BIOMEDE 517 - Neural Engineering - Labs 7, 8, and 9 - Dr. Cindy Chestek

Appendix

Kushal Jaligama

Table 1 - Naïve Bayes Classification Equations and Results

| # | Equation Name | Equation |
|---|---|---|
| 1 | Sum of Logarithmic Probabilities for Feature-Class Pairs | $\sum_{j=1}^{D} \log \hat{g}_{k=1}^{(j)}(X_i)$ |
| # | Result | Value |
| 2 | Classification Accuracy of Test Data using Training Data Mean | 97.39% |
| 3 | Classification Accuracy of Spoofed Data using Full Dataset Mean | 99.66% |

Table 2 - Linear Discriminant Analysis Equations and Results

| # | Equation Name | Equation |
|---|---|---|
| 1 | Objective Function for Linear Discriminant Analysis | $\hat{f}(x) = \arg\max_k[\hat{\pi}_k * \phi(x; \hat{\mu}_k, \hat{\Sigma})]$ |
| # | Result | Value |
| 2 | Classification Accuracy with Leave-One-Out Cross Validation | 71.79% |

Figure 1 - Confusion Matrix for Linear Discriminant Analysis (LDA)

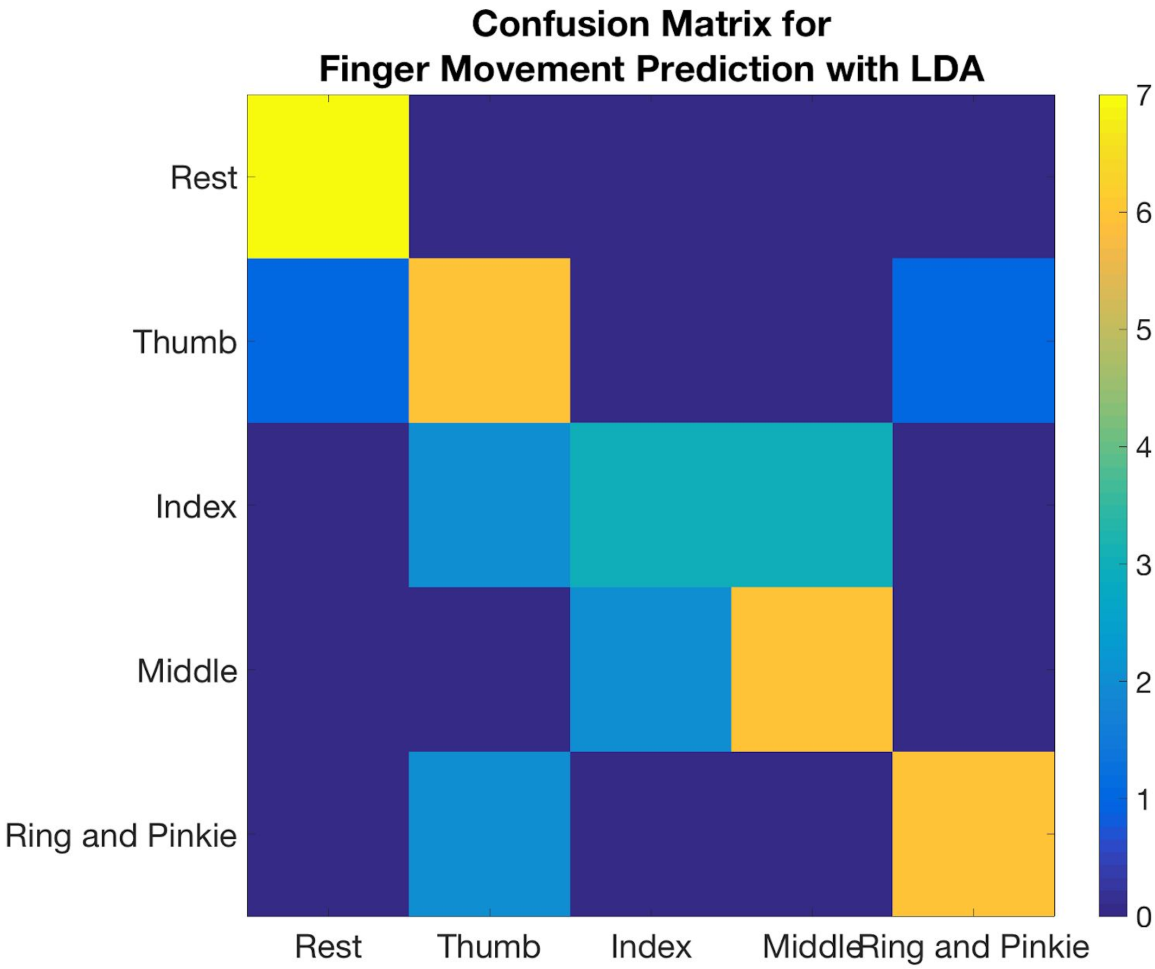
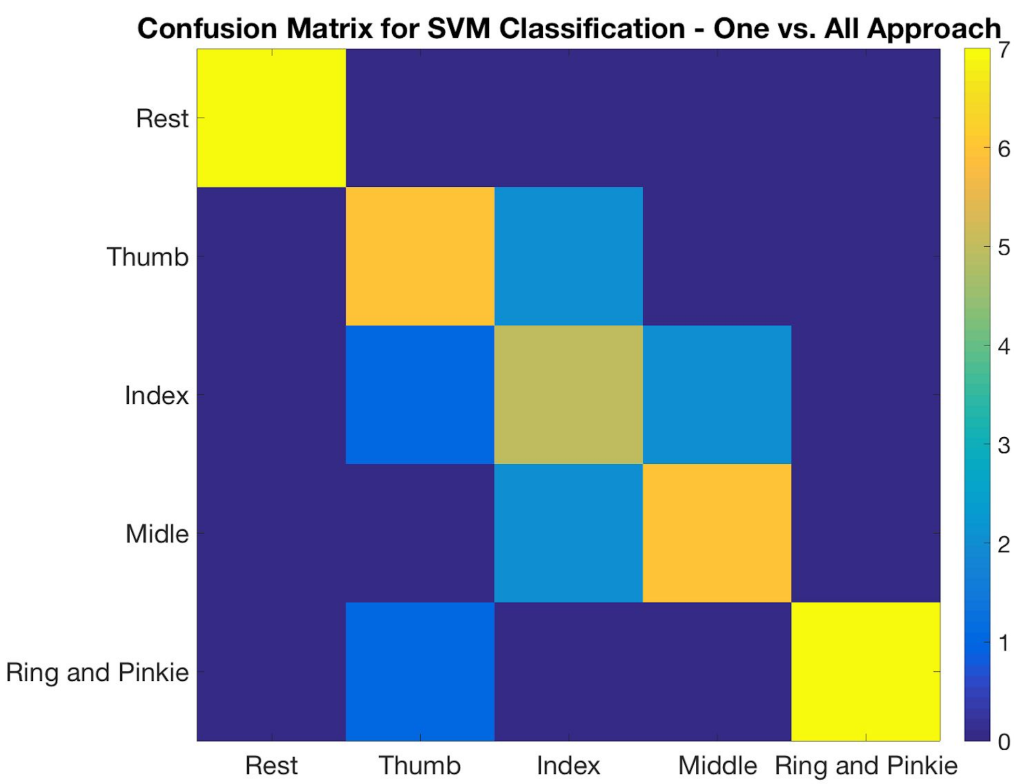


