homework 13: the Moriarty Brain Atlas

Moriarty is gearing up to do a comprehensive single-cell RNA-seq experiment on the dissociated cells of an entire human brain, which is about 200 billion cells. Figuring on a going rate of about $1/cell, Moriarty plans to write an NIH grant for about $200 billion. Since the NIH budget is only $40B/year, his plan for the "Moriarty Brain Atlas" includes forcing everyone else into retirement. He's been going around the lab making unsubtle comments about using your brain for his project, since you won't be needing it any more.

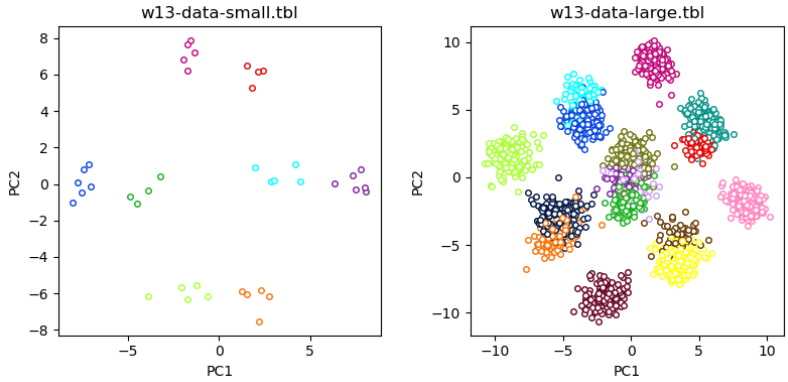
Moriarty's preliminary control data

For his preliminary results section, to demonstrate feasibility, he has collected two single cell RNA-seq data sets of known neural cell types as positive controls, and he's trying to show that he can do cluster analysis to discover cell types.

Dataset 1 (the "small" set) consists of mapped read count data for 40 cells and 24 genes. The last column in the data table identifies the known cell type labels 0-7 for 8 different neural cell types.

Dataset 2 (the "large" set) has mapped read counts for 2131 cells and 32 genes, and the last column assigns labels 0-15 for 16 different neural cell types.

Moriarty has been trying to use PCA to visualize clusters in his positive control data:



Moriarty explains how he constructed his positive controls. Each cell type is specified by a binary code of "high" and "low" expression of n regulatory modules, each encoded by m transcription factors. In the small dataset, there are 3 modules (and hence $2^3 = 8$ different cell types), with 4 TF genes per module (so, 12 TF genes that are varying with cell type), plus 12 other genes that aren't cell type specific. In the large dataset, there are 4 modules ($2^4 = 16$ cell types) with 4 TF genes each, plus 16 other genes that aren't cell type specific. In the small data set, for example, cell type 0 is specified by [lo lo lo] levels of the TF genes in the three modules; cell type 3 by [lo hi hi]; and cell type 7 by [hi hi hi].

The upshot of all his clever construction is that Moriarty expects his control data to fall on the corners of a $n-$dimensional hypercube. He realizes, looking at his PCA plots, that PCA's rigid rotation fundamentally limits his ability to separate clusters of cell types that are combinatorially encoded by transcription factor gene on/off levels this way. The three modules and 8 cell types of the "small" dataset behave like a 3D cube and its 8 corners; imagine rotating a die to maximally separate these corners in a 2D projection, and you'll realize that two of the corners tend to fall on top of each other. The four modules and 16 cell types of the "large" dataset behave like a 4D hypercube and its 16 corners; though this is harder for mere humans to visualize, it's easy to see that as the dimensionality of the (hyper-)cube increases, a rigid PCA rotation

is going to do a less and less good job of projecting the corners to different places in 2D.

Perhaps a nonlinear projection would do better, you tell him. Taking the moral high road, despite those threats about your brain, you offer to show Moriarty how t-SNE works.

1. verify that PCA fails

First, reproduce Moriarty's result. Download dataset 1 and dataset 2. Use PCA to project the data into two dimensions. (Hark back to PCA week, the pset and its answers, if you need to.)

Plot your PCA projections for the two data sets, with points colored according to their known cell types. You should get a figure essentially identical to Moriarty's above.

Why does PCA fail here?

