- go over homework5
- go over project requirements: https://scipycourse2022.github.io/project

dimension reduction

- say you're taking 20 different types of measurements for each sample:
    - e.g. 20 different measurements per animal (size, weight, colour, sex, etc), each animal is a sample
    - e.g. recording activity from 20 neurons simultaneously, one measurement of firing rate per neuron trial, each trial is a sample
- your whole dataset is 20 dimensional, and can be described by an nsamples x 20 array:
    - one sample per row, one type of measurement per column
    - each column is considered a dimension
- are all the dimensions independent of each other? which ones should you consider when trying to cluster your data, and which are correlated with each other and therefore redundant?
- can't visualize data in 20D space, but you can in 2D or 3D
- dimension reduction algorithms can look for redundancy in the data and project it into a new lower dimensional space that still captures the original data fairly well, without throwing away too much information
    - imagine trying to maintain the spatial relationships between points in the dataset, even after reducing it from high D to low D -- that's what dimension reduction algorithms generally try to do
- most common kind of dimension reduction PCA: principal components analysis
    - PCA looks for directions of maximum variance in the high D data, and rotates a low dimensional set of (orthogonal) axes to align with those directions, such that those axes are best able to explain as much of the variance as is possible in that low D space
- *PCA demo*
- lots of other kinds of dimension reduction, or "decompositions", in `sklearn.decomposition` :
    - http://scikit-learn.org/stable/modules/decomposition.html
    - very nice description of PCA, by former neuroscientist Jonathan Shlens:
        - "A Tutorial on Principal Component Analysis": https://arxiv.org/abs/1404.1100

clustering

- now that your data is lower dimensional (typically 2D or 3D), you can plot it and look at the distribution
- the data points come without labels, i.e. they start out unclustered, all have the same colour

- does the data naturally fall into various clusters/categories, or is it just one big continuous cloud of smoothly varying data points?

- if they **do** form clusters, then you might want to analyze each cluster separately instead of lumping all your data together

- clustering is a type of exploratory data analysis

- if you see clusters in your data, then one way to label each data point is to manually draw boundaries between/around clusters, those points that fall to one side or within a boundary are given the same label (i.e. cluster ID)

    - this can be tedious and error-prone
    - automated clustering is preferred in most cases, but it's not magic...

- let's look at two example automated clustering methods, and test them on 2D data:

k-means algorithm:

- probably the most commonly used clustering algorithm

0. Randomly initialize a set of cluster centers (i.e. means)
1. Assign each data point to the nearest cluster
2. Update the position of each cluster center by taking the mean of the positions of all its member points. Go to 1.

- After enough iterations, cluster centers will stop moving, and cluster membership of each point will become stable.

- Simple, fast, but it has some limitations:

    - need to specify how many clusters you want it to find (hence the 'k' in k-means)
    - because it uses only distance to assign points to clusters, it performs poorly for elongated clusters

- *k-means demo*

DBSCAN algorithm:

- DBSCAN = "Density-based spatial clustering of applications with noise"

- density-based instead of just distance based

- does better than k-means for elongated clusters

- figures out the number of clusters automatically, but it has two other parameters that have to be tweaked

- doesn't require that every point be assigned to a cluster - allows for outliers

- *DBSCAN demo*

- lots of other clustering algorithms in `sklearn.cluster` , see:
    - http://scikit-learn.org/stable/modules/clustering.html
    - http://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html
- an even better, simpler density-based algorithm:
    - "Clustering by fast search and find of density peaks", Rodriguez and Laio, Science, 2014
    - http://science.sciencemag.org/content/344/6191/1492
    - unmaintained Python library for it: https://github.com/GuipengLi/Dcluster

dimension reduction & clustering exercises

Feel free to copy and paste from the `clustering_demo.ipynb` file to save yourself some time.

1. Load in example multidimensional data in `measurements.xlsx` . Use pandas to load it in as a Dataframe, and then convert the whole thing to a 2D `nsamples, ndimensions` array. Hint: pull the "values" out of the Dataframe.

2. How many samples and dimensions do the data have?

3. Scatter plot the data in the 1st dimension vs. the 2nd, the 3rd dimension vs. the 4th, and the 5th dimension vs. the 6th. Can you see any structure in the data?

4. Do PCA on the data, and reduce it to only 2 dimension (see `clustering_demo.ipynb`). Scatter plot PC1 vs. PC2. How many clusters do you see?

5. Try and cluster the data using both KMeans and DBSCAN (see `clustering_demo.ipynb`). Again, Scatter plot PC1 vs. PC2, but now apply color to each point to show the clustering. Experiment with the `n_components` in KMeans and the `eps` and `min_samples` parameters in DBSCAN to try and extract the clusters as best you can.