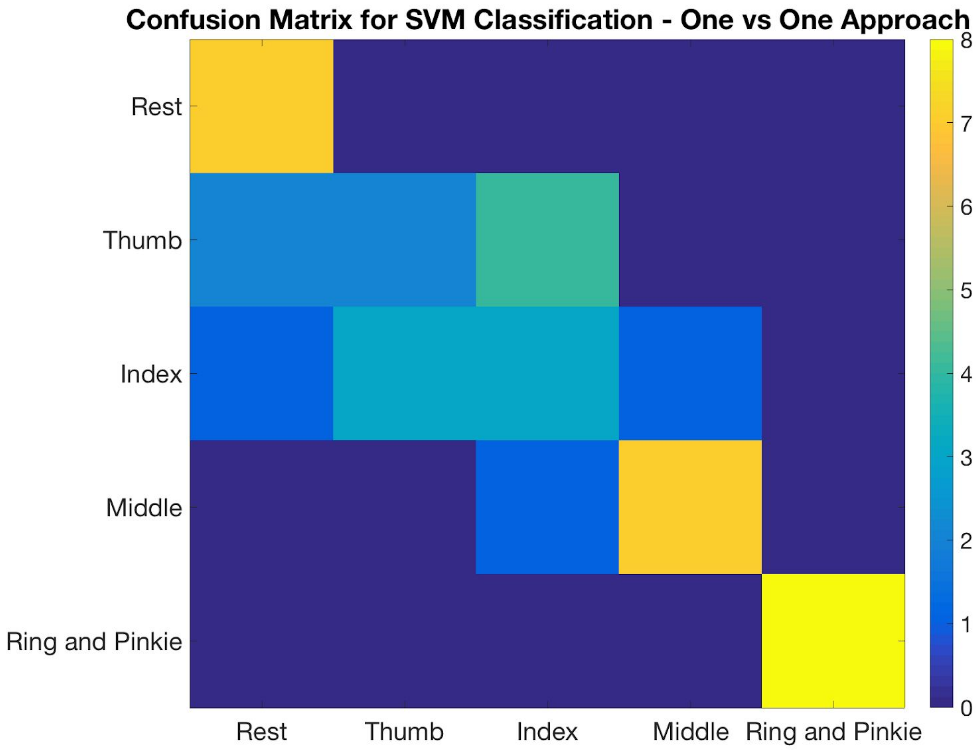

Table 3 - Support Vector Machines Equations and Results

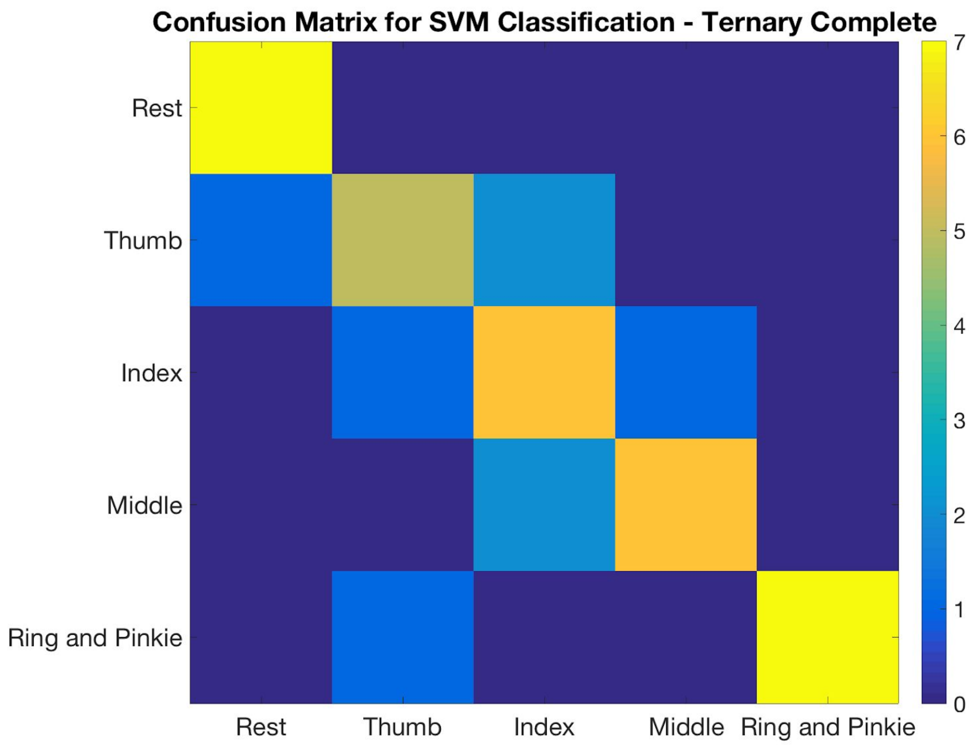
| # | Equation Name | Equation |
|---|---|---|
| 1 | Restructuring of Multiple Classes to Binary Labels | $Y_{i,j} = \begin{cases} 1, & if\ observation\ i\ belongs\ to\ group\ j \\ 0, & if\ observation\ i\ does\ not\ belong\ to\ group\ j \end{cases}$ |
| # | SVM Configuration | Accuracy |
| 2 | Manually Separated Binary Labels - Linear SVM | By Class Accuracy {class:accuracy} = {1:100%}, {2:79.49%}, {3:76.92%}, {4:89.74%}, {5:9744%}; Overall Accuracy = 88.72% |

| | | |
|---|---|---|
| 3 | Manually Separated Binary Labels - Non-Linear SVM | By Class Accuracy {class:accuracy} = {1:89.74%}, {2:79.49%}, {3:79.49%}, {4:87.18%}, {5:94.87%}; Overall Accuracy = 85.15% |
| 4 | Multiclass SVM Model - One vs. All | Accuracy = 69.23% |
| 5 | Multiclass SVM Model - One vs. One | Accuracy = 79.49% |
| 6 | Multiclass SVM Model - Ternary Complete | Accuracy = 79.49% |
| 7 | Multiclass SVM Model - Ordinal | Accuracy = 79.49% |

Figure 2 - Confusion Matrices for Multiclass Support Vector Machines




Confusion Matrix for SVM Classification - One vs One Approach
Rest
Thumb
Index
Middle
Ring and Pinkie
Rest
Thumb
Index
Middle
Ring and Pinkie
0
1
2
3
4
5
6
7
8


Confusion Matrix for SVM Classification - Ternary Complete
Rest
Thumb
Index
Middle
Ring and Pinkie
Rest
Thumb
Index
Middle
Ring and Pinkie
7
6
5
4
3
2
1
0

