Problem Set 2: Graph Clustering and Signal Processing

Unsupervised Learning for Big Data
CPSC 453 / CBB 555 / CPSC 553 / GENE 555

Assigned: Monday, October 5th
Due: Monday, November 2nd, 11:59pm

1 Introduction

In this assignment we will explore graph clustering and signal processing, two common paradigms for analyzing data that can be represented as a graph. In our lectures we learned that we can induce a graph from any high dimensional dataset using a Gaussian kernel; thus, these graph based methods are generally applicable to any high dimensional dataset.

Clustering is a key method to understand and unravel heterogeneity in high dimensional data. In general, cluster analysis must start with a set of assumptions on the data. For instance, with k-means, the assumption is that our clusters are spherical and roughly the same size. On the other hand, agglomerative methods like Louvain do not have such restrictions on geometry, but assume that we wish to approximately maximize some target function (such as modularity). Louvain is a graph-based agglomerative method; *spectral clustering*, on the other hand, is related to k-means in the sense that it is a partition based method. However, unlike k-means, spectral clustering uses the graph Laplacian eigenvectors as a set of coordinates. This means that clusters are formed using graph geometry. Clustering has been used in a variety of biological applications including gene clustering, cell subtype finding, patient type identification, etc.

Separately, graph signal processing has emerged in recent years as a tool for manipulating data that rests on a graph. A major goal of this field is to take some of the major advances in classical signal processing and apply them to graphs. In lecture, we learned about the Graph Fourier Transform, which allows us to study the behavior of data in the frequency space, which encodes geometric information and the behavior of neighborhoods. In this assignment, we'll see that these ideas are intricately linked to the ideas of spectral clustering.

2 Preliminaries

In order to understand spectral clustering, it is useful to first understand some key features of graph Laplacian spectra. In class, we discussed the combinatorial graph Laplacian

$$\mathcal{L}^c = D - W, \tag{1}$$

and the normalized graph Laplacian

$$\mathcal{L}^s = D^{-\frac{1}{2}}\mathcal{L}^c D^{-\frac{1}{2}}, \tag{2}$$

where D is the degree matrix for a graph on N vertices with weights W, i.e. $D = \text{diag}(d(i)), d(i) = \sum_{j=1}^{N} W_{ij}$. In the following discussion, I will specify general Laplacians as $\mathcal{L}$ whereas vectors and matrices related to a specific normalization will feature a superscript.

We refer to the eigenbasis of the Laplacian $\mathcal{L} = \Psi\Lambda\Psi^T$ as the *Fourier basis*. The eigenvectors form a basis for frequency analysis.

2.1 Ordering the Laplacian Basis

Abusing notation, let's write the set of eigenpairs as $B = (\Psi, \Lambda)$ where $\Psi = \{\psi_i\}_{i=1}^N$ and $\Lambda = \{0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N\}$. Note that our eigenvalues are ordered in reverse from the diffusion map $M^s = \Phi\chi\Phi^T$ eigenvalues $\chi = \{1 = \mu_1 \geq \mu_2 \geq \cdots \geq \mu_N\}$. This ordering allows us to retain classical notions of "high" and "low" frequency. The smoothness of our Fourier basis functions corresponds to their eigenvalue. The ordering holds for both $\mathcal{L}$.

2.2 Graph Fourier Transform

We will compute the orthonormal expansion of signals in the Fourier basis of our graph. The Graph Fourier Transform gives us this representation. For a signal $s \in \mathbb{R}^N$ defined on the vertices of a graph, we have

$$\hat{s}(\lambda_l) = \langle s, \psi_l \rangle, \tag{3}$$

so

$$\hat{s} = \Psi^T s. \tag{4}$$

As in the classical Fourier transform, we have a unitary transformation here so we can perform the Inverse Graph Fourier Transform as

$$s = \Psi\hat{s}. \tag{5}$$

Throughout this problem set we'll examine this frequency-based representation of input signals such as cluster labels.

2.3 Laplacian Spectra

We should observe a few things about the eigensystems of $\mathcal{L}$. We may refer to the distribution of the graph Laplacian eigenvalues as the *spectrum* of a graph, in the same way we refer to the distribution of frequencies as a spectrum. The meaning of the eigenvalues of $\mathcal{L}$ (i.e., how they relate to graph properties like connectivity) is the key subject of the field of Spectral Graph Theory. Many of the results in Spectral Graph Theory are related to very specific graph constructions, so in general it is very hard to describe the spectrum of an arbitrary graph like one we consider. That said, some applicable key results revolve around the magnitude of λ_2 and its relationship to connectivity and mixing times. Others exist, but we will not cover them in this course. It is worth noting that many of the design choices for our algorithms revolve around constructing a graph that has certain properties (adaptive bandwidth, positive semi definite kernels) or manipulating a graph spectrum in specific ways. Let's make a few easy derivations that are good things to know for all graphs.

