Neuro 530 Assignment – Linear regression and classification

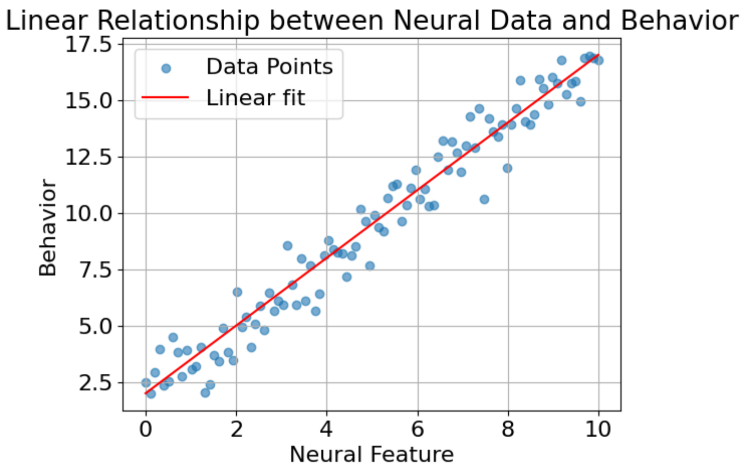
In this assignment, you will analyze neural recordings from a monkey performing arm-reaching tasks. Your goal is, given those recordings, to train linear models capable of predicting continuous behavior and classifying between a fixed set of states.

Theoretical Background

Linear Regression

When attempting to decode **continuous** behavior (2D movement of the arm, movement of fingers, walking, etc.) from neural activity, the simplest possibility is assuming that the relationship between them is linear, meaning that we believe we can reconstruct behavior by just adding and scaling the different neural channels. This is, actually, not the worst assumption, as we can get decent performance out of a linear decoder.

The simplest linear decoder is a linear regression, in which we try to **fit a line to the data** so that the error between predictions and behavior is as small as possible. If we have a single channel of neural activity, and we are trying to predict a single behavior variable (such as the X position of the arm in space), the solution would look something like this:



From high-school math, we know we can describe this fit with an equation. If X represents the neural feature (for example, threshold crossings for every 50ms of time), and Y represents the behavior, then an equation that represents the line would be:

$$Y = Xm + b$$

Where *m* represents the slope of the line, and *b* represents the "bias" or how much the line should be displaced vertically. If we have at least two data points, we can calculate how much the slope and bias need to be pretty easily. This is called the "training" of a linear model.

We can extend this to when we have multiple neural channels, or we are trying to predict multiple behaviors simultaneously. For that, we can represent the same equation, but in matrix form:

$$Y = XW + b$$

Now, *X* and *Y* are matrices representing the neural features and behavior, respectively. *W* is the "slope", but now in multiple dimensions, and is also a matrix. Finally, the bias *b* is now a vector that displaces the overall "line" fit.

Similar to the 1D case, we can find a *W* and a *b* that fit the best line to the data. The solution is given by least squares, and we have two options, depending on whether we want to include a bias or not. Without a bias, the solution looks like this:

$$W = (X^T X)^{-1} X^T Y$$

Where *X* is a matrix that contains all our neural features over time (matrix of size TxN, with T being the total time points, and N being the total features), and *Y* is a matrix that contains the output behavior over time (matrix of size TxB, with B being the number of output behaviors). Now, the cool thing about linear algebra, is that to compute the bias we can do a little trick, which makes calculating it very easy. The trick involves adding a column of ones to the *X* matrix; Since in the equation we have the multiplication *XW*, then by computing *W* with X containing an extra column of ones, the last row of *W* will give us the bias for free. The equations for that would look something like this:

$$\hat{X} = [X\ 1] \text{ (shape TxN+1)}$$

$$W = (\hat{X}^T \hat{X})^{-1} \hat{X}^T Y \text{ (shape N+1xB)}$$

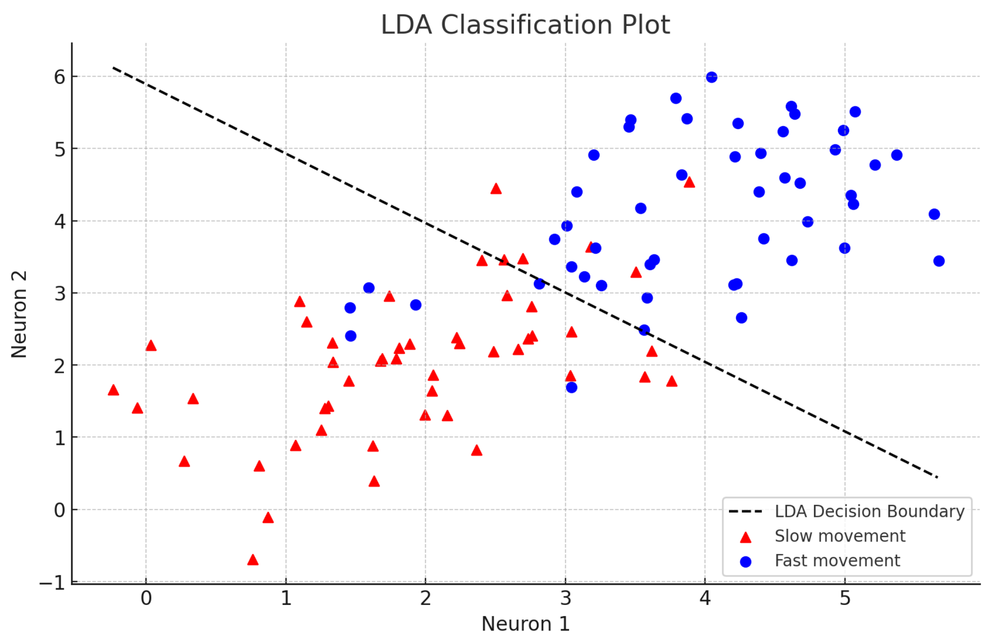
When implementing in Matlab, make sure to **add the column of ones to X, both at training time and at testing time**. Here is an example:

```
% Generate synthetic data
T = 100; % Number of time points
N = 5; % Number of features
features = randn(T, N); % Randomly generate feature data
beta1 = randn(N, 1); % Random coefficients for first output
beta2 = randn(N, 1); % Random coefficients for second output

% Generate outputs with some added noise
output1 = features * beta1 + randn(T, 1) * 0.5;
output2 = features * beta2 + randn(T, 1) * 0.5;
outputs = [output1, output2]; % Combine outputs into a single matrix
% Split data into training and testing sets (80% training, 20% testing)
cv = cvpartition(T, 'HoldOut', 0.2);
idxTrain = training(cv); % Indices for training set
idxTest = test(cv); % Indices for testing set
```

```matlab
% Separate the training and testing data
featuresTrain = features(idxTrain, :);
outputsTrain = outputs(idxTrain, :);
featuresTest = features(idxTest, :);
outputsTest = outputs(idxTest, :);

% Train the model using least squares
XTrainHat = [featuresTrain ones(size(featuresTrain, 1), 1)];
YTrain = outputsTrain;
W = (XTrainHat' * XTrainHat) \ XTrainHat' * YTrain;

% Test the model on the test data
% First, add column of ones to X
XTestHat = [featuresTest ones(size(featuresTest, 1), 1)];
YTest = outputsTest;
% Get predictions
YTestPred = XTestHat * W;

% Get mean-squared error and correlation
mse = mean((YTest - YTestPred).^2, 1)
correlation = diag(corr(YTest, YTestPred))
```

Classification

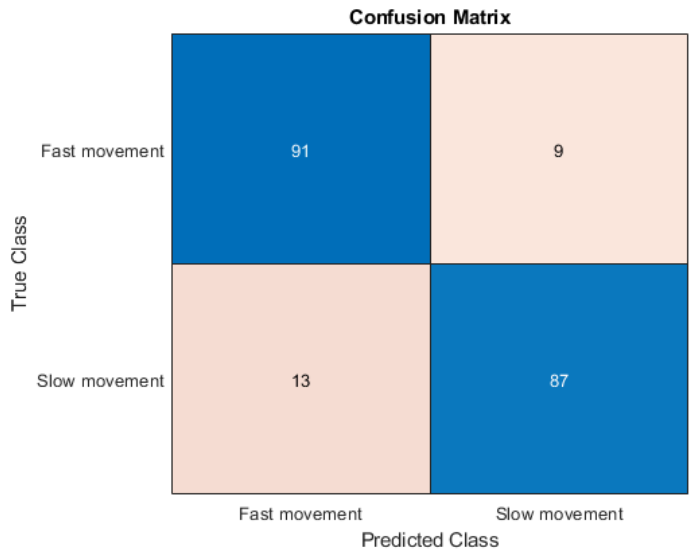
Sometimes, instead of decoding continuous behavior, we are interested in predicting a state from a limited set of possible states. For example, we may be interested in, given some neural activity, determining whether the subject was awake or asleep or, if awake, if they were moving fast or slow. This is called a classification problem, and one of the simplest classification algorithms is the Linear Discriminant Analysis (LDA). This algorithm tries to find the line that separates the data in a way that maximizes the classification accuracy. For example, if we had two neurons firing, and we were trying to determine whether the subject was moving his arm fast or slow, we would get something like this:



In Matlab, we can use the built-in functions to train and test the LDA algorithm. Here is an example, with randomly generated data.

```
% Generate synthetic data
rng(0); % For reproducibility
mu1 = [2, 2]; Sigma1 = [1, 0.5; 0.5, 1];
mu2 = [4, 4]; Sigma2 = [1, 0.5; 0.5, 1];
numDataPoints = 500;
X1 = mvnrnd(mu1, Sigma1, numDataPoints);
X2 = mvnrnd(mu2, Sigma2, numDataPoints);
X = [X1; X2];
y = [repmat("Slow movement", numDataPoints, 1); repmat("Fast movement",
numDataPoints, 1)];
% Split the data into training and testing sets (20% holdout)
holdoutRatio = 0.2;
cv = cvpartition(size(X,1), 'Holdout', holdoutRatio);
idxTest = cv.test;
XTrain = X(~idxTest,:);
YTrain = y(~idxTest);
XTest = X(idxTest,:);
YTest = y(idxTest);
% Classify using LDA
YPred = classify(XTest, XTrain, YTrain);

% Compute and visualize confusion matrix
[C, order] = confusionmat(YTest, YPred);
figure;
cm = confusionchart(C, order);
```

For which the result is:



Assignment

Datasets

Regression dataset

The regression dataset contains the X and Y position and velocity of a monkey doing a 2D reaching task. 95 channels of intracortical neural data were recorded while the monkey performed the movements, and then kinematics and neural activity were grouped in 100ms bins. Each row of the dataset is then a single time bin, and there are 99 total columns. X_1 and X_2 contain the X and Y positions, respectively, while X_3 and X_4 contain the X and Y velocities. channel_Z contains the threshold crossings for channel Z in the corresponding time bin. You can easily load and format the data using the following code. The challenge is to accurately predict position and velocity for X and Y using a linear regression.

```
TTrain = readtable("regression_train.csv");
% Extract features and behavior
columnsFeatures = startsWith(TTrain.Properties.VariableNames, 'channel');
columnsBehavior = startsWith(TTrain.Properties.VariableNames, 'X_');
XTrain = table2array(TTrain(:,columnsFeatures));
YTrain = table2array(TTrain(:, columnsBehavior));
```

Classification dataset

The classification dataset contains aggregated trials for a monkey while 95 channels of ECoG activity were recorded. Each row corresponds to a different trial, and the “Direction” column determines what direction the monkey reached to during that trial. The columns “channel_Z” correspond to the average ECoG activity for the corresponding channel Z during each trial. The objective is to accurately classify what direction the monkey was reaching. You can load the dataset using the following code:

```
TTrain = readtable("classification_train.csv");
% Extract directions and features
columnsFeatures = startsWith(TTrain.Properties.VariableNames, 'channel');
XTrain = table2array(TTrain(:,columnsFeatures));
YTrain = categorical(TTrain.Direction);
```

Linear Regression

1. Spend some time looking at the data. Plot all of the behavior variables. How do the position and velocity traces differ?
2. Similar to the example shown in the theoretical background, train a linear regression model using the “regression_train” dataset. Make sure you add the necessary column of ones so that the resulting W matrix contains the bias.
3. Test the model on the “regression_test” dataset. Compute correlation and mean-squared error between the prediction and the test set, and visualize all behavior variables with their respective prediction using a 2x2 subplot.
4. (Optional) Previous research has shown that the behavior is not only related to the instantaneous neural activity but also to the activity from a few hundred milliseconds in the past. Using the provided function, add history to your neural features, train and test a model, and then visualize the results. Try with different amounts of history. How do these results compare to not adding any history?

Classification with LDA

1. Similar to the example shown in the theoretical background, now train an LDA model using the “classification_train” dataset.
2. Test the model on the “classification_test” dataset, and visualize the results using a confusion matrix.