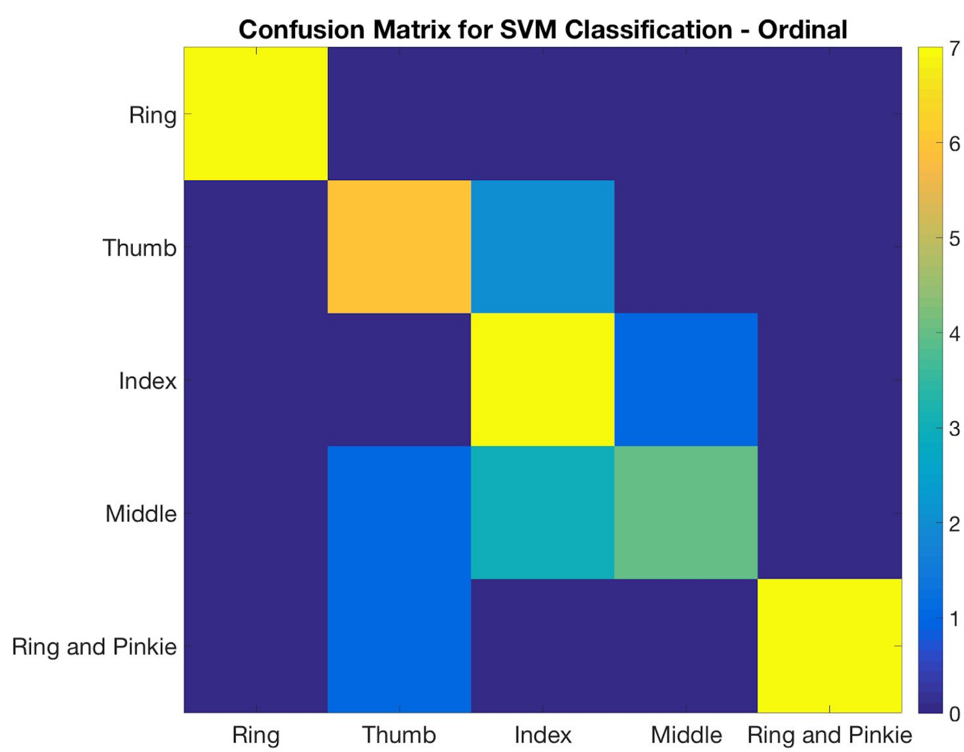

Confusion Matrix for SVM Classification - Ordinal
Ring
Thumb
Index
Middle
Ring and Pinkie
Ring
Thumb
Index
Middle
Ring and Pinkie
7
6
5
4
3
2
1
0

Table 4 - Linear Regression Equations and Results

| # | Equation Name | Equation |
|---|---|---|
| 1 | Objective Function | $\min_{\vec{w},b}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{w}^T\vec{x}_i + b - y_i)^2\right]$ |
| 2 | Decoder Matrix | $B = (transpose(y)y)^{-1} transpose(y)x$ |
| **#** | **Result Name** | **Value** |
| 3 | Sum of Mean Squared Error of Predictions on Test Data | 5.8726e+03 |
| 4 | Correlation Coefficients of Prediction to Actual Motion | 0.9467, 0.9193, 0.8762, 0.8450<br>Position X, Position Y, Velocity X, Velocity Y |

Table 5 - Ridge Regression Equations and Results

| # | Equation Name | Equation |
|---|---|---|
| 1 | Objective Function | $\min_{\vec{w},b}\left[\frac{1}{n}\sum_{i=1}^{n}(\vec{w}^T\vec{x}_i + b - y_i)^2 + \lambda\|\|\vec{w}\|\|_2^2\right]$ |
| 2 | Decoder Matrix | $B = (transpose(y)y \ + \ n\ lambda\ I)^{-1} transpose(y)\ x$ |
| **#** | **Result Name** | **Value** |
| 3 | Optimal Lambda | 0.0286 |
| 4 | Sum of Mean Squared Error of Predictions on Test Data | 5.8390e+03 |
| 5 | Correlation Coefficients of Prediction to Actual Motion | 0.9469,  0.9194, 0.8766, 0.8461<br>Position X, Position Y, Velocity X, Velocity Y |