The normalized Laplacian $\mathcal{L}^s$ and the combinatorial Laplacian $\mathcal{L}^c$ do not share eigenvectors or eigenvalues. Let's focus on the combinatorial Laplacian first. The easiest observation to make is that the first eigenvector of $\mathcal{L}^c$ is the all ones vector, $\psi_1^c = \mathbb{1}$. This eigenvector has eigenvalue $\lambda_1^c = 0$. To derive this, note that the row sums of $\mathcal{L}^c$ are 0 and multiply $\mathcal{L}^c\mathbb{1}$. Beyond λ_1^c, the magnitude eigenvalues of the combinatorial Laplacian may scale with degree, though many other properties change the way they distribute.

For the normalized Laplacian, we have the first eigenpair $\{\lambda_1^s, \psi_1^s\} = \{0, \frac{d^{1/2}}{\|d^{1/2}\|}\}$. Because the normalized Laplacian has a degree normalization (the sandwiched inverse square roots), its eigenvalues fall in the interval $0 \leq \lambda_i \leq 2$ for all i where $\lambda_N = 2$ if and only if the graph is bipartite. Earlier we mentioned the Diffusion Map. It turns out that these matrices share eigenvectors and their eigenvalues are easily related. Let

$$M^s = D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{6}$$

we have

$$\mathcal{L}^s = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{7}$$

$$= I - M^s \tag{8}$$

So $\lambda_i^s = 1 - \mu_i$, and $\phi_i = \psi_i$. If we make a normalization or choose W to be positive semi definite, then it is obvious that $0 \leq \lambda_i \leq 1$.

Finally, I want to note that the number of zero eigenvalues for any $\mathcal{L}$ encodes the number of connected components, i.e. a disconnected graph of k components has $k = |\{\lambda_i : \lambda_i = 0\}|$. The corresponding eigenvectors serve as indicators for each disconnected component. To see this, note that the columns and rows of any disconnected Laplacian can be sorted to yield a block diagonal matrix. Thus, the full Laplacian eigensystem is the union of the block Laplacian eigensystems. We will use this in spectral clustering. A fuzzy but useful observation from this is that graphs that are nearly disconnected will have low frequency eigenvectors that have indicator-like properties. This should lead you to some clues about the Fourier coefficients of cluster labels.

To begin, we'll first code up a platform for our synthetic example

3 Filtering Signals on the Stochastic Block Model

1. One of the most common synthetic examples for evaluating clustering algorithms is the Stochastic Block Model (SBM). This model is entirely parameterized by

   (a) The number of points N

   (b) The number of clusters and cluster size

   (c) The ratio of edge probabilities between and within each cluster, $\epsilon = \frac{p_{ij}}{p_{ii}}$.

   - Fill in the function

   ```
   sbm(N, k, pij, pii,sigma):
   ...
   return A, gt, coords
   ```

   - The function will take as input the number of points N, the number of clusters k, and two floats that encode the probability of forming an edge from one cluster to another `pij`, and the probability of forming an edge within a cluster `pii`.
   - To make the adjacency matrix, generate an $N \times N$ matrix and loop through each entry. If the entry connects two points within the same cluster, sample from a uniform distribution and compare to `pii` to generate binary values determining whether an edge should exist (and likewise if they are from different clusters). You might find some clever ways to speed up this process (e.g. by exploiting A's symmetry; or by building a matrix of probabilities, another of uniform samples, and using whole matrix operations).
   - For ground truth `gt`, you'll want to uniformly partition your N points such that you have an $N \times 1$ vector that encodes which cluster they belong to. If the number of clusters does not evenly divide the number of points, distribute the remaining points in order to the lowest numbered clusters (i.e. if there are 5 leftover points and 19 clusters, give one point each to clusters 1 through 5).
   - To generate coordinates for each cluster, use 2-dimensional normal distributions with means equally spaced around the unit circle. Use the variance encoded by `sigma`.