2. implement t-SNE for yourself

Implement your own version of t-SNE, following the description in the lecture notes and the van der Maaten and Hinton (2008) paper. Look especially to "Algorithm 1" in van der Maaten and Hinton (2008) for an outline.

Your version should implement:

- the objective function (the Kullback-Leibler distance between the high-dimensional P and the 2-dimensional Q, equation 2 in the van der Maaten paper), and its gradient (eqn 5);
- for that, you will also need to implement the calculation of P (pg. 2584) and Q (eqn 4);
- for P, first you need $p_{j|i}$ (eqn 1), and for $p_{j|i}$, you need to calculate σ_i values for each point i given a target perplexity value. You'll

probably find that the σ_i optimization is the most annoying step; see lecture notes and the hints below for help on doing this one-dimensional optimization of each σ_i.

You *don't* need to implement the t-SNE optimization for the pset (though you can if you want! I'd be impressed.) Instead, use SciPy's `scipy.optimize.minimize()`, as we used in regression in week08. Write your objective function and gradient routine to be compatible with the SciPy optimizer.

t-SNE's objective function isn't convex. There are lots of local optima, and it's easy for it to get locked up in crappy spurious ones. Real t-SNE implementations are decked out with bells and whistles in their optimization. "Early compression" and "early exaggeration" allow points to slide past each other in early iterations to find a better global clustering. As the number of data points increases, these tricks become more and more necessary. If you try the larger Moriarty data set, you'll probably find, as I did, that the SciPy optimizer doesn't suffice for it. It will suffice on the smaller Moriarty data set though.

Use your t-SNE implementation to visualize Moriarty's smaller data set in two t-SNE dimensions; show your t-SNE figure.

3. using the canned t-SNE from scikit

scikit-learn provides a suite of machine learning tools in Python, including `sklearn.manifold.TSNE`. You probably already have it installed with your Anaconda installation. See if "`from sklearn.manifold import TSNE`" works for you. If not, install `scikit-learn` with conda:

```
% conda install scikit-learn
```

Use `sklearn.manifold.TSNE` to visualize both the

small and large Moriarty data sets in two t-SNE dimensions, for four different choices of perplexity: 2, 5, 30, and 100. Show your plots. How does the choice of perplexity affect the result? Comment on your findings.

turning in your work

Submit your Jupyter notebook page (a `.ipynb` file) to the course Canvas page under the Assignments tab. Please name your file `<LastName><FirstName>_<psetnumber>.ipynb`; for example, mine would be `EddySean_13.ipynb`.

Remember that we grade your psets as if they're lab reports. The quality of your text explanations of what you're doing (and why) are just as important as getting your code to work. We want to see you discuss both your analysis code, and your biological interpretations.

---

hints

- Part 1 is a warmup, basically making sure you can load the data. If you need a hint here, you can download my script for producing each half figure. If you get stuck on piddly non-tSNE issues with these data, the comments in this script might help you.
- Part 2 is the hard part, obviously. If you bog down in it, skip ahead and do part 3 first. Using the scikit-learn t-SNE implementation gives you something to compare your implementation against.
- In Part 2, for fitting σ_i, you can define an objective function that calculates the difference between the desired perplexity and the calculated perplexity given a current guess at σ_i. You want to find the σ_i such that this difference is 0. This becomes a one-dimensional root finding algorithm, which

you can solve with `scipy.optimize.bisect()` or the fancier `scipy.optimize.brentq()`. You need to bracket the root first, by making sure you have two guesses a, b for σ_i that give perplexity differences of opposite sign (and therefore there must be a value of σ_i that achieves a difference of 0 somewhere in the (a, b) interval).

- van der Maaten recommends a default perplexity of 30, which is what scikit-learn defaults to. You'll probably find that you need to set a lower perplexity on the smaller Moriarty dataset. For such a small number of data points, it doesn't make sense to set the perplexity (the effective number of neighbors) much larger than your rough expected cluster size.

- Part 3 is intended to get you curious and playing around with how a t-SNE clustering varies as you play with the perplexity -- and to get you wondering how it would perform on other positive control data where you know what the clusters are supposed to look like. A t-SNE article with beautiful visualizations on distill.pub will give you more ideas.