Figure 3 - Sum of Mean Squared Errors versus Lambda

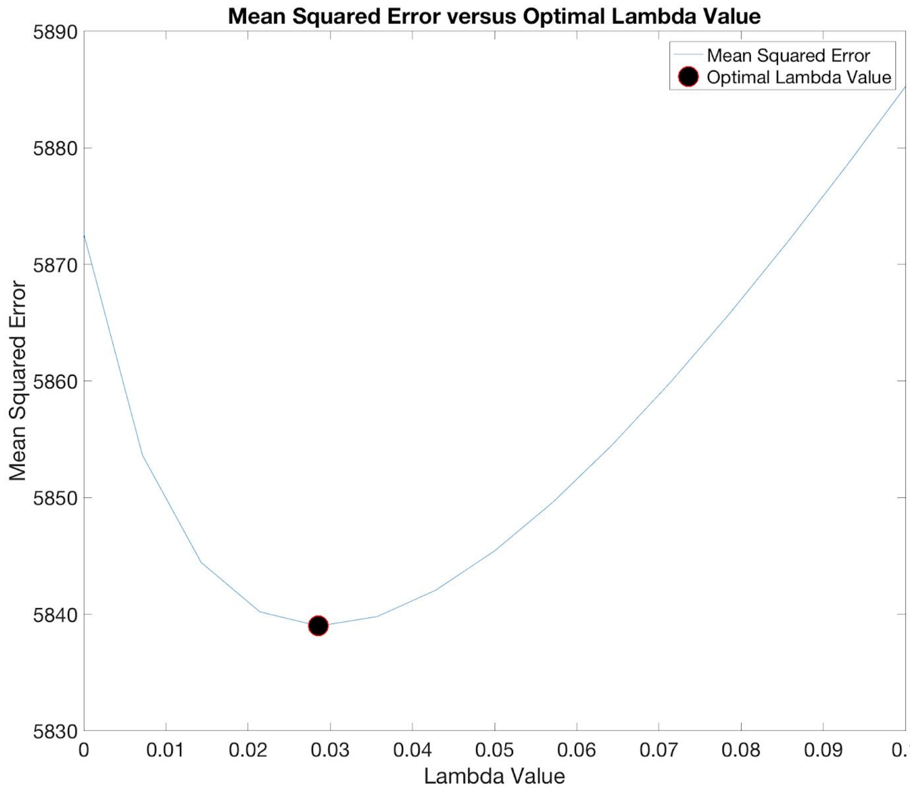



Table 6 - LASSO Results

| # | Result Name | Value |
|---|---|---|
| 1 | Optimal Lambda | 0.0286 |
| 2 | Sum of Mean Squared Error of Predictions on Test Data | 6.1863e+03 |
| 3 | Correlation Coefficients of Prediction to Actual Motion | 0.9475, 0.9204, 0.8769, 0.8459<br>Position X, Position Y, Velocity X, Velocity Y |

Table 7 - Kalman Filter Results

| # | Result Name | Value |
|---|---|---|
| 1 | Sum of Mean Squared Error of Predictions on Test Data | 1.0170e+07 |

| | | |
|---|---|---|
| 2 | Correlation Coefficients of Prediction to Actual Motion | 0.8955, 0.8928, 0.7893, 0.8031<br>Position X, Position Y, Velocity X, Velocity Y |

