized given its complexity.

% In conclusion, the SVM model is accurate, the k-NN overfits and the logistic regression
% underfits. All models demonstrate HFO clutsering in the upper right.
```

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[2] `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)

```
folds=cell(10,1);
permutations=randperm(200,200);


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{:}]))

% 200 as suggested.
```

```
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)

```
narray = folds;

lerror = [];
for i=1:length(folds)

   training_set=[narray{1:9}];

    Mdl_folds=fitcknn(z_train(training_set,1:2),feats(training_set,3));
    predicted_fold=predict(Mdl_folds,(z_train(narray{10}(1:20),1:2)));
    compare=feats(narray{10}(1:20),3);
```

[2] 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];`.

```
    lerror(i)=((sum(predicted_fold~=compare))/20)*100;

    narray=circshift(narray,1);
end

error_validation = mean(lerror)
```

```
error_validation =

   20.5000
```

```
% Validation error: 18%.
```

(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)

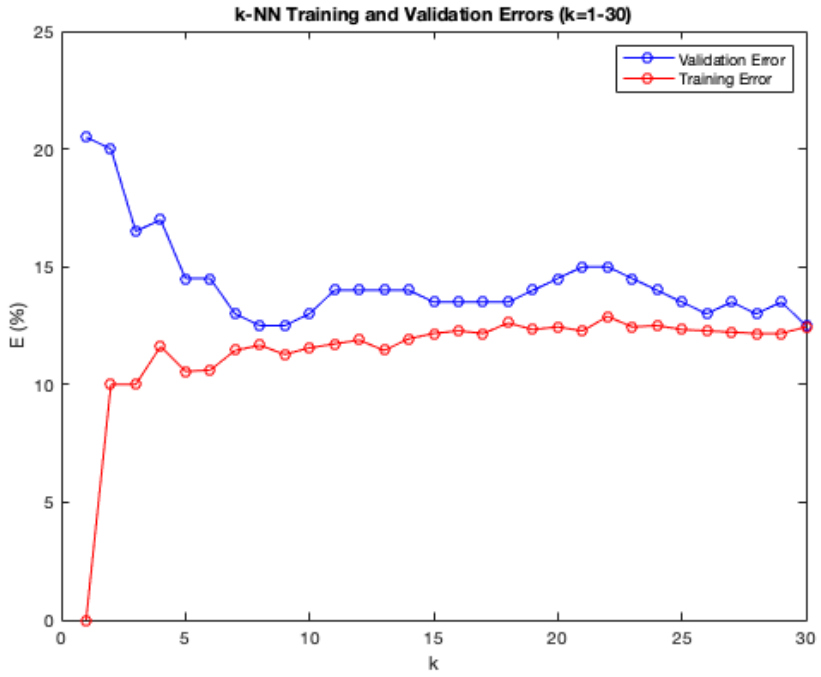
```
% 20% is much larger than 17.3810%. The comparative difference of 180 and
% 200 datapoints means that the latter's inclusion of extra data points
% leads to a smaller error with the training of the model having more
% information to compare against.
```

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)

```
narray = folds;

lerror = [];

trainl = [];
ferrorl = [];
terrorl = [];

incorrect = 0;

for k=1:30
    for i=1:length(folds)

        training_set=[narray{1:9}];

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

        predicted_fold=predict(Mdl_folds,(z_train(narray{10}(1:20),1:2)));
        compare=feats(narray{10}(1:20),3);
        lerror(i)=((sum(predicted_fold~=compare))/20)*100;

        narray=circshift(narray,1);
    end
    ferrorl(k)=mean(lerror);
    train_error_list(k)=mean(trainl);
end
```

```
k = 1:30;

figure
plot(k,ferrorl,'b-o')
hold on
plot(k,train_error_list,'r-o')
xlabel('k')
ylabel('E (%)')
title('k-NN Training and Validation Errors (k=1-30)')
legend('Validation Error','Training Error')
```



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

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

```
optimal_valid_error =

   12.5000


optimal_train_error =

   12.4444
```

```
% Given a validation error of 12.5% and a testing error of 12.1667%, the
% optimal k seems to be 30. Other permutations with smaller values of k
% (somewhere in the 5 to 15 range) have higher variability but for the
% larger k we observe more stable validation errors. The difference in the
% range and  at k = 30 is negligible (<3%) and so we should take a more
```

```
% stable number.
```

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

```
% k-NN optimizes for k, and so there will not necessarily be an improvement
% in rates of error with an increase in the size of k. Moreover, larger k's
% increase the number of considered neighbouring points, which means there
% is a reliance on the specific relationships and the more clusters
% considered the less dependent on a specific relationship the model is,
% which means it can be more readily generalized to non-training data.
```

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

```
incorrect_test = 0;

oknn = fitcknn(z_train(:,1:2), feats(:,3), 'NumNeighbors', 30);

pdt_oknn = predict(oknn, (z_test(:,1:2)));

for i=1:length(pdt_oknn)
    if pdt_oknn(i) ~= tfeats(i,3)
        incorrect_test = incorrect_test +1 ;
    end

end

error_optimal = (incorrect_test/length(tfeats)) * 100

% The error for k = 30 is 11.4286%
```

```
error_optimal =

   11.4286
```

(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 reflects our aformentioned arguements as the optimal value has
% generated a model that overfits less than 3.3 (17.4% vs 11.4%), which is
% even lower than the SVM model! As our most accurate, it is (by a close
% shave) the best performing model.
```