EEG Network Project

One suggestion is to study "networks" in the brain. Loosely speaking a network is parts of the brain that are active and communicating together in a persistant way representing the state of the brain. For instance there is a "default network" which is what the brain is doing when it is not focused and the mind is relaxed and just wondering.

If we interpret these networks more literally as an undirected graph, then we would like to find a stable set of undirected graphs at different times. As time passes these networks will change, transitioning from one to another but we hope we can find a fixed set of networks representing states and/or patterns of activity.

Background Data

EEG measures electrical activity at various locations on the skull. This measurement usually due to an aggregate of brain electrical activity although the signal often also contains "noise" due to muscle activity such as movement or blinking of eyes. The activity of the eyes is separatly measured so it can be removed. There are also methods (ICA) to remove muscle activity. In the suggested DATA set, a script in included to process the raw data and remove (mostly artifacts).

Here are some suggestions for an EEG data project.

Data:

64 Channels of EEG data are

https://osf.io/chav7/

You should read the Readme files and the wiki for details on the data. Be aware that this data was collected in a project to detect a feature called a spindles and this project is not related to spindle detection. You do not have to understand what a spindle is or how they are being detected.

Suppose that the data is for 30 mins. We want the rows of our data matrix to represent observations, in other words each row should represent a fixed time. The EEG is sampling once every millisecond so there would be 64 columns and 30*60*1000 rows in the raw data matrix. Actually the ICA will remove artifacts and in the end you may wind up with 61 columns after the motion is removed.

Pre-Processing

Reading the data, as explained in the documention requires using mne. And looking at the file inspect_and_clean.py and other python files included with the data. You need to use inspect_and_clean.py to clean the data. When you do a line plot of each 10 second segment for each of the 61 channels of the data **after** cleaning you should see some squiggly lines that have roughly mean zero. Red flags (bad) are

(1) spikes ... if you see something that looks pretty flat with short large spikes, the data still has artifacts. The real data is the stuff that looks flat. Those spikes will render everything else meaningless. It is like trying to listen for a heartbeat while the neighbor is having a rave party. This should be mostly taken care of by ICA in inspect_and_clean.py.

(2) non-mean zero ... if the data drifts up and down considerably so it doesn't end where it started, it isn't detrended. There is a function for detrending ... look it up. This should mostly be taken care of by the notch filtering between 0.3 and 30 Hz. The filtering below 0.3 hz should detrend.

When you are done you should see what looks like really wigly data. You can see some of the ipython notebook figures included with the data. Some of those figures the wiggles follow a curve that kind of drifts up and down violating (2) ... it shouldn't look like that. That is before the cleaning. The cleaning is everyting. If the data isn't cleaned or has drift, everything you do afterwards will be **complete garbage.** You could get great clusters, or terrible clusters but it will all be nonsense... you might has well have done it on random numbers.

Time Windows

A single set of 61 microvoltages at a time doesn't represent that much by itself. It is similar to reading a single letter from an essay: doesn't tell you that much about the meaning at that. Instead we will be considering chunks (windows) of time. This is more like looking at words or sentences instead of individual letters. How long those windows should be is a question. The prior group that look at it in visualization looked at 30 second windows. If the data is only 30 minutes long that would only be 60 rows which is probably too few rows to get a reasonable sample. If one looks at 10 second windows then one has 360 (=30 mins * (60 sec/min)/10 sec) rows which could be enough for clustering. If you use 10 second windows then each row will have 61*10000 = 610,000 columns. For the momement I will assume 10 second windows.

Dimension Reduction (possible)

610,000 columns is crazy. Besides making everything hard to compute there is probably too much noise in there. The problem with noise is that it can make two related things look unrelated. For each 10 second window (row) and each of the 61 channels we had 10 * 1000=10,000. We could try and describe each vector in 10,000 as a much smaller array. To do this onece for everybody, take the 61 channels and the 360 rows and make them separate rows so we now have 360 * 61 = 21,960 rows and 10,000 columns.

You may have to double check none of those is too weird in terms of spikes or drifting or it will throw off your answer. You can take the mean row, which will have 10,000 and make a histogram of the distance from each row to the mean. Look at the top 10% plot them. If they look very different from a typical row, you may need to throw them out.

Now run PCA (fit) and look at the eigenvalues (powers/aka explained variance) like we did in class and look at the elbowmethod. See how many dimensions you really need. You may only need 20-100. Use the transform method to project into the

lower dimensional space. Whatever dimension you come up with, you should have an argument that that is a good number. Don't just use the numbers I am stating here. After you have fit and projected to, lets say 100 dim, then go back to the origanal matrix projecing the 10,000 column dims in each channel into 100 dim.