<u>Code Utilized for This Lab Report</u>

```
% BIOMEDE 517 - Neural Engineering
% Lab 7 Part 1 - Naïve Bayes Classifier with Poisson Distribution
% Kushal Jaligama

% Predicting reach direction of monkey using 95 neurons

clearvars
close all

% Load data from firingrate.mat
% 95 neurons, 8 reach dirs, 182 samples for each neuron-dir comobo
load('firingrate.mat')

training_samples = 91;
total_samples = 182;
test_samples = total_samples - training_samples;
% Split data in half, training and testing test data sets
% Since we have 182 samples, use training_samples samples for each half
training_data = firingrate(:, 1:training_samples, :);
test_data = firingrate(:, training_samples+1:total_samples, :);

% Training Step - Parameters for Poisson Distribution
% Calculate lambda indexed by neuron, target
lambda = zeros(95, 8); % n x m matrix
% mean firing rate of neuron n when reaching to direction m
for dir = 1:8
  for j = 1:95
    lambda(j, dir) = mean(training_data(j, :, dir));
  end
end

% Prediction Step - Testing the Classifier
% Loop through all individual trials of test data
```

```
predictions = zeros(test_samples, 8); % This is to assign the features

for class = 1:8
   for i = 1:test_samples
      % Get the feature vector (95 neurons, i'th sample, class)
      X_i = test_data(:, i, class);
      for k = 1:8
         % Calculate log prob density for the feature vector
         g_hat = poisspdf(X_i, lambda(:,k));
         % We will assign sample based on sum of probabilities
         arg_k(k) = sum(log(g_hat));
      end
      % Assign feature to maximal k class
      % (best direction for this set of neuron firing rates)
      [maximal_k, index] = max(arg_k);
      predictions(i, class) = index;
   end
end

% Calculate accuracy of class assignments
number_correct = 0;
number_total = 0;
for class = 1:8
   for i = 1:test_samples
      if predictions(i, class) == class
         number_correct = number_correct + 1;
      end
      number_total = number_total + 1;
   end
end
accuracy = number_correct / number_total

% Applying classifier to spoofed dataset
% Calculate lambda indexed by neuron, target for full dataset
lambda_full = zeros(95, 8); % n x m matrix
for dir = 1:8
   for j = 1:95
      lambda_full(j, dir) = mean(firingrate(j, :, dir));
   end
```

```
end

% Generate random data along a Poisson Distribution
spoof_data = zeros(95, total_samples, 8);
for i = 1:total_samples
   spoof_data(:,i,:) = poissrnd(lambda_full);
end

% Prediction Step - Testing the Classifier
% Loop through all individual trials of test data
predictions_full = zeros(total_samples, 8); % This is to assign the features

for class = 1:8
   for i = 1:total_samples % samples
      % Get the feature vector
      X_i_LDA = spoof_data(:, i, class);
      for k = 1:8
         % Calculate log prob density for the feature vector
         g_hat_LDA = poisspdf(X_i_LDA, lambda_full(:,k));
         % We will assign sample based on sum of probabilities
         arg_k_LDA(k) = sum(log(g_hat_LDA));
      end
      % Assign feature to maximal k class
      % (best direction for this set of neuron firing rates)
      [maximal_k, index] = max(arg_k_LDA);
      predictions_full(i, class) = index;
   end
end

% Calculate accuracy of class assignments
number_correct = 0;
number_total = 0;
for class = 1:8
   for i = 1:total_samples
      if predictions_full(i, class) == class
         number_correct = number_correct + 1;
      end
      number_total = number_total + 1;
   end
```

```
end
accuracy_full = number_correct / number_total

% BIOMEDE 517 - Neural Engineering
% Lab 7 Part 2 - Linear Discriminant Analysis
% Kushal Jaligama

clearvars
close all

% Load data from ecogclassifydata.mat
load('ecogclassifydata.mat')
% Average power values from 0.5s prior to movement to 1.5s after
% movement in 60-120 Hz band
numTrials = 39;
% 27 electrode pairs, 'group' indicates which finger
% 1 is rest, 2 is thumb, 3 is index, 4 is middle
% 5 is ring and pinkie.

% Make LDA prediction of test sample's class
