The goal of these tutorial and lab exercises is to become familiar with state of the art pattern recognition myoelectric control systems.

**Part1:**

**Step 1**: Simulate three different distributions in a 2-dimensional feature space. You can use the Clancy emg model from yesterday, or perform a trivial simulation as shown below.

%Generate 3 distributions... it doesn't really matter what they look like.

x\_1 = randn(1,500)+0;

y\_1 = randn(1,500)+0;

x\_2 = randn(1,500)+0;

y\_2 = randn(1,500)+3;

x\_3 = randn(1,500)+3;

y\_3 = randn(1,500)+0;

x = [x\_1 x\_2 x\_3];

y = [y\_1 y\_2 y\_3];

%Assign class labels for each distribution

classValues(1:500) = 1;

classValues(501:1000) = 2;

classValues(1001:1500) = 3;

%Plot Results and color code based on their class label

plot(x(1:500),y(1:500),'b.');

hold on;

plot(x(501:1000),y(501:1000),'r.');

plot(x(1001:1500),y(1001:1500),'k.');

**Step 2**: Train and visualize a linear classifier. I have provided code to implement a linear discriminant analysis classifier. The derivation for this classifier is shown in Appendix A if you are interested in the math. You should see that the distributions that I simulated above are pretty much ‘linearly separable’ meaning they can be separated with straight-lines (or hyperplanes when working in multi-dimensional feature space).

[Wg,Cg] = lda([x;y],classValues);

%Visualize the classifier using a mesh of testpoints that span the feature %space

[X,Y] = meshgrid(linspace(-10,10), linspace(-10,10));

X = X(:); Y = Y(:);

outputs = [X Y]\*Wg+repmat(Cg,length(X),1);

[dum predictedVals] = max(outputs');

ind\_C1 = find(predictedVals == 1);

ind\_C2 = find(predictedVals == 2);

ind\_C3 = find(predictedVals == 3);

%Plot the values

plot(X(ind\_C1),Y(ind\_C1),'mo');

plot(X(ind\_C2),Y(ind\_C2),'co');

plot(X(ind\_C3),Y(ind\_C3),'go');

plot(x(1:500),y(1:500),'b.');

plot(x(501:1000),y(501:1000),'r.');

plot(x(1001:1500),y(1001:1500),'k.');

**Step 3:** Train a non-linear classifier. We’ll use the neural network toolbox for this. There is a vast body of literature on neural networks. Rather than understanding the mathematics, we’ll focus on visualizing the type of boundaries we can make by implementing them. We’ll use a the neural network toolbox and implement a multi-layer perceptron neural network with 5 hidden layer nodes. The default settings provided by the tool-box should work just fine. We will have to manipulate our ‘classValues’ variable a little bit to make it play nice with the toolbox.

feats = [x;y];

meshVal = [X Y]';

numClasses = 3;

networkTargets = zeros(numClasses,length(classValues));

for i = 1:numClasses

 indOn = find(classValues==i);

 networkTargets(i,indOn)=1;

end

When you get to the ‘Deployment’ step in the Neural Network wizard, you should generate a Matlab ‘Matrix Only Function’ and save your file to something intuitive. For example, I saved my network as hargroveNet1.m

outputs = hargroveNet1(meshVal);

[dum predictedVals] = max(outputs);

ind\_C1 = find(predictedVals == 1);

ind\_C2 = find(predictedVals == 2);

ind\_C3 = find(predictedVals == 3);

%Plot the values

plot(X(ind\_C1),Y(ind\_C1),'mo'); hold on;

plot(X(ind\_C2),Y(ind\_C2),'co');

plot(X(ind\_C3),Y(ind\_C3),'go');

plot(x(1:500),y(1:500),'b.');

plot(x(501:1000),y(501:1000),'r.');

plot(x(1001:1500),y(1001:1500),'k.');

**Step 5:** Repeat steps 1-4 with distributions that aren’t linearly separable. For example.

x\_11 = randn(1,250)+0;

y\_11 = randn(1,250)+0;

x\_12 = randn(1,250)+6;

y\_12 = randn(1,250)+0;

x\_2 = randn(1,500)+0;

y\_2 = randn(1,500)+3;

x\_3 = randn(1,500)+3;

y\_3 = randn(1,500)+0;

x = [x\_11 x\_12 x\_2 x\_3];

y = [y\_11 y\_12 y\_2 y\_3];

