NENS 230

Assignment #8

Statistical Tests and Principle Components Analysis (PCA)
Due: Tuesday, November 18th, 2014

Goals

- Use MATLAB to perform statistical testing (T-test)
- Learn to perform dimensionality reduction using Principle Components Analysis (PCA)

Problem 1: Statistical testing on behavioral data

Introduction

Neural plasticity, or the ability of neural circuits to change, is essential for both normal brain function and recovery from injury. After stroke, increased plasticity is associated with faster recovery. On the other hand, inflammation can impair plasticity and exacerbate damage. In a recent paper, researchers at Stanford tested whether mice that lacked certain immune system molecules might recover faster from a stoke (Adelson et al. 2012).

One set of mice were missing the gene for PirB, an innate immune receptor for MHC molecules. In other mice, two of its ligands (MHCI Kb and Db) were deleted. These molecules are known to be upregulated during inflammation, and to limit synaptic plasticity in healthy brains. We will analyze some mock data to determine if PirB deletion improves motor recovery.

Hypothesis testing

Lets see if PirB deletion improves recovery from stroke. The provided Excel file contains data from mice walking across a horizontal ladder as a test of motor performance. When a paw slips through the ladder, a foot fault is recorded fewer faults mean better performance. Figure 1C from the paper shows data on MHC1 KbDb knockout animals and is a good model for what your final plot should look like here the MHCI knockouts have improved motor performance!

Name your script for this assignment `assignment8.m`. Load data from the provided Excel spreadsheet footfaults.xls using xlsread. Each column is a different time point, each row a different mouse. Worksheet 1 contains data from 8 control animals tested at 2, 7, 14, 21, and 28 days after stroke. Worksheet 2 contains data from knockout animals missing the gene for PirB. Plot a time series of control and mutant data. Include bar indicating standard error and label the x-axis with the correct day numbers. You can use the provided `errorb()` function to add the error bars, but you'll need to calculate the standard error.

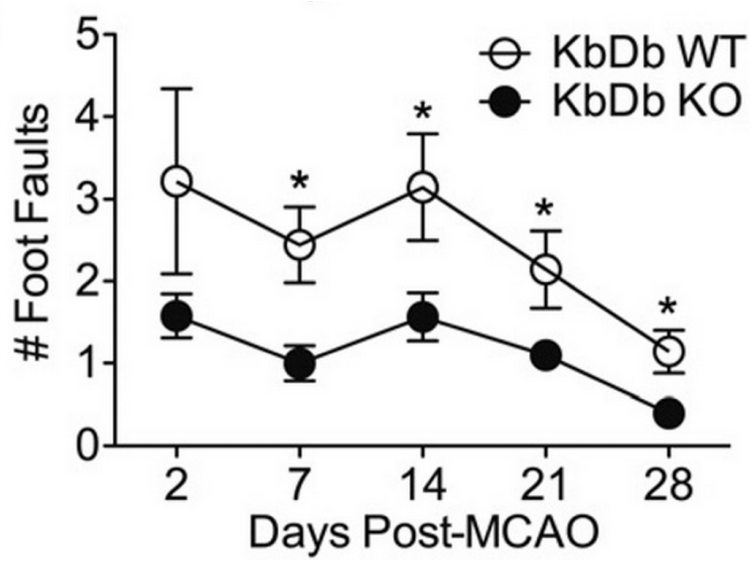

Figure 1:

Test for significance at each time point, using a t-test with the standard significance level of 0.05 but correct for multiple comparisons since the data at different time points are not independent! An easy way to do this is known as the Bonferroni correction (http://en.wikipedia.org/wiki/Bonferroni_correction), in which you simply make each test more stringent by dividing the required p value by the number of comparisons. So to ensure an overall false-positive rate of 5%, with comparisons at five time points, you would make the effective p-value 0.05 divided by 5. Since we are making 5 non- independent tests (one for each time point), we need to adjust the significance level by a factor of 5 to alpha = 0.01. If a time point is significant, plot a star at a fixed distance above the control error bar. Your plot includes mock data which is different from that in the figure shown here, so don't worry if the plots don't look identical. The general form, placement of the asterisks, etc. should be similar, however.

Reference Adelson et al. (2012). Neuroprotection from Stroke in the Absence of MHCI or PirB. Neuron, 73, 1100-7.

Problem 2: Dimensionality reduction using PCA on neural spiking data

In this problem, you'll use dimensionality reduction to find low dimensional structure from high dimensional neural data from the motor cortex of a monkey making reaching movements to targets presented on a touch screen. The data is stored in a struct array in `reachData.mat`, and features 109 simultaneously recorded units (neurons) in PMd and M1 of a rhesus macaque monkey. In many fields, and neuroscience in particular, PCA is used to visualize abstract and high dimensional data and to develop intuition on datasets that would otherwise be difficult to interpret. While this technique is incredibly general, here we consider data that comes in the form of the firing rates of many neurons evolving over time, for several discrete conditions, corresponding to reaches to specific targets. This gives us a visualization of the neural state and how it changes over time in a condition-specific manner. We've performed the following preprocessing steps on this data for you:

1. Align spike times for multiple trials to each target
2. Convolve spike trains with gaussian kernel to smooth data over time
3. Averaged the smoothed activity for all trials to a given target to reduce noise

The resulting struct array contains aligned, trial-averaged data for reaches to eight targets and has the following fields:

1. targetDirection: direction of the target in degrees (don't worry about this)
2. firingRateNorm: matrix of normalized firing rates of all neurons with the shape: [#time points $\times$ #neurons]

We'll use the `princomp()` function to reduce the dimensionality of the dataset down to a few principle components. Each principle component can be thought of as a prototype for a well represented pattern of activity in the population. We can then represent the activity of individual neurons as a weighted combination of these principle components or *basis vectors*. This is particularly helpful when we want to visualize the activity of the population in two or three dimensions instead of the full 109 dimensional space.