% Iterate through data and give each sample a turn at being test data
% Rest of samples should be training
for i = 1:numTrials
   test = powervals(i, :);
   training = powervals(1:i, :);
   training(end:numTrials - 1, :) = powervals(i + 1:end, :);
   train_groups = group(1:i, :);
   train_groups(end:numTrials - 1, :) = group(i + 1:end, :);
   predictions(i) = classify(test, training, train_groups, 'linear');
end

predictions = transpose(predictions);
num_correct = 0;

for i = 1:numTrials
   if predictions(i) == group(i)
      num_correct = num_correct + 1;
   end
end
```

```
accuracy = num_correct / numTrials

% This tells you what points were classified correctly and not
conf = confusionmat(group, predictions);
% A value of 1 at (2,1) in conf means one value that was supposed
% to be in class 2 was misclassified to group 1

% Plot this for visualization
imagesc(conf)

% Calculate percent correct by class
for i = 1:5
   total_in_class = sum(conf(i,:));
   percent_correct(i) = conf(i, i) / total_in_class;
end

percent_correct

% BIOMEDE 517 - Neural Engineering
% Lab 7 Part 3 - Support-Vector Machine
% Kushal Jaligama

clearvars
close all

% Load ECoG data
load('ecogclassifydata.mat');

numTrials = 39;
numGroups = 5;

% Restructure group to establish binary labels
Y = zeros(numTrials, numGroups);
for i = 1:numTrials
   for j = 1:numGroups
      if group(i) == j
         Y(i, j) = 1;
      end
```

```
    end
end

% Make predictions on the model with an SVM using leave-one-out
% cross validation
for i = 1:numGroups
    y = Y(:, i);
    SVMmodel = fitcsvm(powervals, y, 'KernelFunction', 'linear', 'Leaveout', 'on');
    predictions(:, i) = kfoldPredict(SVMmodel);
end

% Establish total number of correct predictions for accuracy testing
num_correct = 0;
% Establish number of correct preds in each class
by_class = zeros(1, 5);
for j = 1:numGroups
    for i = 1:numTrials
        if predictions(i, j) == Y(i, j)
            num_correct = num_correct + 1;
            by_class(j) = by_class(j) + 1;
        end
    end
    % Calculate the accuracy for each class
    by_class_accuracy(j) = by_class(j) ./ numTrials;
end

overall_accuracy = num_correct / (numTrials * numGroups)

by_class_accuracy

% Non-linear SVM with radial basis kernel function
for i = 1:numGroups
    y = Y(:, i);
    SVMmodel = fitcsvm(powervals, y, 'KernelFunction', 'rbf', 'Leaveout', 'on');
    nonlin_predic(:,i) = kfoldPredict(SVMmodel);
end

% Calculate overall prediction accuracy and by class
nonlin_num_correct = 0;
```

```
nonlin_by_class = zeros(1, 5);
for i = 1:numGroups
    for j = 1:numTrials
        if nonlin_predic(j, i) == Y(j, i)
            nonlin_num_correct = nonlin_num_correct + 1;
            nonlin_by_class(i) = nonlin_by_class(i) + 1;
        end
    end
    nonlin_by_class_accuracy(i) = nonlin_by_class(i) ./ numTrials;
end

nonlin_overall_accuracy = nonlin_num_correct / (numTrials * numGroups)

nonlin_by_class_accuracy

% BIOMEDE 517 - Neural Engineering
% Lab 7 Part 4 - Multi-Class Support-Vector Machine
% Kushal Jaligama

clearvars
close all

% Load ECoG data
load('ecogclassifydata.mat');

numTrials = 39;
numGroups = 5;

% Create a multi-class SVM model that can use int v
SVMmodel = fitcecoc(powervals, group, 'Leaveout', 'on', 'Coding', 'onevsall');
predictions = kfoldPredict(SVMmodel);

num_correct = 0;
for i = 1:numTrials
    if predictions(i) == group(i)
        num_correct = num_correct + 1;
    end
end
```

```
disp('onevsall')
accuracy = num_correct / numTrials
conf1 = confusionmat(group, predictions);

% onevsone - force each SVM to be tested against all non-target
% classes separately in the model
SVMmodel = fitcecoc(powervals, group, 'Leaveout', 'on', 'Coding', 'onevsone');
predictions = kfoldPredict(SVMmodel);

num_correct = 0;
for i = 1:numTrials
   if predictions(i) == group(i)
      num_correct = num_correct + 1;
   end
end

disp('onevsone')
accuracy = num_correct / numTrials
conf2 = confusionmat(group, predictions);

% ternarycomplete - partitions n classes into positive, negative,