**Part 2**

In this part, we will work with two sets of real EMG signals collected from able bodies control subjects. The first set is called the Ninapro dataset which is freely available online. You can download a description of the data collection procedure at: <http://www.idiap.ch/project/ninapro/publications> . The Biorob 2012 pdf explains the data collection. I have preprocessed and organized this data set for you to only consider movements that we commonly use to control prosthetic limbs.

Last week, I also replicated this data collection procedure at RIC and collected some addition raw EMG signals at the Rehabilitation Institute of Chicago and organize the data for you. You can also download some useful functions from Adrian Chan’s webpage: <http://www.sce.carleton.ca/faculty/chan/index.php?page=matlab>

You’ll want to download the getmavfeat.m getarfeat.m getwlfeat getsscfeat.m and getzcfeat.m files. There are lots of other interesting resources here that you might find useful. If you ever end up using these codesnips in the future, please acknowledge Dr. Chan, or write your own using the equations provided in the appendix.

Okay, now that you have all the files downloaded, you are ready to make a state-of-the-art myoelectric control system.

**Step 1**: Load in one of the data files. I’d suggest starting with AB043 from the RIC dataset.

load('pathname\AB043.mat')

The emg variable contains 8 channels of raw emg data sampled at 1000 Hz and the stimulus variable is a vector that provides a label of what the person was attempting during each movements. (0 = resting, 1 = wrist flexion, 2 = wrist extension, 3 = wrist pronation, 4 = wrist supination, 5 = radial deviation, 6 = ulnar deviation, 7 = hand open, 8 = cylindrical grasp, 9 = key grasp, 10 = fine pinch grasp).

Plot some of the raw signals and have a look at the data. You’ll see ten repetitions of each movements with some resting between each contraction.

**Step 2**: Consider just 2 channels (try channels 2 and 7) and 4 movements (movement numbers 1,2, 7 and 10). Extract the amplitude features from these channels and then plot the feature space representation. Try using a window length of 50 samples with a 50 sample increment in the feature extraction function.

feats = getmavfeat(emg(:,[2 7]),50,50);

targets = restimulus(50:50:end);

formattedData = [];

formattedTargets = [];

classVec = [1:2 7 10];

colorInd{1} = 'b.';

colorInd{2} = 'r.';

colorInd{7} = 'k.';

colorInd{10} = 'm.';

figure;

hold on;

for i = classVec

 indOn = find(targets==i);

 plot(feats(indOn,1),feats(indOn,2),colorInd{i});

end

**Step 3**: Repeat with a window lengths of 250, and 500. What do you notice about the resulting feature space.

**Step 4**: Add another channel (channel 5 for example) and create a 3 dimensional plot using a 250 ms window length. Use the orientation tool on the plotter to see how the 3-rd dimension helps to separate the feature space.

**Step 5:** Use all channels and then try using dimensionality reduction to visualize the data. Go back to Dr. Chan’s website and download the ULDA feature reduction. Alternatively, you can use principle components analysis.

feats = getmavfeat(emg(:,[1:8]),250,50);

targets = restimulus(250:50:end);

formattedData = [];

formattedTargets = [];

classVec = [1:2 7 10];

colorInd{1} = 'b.';

colorInd{2} = 'r.';

colorInd{7} = 'k.';

colorInd{10} = 'm.';

figure;

hold on;

subClassFeats = [];

subClassTargs = [];

counter = 1;

for i = classVec

 indOn = find(targets==i);

 subClassFeats = [subClassFeats;feats(indOn,:)];

 subClassTargs = [subClassTargs ones(1,length(indOn))\*counter];

 counter = counter+1;

end

[feature\_train\_ulda,feature\_test\_ulda] = ulda\_feature\_reduction(subClassFeats,3,subClassTargs',subClassFeats);

for i = 1:counter-1

 indOn = find(subClassTargs==i);

 plot3(feature\_train\_ulda(indOn,1),feature\_train\_ulda(indOn,2),feature\_train\_ulda(indOn,3),colorInd{i})

end

**Step 6:** Use the other feature extraction files to create additional features (6th order AR model, waveformlength, slope sign changes, and zero crossing). Use dimensionality reduction to view the resulting classes.