2. Build a Laplacian function

   ```
   L(A, normalization):
   ...
   return L
   ```

   - The function will take as input a graph adjacency matrix `A` and return a Laplacian matrix `L`.
   - If `normalization = False` then use equation 1. Otherwise, use equation 2.

3. Build a function for computing the Fourier basis

```
compute_fourier_basis(L):
...
return e,psi
```

- The function will take as input a graph Laplacian matrix `L` and return its Fourier basis `e, psi`.
- You can use `eigh` to obtain this eigensystem.

4. Build a function for computing the graph Fourier transform (GFT)

```
gft(s, psi):
    ...
return s_hat
```

- The function will take as input a graph signal `s` and a Laplacian Fourier basis `psi`. It will return the GFT of the signal `s_hat`.
- To compute this, take the inner product $\hat{s}(i) = \langle s, \psi_i \rangle$

5. Build a function for filtering signals:

   (a) We'll assume that we are working with exact filters. This means that we already have a precomputed Fourier basis, and we don't need to form an approximation. It turns out that in practice this is slow, and we can actually do approximations to avoid the diagonalization of the Laplacian. However, it's conceptually easiest to build exact filters.

   (b) Define a graph filter as some function h of the Laplacian eigenvalues Λ, such that $h(\Lambda) = \text{diag}\left(h(\lambda_1), \cdots h(\lambda_n)\right)$ and $h(\lambda_i) \in [0, 1)$. This is simply a polynomial that we plug our eigenvalues into, with the restriction that our values range from 0 to 1. Then we can define the vertex-domain filter matrix as

$$H := \Psi h(\Lambda) \Psi^T. \tag{9}$$

   If we want to filter some s, we write Hx as the filtered version. Let's interpret this in terms of the GFT by writing

$$Hs = \Psi h(\Lambda) \Psi^T s = \Psi h(\Lambda) \hat{s}.$$

   We're weighting $\hat{s}$ by $h(\Lambda)$, then taking the inverse Fourier Transform (i.e. for some function frequency valued function $\hat{f}$, $f = U\hat{f}$.

```
filterbank_matrix(psi, e, h):
    ...
return H
```

- The function will take as input a Laplacian Fourier basis `psi, e` and a filter `h`. `h` must be a function that accepts eigenvalues and returns a value in the range of 0 to 1. The function will return the filter bank matrix from equation 9.
- First, evaluate the filter at each eigenvalue. Then take the product from equation 9.
- If you want to filter signals, simply take Hs

In class, we discussed the notion that the eigenfunctions of the Laplacian capture the smoothness of signals. Let's take a look at the cluster indicators from our SBM.

- Generate a set of SBMs with various values of p_{ij}. Start with $p_{ij} = 0$ and work in small increments until the graph is highly connected between clusters. Use 500 points and 8 clusters.

    1. Use `plt.imshow` to visualize the adjacency matrix of your SBMs
    2. Plot the absolute value of the Fourier transform of your ground truth cluster labels for each SBM realization. You can do this using `plt.stem`. Make sure that your x-values correspond to the graph frequencies (i.e. eigenvalues).
    3. At each realization, plot the coordinates of the SBM colored by the second eigenvector of the graph Laplacian.
    4. Use your filter function to filter random Gaussian noise. Try low pass filters which are "ideal", that is, if $\lambda < c$, the filter evaluates to 1. Otherwise, the filter evaluates to 0. Plot the Fourier spectra of a few realizations of Gaussian noise, and plot the filtered noise on the data coordinates.

Question 3.1. *How does the spectrum of the cluster labels changing at various levels of connectedness?*

Question 3.2. *What do the first few Laplacian eigenvectors represent in the SBM? Why?*

Question 3.3. *What does the filtered noise look like on average? What happens when you vary the parameter c?*

4 Filtering Signals on the Swiss Roll

Now, let's generate some signals on the Swiss roll and filter those.

- Using the Swiss roll from Assignment 1, build a graph using the Gaussian kernel parameters that produce a reasonable diffusion map (i.e. it looks like a plane).