% and zero valued classes that are cycled during training
SVMmodel = fitcecoc(powervals, group, 'Leaveout', 'on', 'Coding', 'ternarycomplete');
predictions = kfoldPredict(SVMmodel);

num_correct = 0;
for i = 1:numTrials
   if predictions(i) == group(i)
      num_correct = num_correct + 1;
   end
end

disp('ternarycomplete')
accuracy = num_correct / numTrials
conf3 = confusionmat(group, predictions);

% ordinal - uses n-1 binary SVMs for n classes. sampling space is
% partitioned at each class threshold
SVMmodel = fitcecoc(powervals, group, 'Leaveout', 'on', 'Coding', 'ordinal');
```

```
predictions = kfoldPredict(SVMmodel);

num_correct = 0;
for i = 1:numTrials
   if predictions(i) == group(i)
      num_correct = num_correct + 1;
   end
end

disp('ordinal')
accuracy = num_correct / numTrials
conf4 = confusionmat(group, predictions);

figure(1)
imagesc(conf1)
figure(2)
imagesc(conf2)
figure(3)
imagesc(conf3)
figure(4)
imagesc(conf4)

% BIOMEDE 517 - Neural Engineering
% Lab 8 All Parts - Continuous Decoders
% Kushal Jaligama

% Real-time decoding algorithms starting with linear regression
% Using data from a reach task

clearvars
close all

% Part 0 - Process Data
load('contdata.mat')
% Columns of X contain X position, Y position, X velocity, Y velocity
% Columns of Y contain firing rates of 950 recorded units
numObservations = 31413; % Number of time points

% Split data into training and test 50/50
```

```
training_rows = floor(numObservations / 2);
training_x = X(1:training_rows, :); % Position X, Y, Velocity X, Y
training_y = Y(1:training_rows, :); % Firing Rates of 950 units

test_rows = ceil(numObservations / 2);
test_x = X(test_rows:end, :);
test_y = Y(test_rows:end, :);

% Add a column of ones to the neural data to calculate an intercept
% term in the regression models
training_y = [ones(training_rows, 1) training_y];
test_y = [ones(test_rows, 1) test_y];

% Part 1 - Linear Regression

% Solve a linear regression equation with training data
% x = yB => B is the linear decoder matrix we want to find
training_y_t = transpose(training_y);
B = inv(training_y_t * training_y) * (training_y_t * training_x);

% To predict new data, multiply it by your linear decoder matrix, B
linear_predictions = test_y * B;

% Measure the mean squared error of predictions on the test data
linear_mean_squared_error = sum(mean((linear_predictions - test_x).^2))
% Correlation of predictions to actual motion
for i = 1:4
   linear_corr_coeffs(i) = corr2(test_x(:, i), linear_predictions(:, i));
end
linear_corr_coeffs

% Part 2 - Ridge Regression
least_error = intmax;
lambda_vals = linspace(0, 0.1, 15);
ridge_errors = zeros(1, 15);
% Perform ridge regression on all of the lambda values
for i = 1:15
   lambda = lambda_vals(i);
   square = training_y_t * training_y;
```

```
   B_ridge = inv(square + training_rows * lambda * eye(size(square))) * training_y_t *
training_x;
   prediction_ridge = test_y * B_ridge;
   % Calculate the mean squared errors of predictions on test data
   ridge_errors(i) = sum(mean((test_x - prediction_ridge).^2));
   % Use error terms to find the best lambda value
   if ridge_errors(i) <= least_error
      optimal_lambda = lambda;
      least_error = ridge_errors(i);
   end
end

optimal_lambda
least_error

plot(lambda_vals, ridge_errors)
hold on
plot(optimal_lambda, least_error, 'ro')

% Use the best lambda value to get best prediction
square = (training_y_t * training_y);
B_ridge = inv(square + training_rows * optimal_lambda * eye(size(square))) * training_y_t *
training_x;
best_ridge_predictions = test_y * B_ridge;
best_ridge_mean_squared_error = sum(mean((test_x - best_ridge_predictions).^2))
% Correlation of predictions to actual motion
for i = 1:4
   ridge_corr_coeffs(i) = corr2(test_x(:, i), best_ridge_predictions(:, i));
end
ridge_corr_coeffs

% BIOMEDE 517 - Neural Engineering
% Lab 9 Part 1 - LASSO
% Kushal Jaligama

close all

% Part 0 - Process Data
load('contdata.mat')
```

```
% Columns of X contain X position, Y position, X velocity, Y velocity