**Step 7:** Add the remaining classes (movements) to the dataset.

**Step 8:** Instead of visualizing the data, train a LDA classifier or a neural network like you did in Part 1. Use half the data (contractions 1-5 of each movement) to train the classifier and test the classifier with the remaining contractions.

**Step 9:** Train the classifier for all the other subjects in the RIC database and compute the average classification errors across subjects.

**Step 10**: Do the same thing for the Ninapro database. Compare the results. Experiment with varying feature sets, number of movements, and classifiers.

# Appendix A – TDAR Features and LDA Classification

The TD statistics proposed by Hudgins *et al.* [20] consisted of the MAV, the number of zero crossings, the number of slope sign changes, and the waveform length. These four statistics were computed for each MES input channel and are concatenated to form a feature vector.

* **Mean Absolute Value:** An estimate of the mean absolute value of the signal,, in analysis window  with  samples is given by

 , 

where  is the  sample in analysis window , and  is the total number of analysis windows over the entire sampled signal.

* **Number of Zero Crossings:** A simple frequency measure can be obtained by counting the number of times the waveformcrosses zero. To reduce the number of noise induced zero crossings, a threshold value of  is included in the calculation. Given two consecutive values of the signal,and , the zero crossing count is incremented if



The value of  is dependent on the system noise and needs to be selected appropriately.

* **Slope Sign Changes**: Another feature which may provide a measure of frequency content is the number of times the slope of the waveform changes sign. Once again a threshold value must be used to reduce noise induced slope sign changes. Given three consecutive values of the signal,, and  the slope sign change count is incremented if



and



The value of  is once again dependent on the noise and should be selected appropriately. The value of the  can be selected by examining the noise levels of the data.

* **Waveform Length:** This feature provides information on the complexity of the waveform in each analysis window. It is the cumulative length of the waveform defined as



where,, is the difference between consecutive signal samples. The resultant value provides a measure of waveform amplitude, frequency, and duration all within a single parameter.

In a linear autoregressive model of order, a time series  is modeled as a linear combination of  earlier values in the time series, with the addition of a correction term 



The coefficients, for, are found by minimizing the mean squared error between  and. The RMS value of the signal was determined using



where is the output for each analysis window,  is the measured signal, and  is the length of the analysis window. Previous work has shown that a 6th order AR model plus the RMS signal value performs best, resulting in seven features per channel.

A LDA classifier is a simplified implementation of a Bayesian statistical classifier. The Bayes classification rule states: assign the -length pattern  to the class, so that the following inequality is satisfied

 for all 

However, these *a posteriori* probabilities cannot be directly measured. Instead, Bayes’ Theorem



can be used to provide the solution by deriving the *a posteriori* probabilities from estimates of the *a priori* probabilities



where is the probability density function for the samples within the  class and  is the probability density function of the input space and is a constant over all the classes. It is usually assumed that the probabilities of the output classes, are equal. Now application of Bayes’ classification rule essentially becomes evaluating = for each of the classes and choosing the maximum value.

The LDA implementation simplifies the Bayesian classifier by assuming that all probability density functions are Gaussian. The multivariate Gaussian probability density function for  classes of patterns can be expressed as



where  is thelength mean vector for the  class and is the  covariance matrix for the  class. Now,



Expressing this value in the natural logarithm form and canceling constant terms yields



Furthermore, by assuming that all the covariance matrices are equal, the set of discriminant functions becomes



It should be noted that the discriminant functions are not equivalent to the reduced LDA subspace associated with the feature reduction described in Section 3.1. The classification now becomes a problem of  equations and  unknowns, where  is the number of classes and  is the length of the feature vector.  could also be expressed in terms of weight matrix and a offset array







Clearly, after the weights and the offset have been calculated from an appropriate set of training data, feed-forward classification using a LDA is computationally simple.