Our data is currently stored in a struct, but we can think of it as a data cube with three dimensions: 1) Time, 2) Neurons, 3) Condition. PCA only takes a matrix as inputs, and we'll need to reshape our data accordingly. Create a variable to store the full data matrix, and vertically concatenate the data from all of the reaching conditions into the single matrix. Hint: start with an empty array, and then vertically concatenate the data from each condition to the bottom of this array (In this case, MATLAB knows how to concatenate an empty array with a full one, so it doesn't matter that the empty array isn't the same size as the full array).

`princomp` treats the rows of this matrix as individual observations and columns as variables. This is consistent with the structure of the data provided. The function then returns three outputs:

1. `COEFF`: Coefficients for the components. We won't be using this directly
2. `SCORE`: The principle component scores, or how much of each component it takes to create the observed data points. We'll be using these to plot our neural trajectory over time.
3. `Latent`: The eigenvalues of the covariance matrix formed from the input data. These values tell us how much of the total variability in the data is explained by each of the observed components.

If you'd like to learn more about PCA and develop an intuition for how the PCs are found from a set of data, check out:http://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/ or for more depth: http://arxiv.org/abs/1404.1100 and then download the pdf.

We want to plot neural trajectories as they evolve over time. To do this, we want to unpack the SCORES matrix returned by `princomp` and place the appropriate sub-matrices into our reachData struct array. The shape of the SCORES matrix is $(\#timepoints * \#conditions) \times \#neurons$, and the components are ordered by how much variance they explain. The first column is this top principle component (the PC with the largest eigenvalue), and the eigenvalues decrease with the higher components stored as additional columns 2-end. To extract the sub matrices, iterate over the eight conditions, and pull out just the rows that correspond to

data from that condition. For instance, the first condition corresponds to rows 1-1000, the second 1001-2000, etc. Store these sub matrices in `reachData(condition).scores` as you iterate through.

We want to plot the resulting trajectories from the top components. We want to use the `plot3()` function where the first argument is going to be the first column of the `reachData(condition).scores` matrix, the second argument is the second column, and the third argument is the third column. Iterate over the eight conditions, plotting the trajectory corresponding to the top three PC's for that condition. After plotting the eight trajectories, use the command `view([60 30])` to set the camera angle to a specific position, which should make your plot should resemble Figure 2. Remember to label your axes with the appropriate PC #.

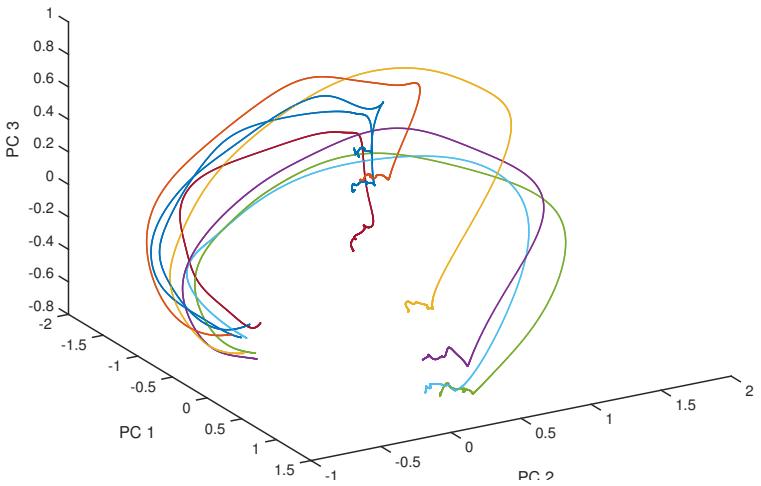

Figure 2:

Lastly, we'd like to know the dimensionality of the data. When we perform PCA, the distribution of the eigenvalues (the `latent` variable returned by `princomp()` tells us whether the data is well captured by just a few PC's (low dimensional) or whether the neurons are largely independent of one another, and require more PC's to capture the structure. Make one last figure in which you plot the # of dimensions (x-axis) vs. the cumulative sum of the eigenvalues normalized by the sum of eigenvalues. You can use the line: `cumsum(latent)./sum(latent)` to accomplish this. This is plot tells us how many dimensions (x-axis) are required to explain a certain amount of variance in the data (cumulative variance explained). Plot a red line at .9 - The point at which your plot crosses this line represents the number of dimensions required to explain 90% of the variance in the data. In our case, this is MUCH lower than the number of neurons we recorded, which tells us that there is significant low-dimensional structure in this data. If, however, the neurons we recorded were much sparser, or more independent, then we would require more dimensions to capture a certain percent of the variance.

Submission

When you are finished, publish your work with the `publish` command. Email the generated PDF file, along with your code, to `nens230-aut1415-staff@lists.stanford.edu` with `[Assignment 8]` in the subject line to submit your assignment.