% Columns of Y contain firing rates of 950 recorded units
numObservations = 31413; % Number of time points

% Split data into training and test 50/50
training_rows = floor(numObservations / 2);
training_x = X(1:training_rows, :); % Position X, Y, Velocity X, Y
training_y = Y(1:training_rows, :); % Firing Rates of 950 units

test_rows = ceil(numObservations / 2);
test_x = X(test_rows:end, :);
test_y = Y(test_rows:end, :);

% Add a column of ones to the neural data to calculate an intercept
% term in the regression models
training_y = [ones(training_rows, 1) training_y];
test_y = [ones(test_rows, 1) test_y];


% LASSO

optimal_lambda = 0.0286

% Set up the B matrix
B = zeros(951, 4);
for i = 1:4
  % Get the weights for each recorded unit (neurons)
  B(:, i) = lasso(training_y, training_x(:,i), 'Lambda', optimal_lambda);
end

lasso_predictions = test_y * B;

% Now calculate the mean squared error and correlation coefficients
% Measure the mean squared error of predictions on the test data
lasso_mean_squared_error = sum(mean((lasso_predictions - test_x).^2))
% Correlation of predictions to actual motion
for i = 1:4
  lasso_corr_coeffs(i) = corr2(test_x(:, i), lasso_predictions(:, i));
end
```

```
lasso_corr_coeffs

% BIOMEDE 517 - Neural Engineering
% Lab 9 All Parts - LASSO and Kalman Filters
% Kushal Jaligama

clearvars
close all

% Part 0 - Process Data
load('contdata.mat')
% Columns of X contain X position, Y position, X velocity, Y velocity
% Columns of Y contain firing rates of 950 recorded units
numObservations = 31413; % Number of time points

% Split data into training and test 50/50
training_rows = floor(numObservations / 2);
training_x = X(1:training_rows, :); % Position X, Y, Velocity X, Y
training_y = Y(1:training_rows, :); % Firing Rates of 950 units

test_rows = ceil(numObservations / 2);
test_x = X(test_rows:end, :);
test_y = Y(test_rows:end, :);

% Add a column of ones to the neural data to calculate an intercept
% term in the regression models
training_y = [ones(training_rows, 1) training_y];
test_y = [ones(test_rows, 1) test_y];

% Kalman Filter

% The physics is represented as:
% x_t is a state ("position, velocity")
% x_t = Ax_(t-1) + w_t
%       physics    noise

% y_t is neural firing rates at time step
% y_t = C_x(t) + q_t
%       LinFilt  noise
```

```
% Transpose the matrices to put time domain as columns
training_x = transpose(training_x);
training_y = transpose(training_y);
test_x = transpose(test_x);
test_y = transpose(test_y);

% Set up the time step variables
training_x_prev = training_x;
% Remove first element of training data (ones)
training_x(:, 1) = [];
training_y(:, 1) = [];
% Remove last element of "prev" matrix for prediction purposes
training_x_prev(:, training_rows) = [];

% Get the physics and linear filter matrices
C = (training_y * transpose(training_x)) / (training_x * transpose(training_x));
A = (training_x * transpose(training_x_prev)) / (training_x_prev* transpose(training_x_prev));

% Get the noise
W = (1 / (numObservations - 1)) * (training_x - A * training_x_prev) * transpose(training_x - A *
training_x_prev);
Q = (1 / (numObservations - 1)) * (training_y - C * training_x) * transpose(training_y - C *
training_x);

% Start making predictions
xhat = zeros(4, test_rows);
xhat(:, 1) = test_x(:, 1);

% a posterior covariance of x, how accuracte is the estimate
postcov = W;

% Predict, innovate, update
for i = 2:test_rows
   % Prediction given last time step
   xhatprev = A * test_x(:, i - 1);
   postcovprev = A * postcov * transpose(A) + W;
   Kt = postcovprev * transpose(C) / (C * postcovprev * transpose(C) + Q);
   xhat(:, i) = xhatprev + Kt * (test_y(:, i) - C * xhat(:, i-1));
```

```
   postcov = (eye(4) - Kt * C) * postcovprev;
end

% Now calculate the accuracy of this entire thing
kalman_mean_squared_error = sum(mean(test_x - xhat).^2)
% Correlation of predictions to actual motion
for i = 1:4
   kalman_corr_coeffs(i) = corr2(test_x(i, :), xhat(i, :));
end
kalman_corr_coeffs
```