After dimension reduction to 100 dim/channel, you would have 360 rows again, but this time 61*100 or 6,100 columns.

One Way to Cluster

The first idea is to cluster think of each time separately. For each row you have 61 channels of 100 dim. So for each time you can make a new data matrix with 61 rows and 100 columns and cluster that. You can try different clustering methods like k-means, mean-shift etc. You will need to look at the results so you may do pca **again** (**after** clustering) to project to 2D and color the points by cluster. If it looks like a really bad mess the clustering *may* not be great. You can also use t-sne instead of PCA and see if the colored points clump to gether better.

You also may want to try at different times (you have 360) to see how much it changes and how much it stays the same. This is something you should try but there is a problem. The 61 channels are related to locations in the brain and so close electrodes **will** be close in the 100 dim data space. This is not because they are part of the same "network" but because they are measuring the same activity beween them. This effect may be difficult to remove using this approach.

Graph Approach

For 61 channels there are 61 * 30/2 = (2 choose 61) = 1830 pairs of channels. A sample pair channel A and channel B you could make a scatter plot (you should). If A and B are far about the scatter should be pretty different (messy cloud of 100 points). If A and B are close they should be tighter. You can think of each of the 1830 pairs as an edge of a graph between two of the 61 nodes at a 10 second time. If thos are similar we should have strong edge, if not, we should have a weak edge. We need a similarity measure to figure out how strong the edges are.

Similarity Measures/Distances

It is not clear at all which notion of similarity or distance is right. You are probably going to need to try a few and see which works best.

pearson-r If you measure the pearson-r (scipy.stats.pearsonr) this should be close to 0 if they are far appart and closer to 1 (or -1) if they are nearby. Pearson-r is one measure of similarity.

cos-similarity Another is cos-similarity which we went over in class. You should definitely look at this too. This should be almost the same thing, if v is the vector from channel A and w from channel B, the sin similarity = dot(v,w)/(sqrt(dot(v,v)) * sqrt(dot(v,v))).

euclidean-distance/similarity Unlike the similarities above we can look at the plain old root-mean-square distance. The distance d(v,w) = sqrt(dot(v-w,v-w)). With a distance 0 is close (similar) while a big number is dis-simlar. To make this a simialrity we can s_euclid(v,w) = exp(-dot(v-w,v-w)/sigma), where sigma is some number one

needs to play with to get right. If it is too big everything will be too similar and if it is too small, nothing will be similar. This is a hyperparameter.

There are more but thats probably enough for now. With any of these measure we get a number for each pair. That means we get a matrix with 360 rows (the 10 second intervals) and 1830 columns (the similaries between channels by comparing the 100 dim vectors for each channel).

Clustering Similarities of Networks

Now we can try to eliminate the problem of similarities due to proximity. One way is that we take the mean of the 360 rows which is a single 1830 column, and subtract it from each of the rows. This is the kind of thing we do in pca. If we do that then what is left over is how **more** similar two rows are, than average. At this point we can apply k-means, mean-shift or other clustring to the 360 columns and see if there are clusters. Members of the clusters are **times** when the graphs are similar. The mean of such a cluster would be a graph ... network. We could then threshold the values of the 1830 ... again that theshold would be a hyper-parameter. You might even need to do that before clustering. The connected components which were not just single nodes, would be the parts of the network. If different times showed different network, this would be pretty interesting. If the times that were grouped were marked as the subject doing similar things, that would be very interesting too.

Evaluation/Validation

There are several data sets. As usual train-test, and cross validation are pretty important to find hyper-parameters (like k in k-means or sigma in a similarity matrix, or the dimension in dimenstion reduction). For instance the k-means predict that the means of the clusters will be close (small distance) to 1830 dim vectors of the leave-out set. You can measure and see how changing the sigma, or the dim in dimension reduction makes that number go down/up and pick the smallest. After you did ALL your optimization and training, you want to look at a completely different subject to see if the networks you predict from one ... are the same you see (with the same parameters from first subject ... same pca fit and all) in the other. If so congrats!

Contacts

Enan Rahman <enan7890@gmail.com> and Fioger Shahollari <fioger95@gmail.com> worked very hard on putting together a good visualization. They worked through bi-clustering and didn't have the python code that is now included with the data set but they learned many valuable lessons.