    1. Use the coloring from the previous problem set to generate 3 signals: one low frequency, one medium frequency, one high frequency. Plot these signals and show their graph Fourier transform. Report the function that you used to generate them. *Hint:* Trigonometric functions work well here.
    2. Try shifting the *phase* of your signals. Does it work? You could do this by shifting your trig functions along the color vector.
    3. Use your filter function to filter these signals. Try low pass filters which are "ideal", that is, if $\lambda < c$, the filter evaluates to 1. Otherwise, the filter evaluates to 0. Similarly try "ideal" high pass filters. Plot the result of two of your favorite filters for your three signals.
    4. Craft a band pass filter using a Gaussian function. You can do this by setting the mean of the Gaussian to the target λ_k at the middle of the band. Add a normalization out front of it so that you can tune the width using a σ. Now, let

    $$\delta_i(j) \begin{cases} 1 & \text{if } j = i \\ 0 & \text{else.} \end{cases}$$

    Use this to translate your Gaussian to a certain point on the graph. You can do this by taking $H\delta_i$. Translate it over the Swiss roll and note what changing λ_k does to the output. Note: In the classical setting, we might call this a Gabor filter.
    5. Plot all of your filters by evaluating them over the interval $[0, \lambda_N)$. You can use `np.linspace(0,lmax,1000)` where `lmax` is the largest eigenvalue of the Laplacian.

Question 4.1. *What kind of frequency content does a stable clustering have?*

Question 4.2. *What does smoothness mean in terms of graph signals and their frequency spectrum?*

Question 4.3. *Band-limiting is a nice trait if one wants to design an algorithm. It means that there is no frequency content above a certain frequency, i.e. the band limit. For what values of p_{ij} are SBM cluster labels band-limited?*

Question 4.4. *What sort of information is frequency encoding in each dataset?*

Question 4.5. *How is phase encoded on the graphs?*

Question 4.6. *Under what scenarios would you want a band pass or high pass filter? How do these filters work and what do they do to the spectrum of the signals?*

Question 4.7. *(Bonus) Are there any similarities between the band pass experiment and classical translation and modulation?*

I highly recommend the works of [1, 2, 3] for more insights into what we're doing here.

5 k-means Clustering

In the following, you will implement k-means using the k-means++ initialization.

1. Fill in the k-means function

```
kmeans(X, k, nrep=5, itermax=300):
    init = kmeans_plusplus(X,k)
    ...
return labels
```

   - K-means works by iteratively selecting cluster labels that are closest to a point.
   - That is, at each point i, select the centroid of the nearest cluster (use squared euclidean distance)
   - After you have reassigned all points based on their nearest centroid, you then update all the centroids based on their new members.
   - Repeat the process until the cluster assignments stop changing. You can use a threshold here.
   - If the algorithm doesn't converge, you'll want to terminate it when you reach `itermax` iterations.
   - To gain accuracy, you'll want to take `nrep` repetitions and choose the cluster assignments that have the smallest within cluster distance to centroid.

2. To initialize k-means, you'll use the k-means++ initialization. Fill in this function

```
kmeansplusplus(X, k):
    ...
return centroids
```

   - This algorithm has been shown to be a relatively good way to initialize k-means
   - To start, choose a single initial point at random. Let's call this x_1
   - For each remaining point, compute the squared distance from the first point and make it a probability distribution $p(x_j) = \frac{D^2(x_1,x_j)}{\sum_k D^2(x_1,x_l)}$.
   - Sample an x_j with probability proportional to $p(x_j)$. You can use the `np.random.choice` function for this with weights. This will be the next centroid.
   - Repeat this process until you have k initial centroids.

You will first test this on a Gaussian mixture model with K components. Points here are distributed about K means with variance X. To generate this GMM, we'll just use your `sbm` function.

1. Run k-means on your SBM with various values of sigma. (Here you just use the coordinates you generated, ignore the adjacency matrix).

2. Next, generate a concentric spherical dataset by sampling 1000 points from a 3d unit normal distribution. Normalize all points using Euclidean distance. This should form a hollow sphere. Multiply the first 500 points by 10. This should create concentric spheres. Each sphere will be a separate cluster. Run k-means on this.

Question 5.1. *Compare the labels you obtain on the SBM to the ground truth clusters. At what values of sigma does k-means clustering start to fail?*

Question 5.2. *Does k-means converge to a reasonable clustering on the concentric spheres? Why?*

6 Spectral Clustering

Next we will compare k-means to spectral clustering. Spectral clustering works by treating the eigenvectors of $\mathcal{L}$ as coordinates for your points. We will use the method presented in [4]. This is a useful tutorial [5]. Spectral clustering proceeds as follows:

1. Build or obtain a graph

2. Construct the normalized graph Laplacian

3. Take the first k eigenvectors

4. Normalize the rows of these eigenvectors using the ℓ_2 norm

5. Run k-means on this normalized output

Fill in the function

```
SC(L, k, psi=None, nrep=5, itermax=300, sklearn=False):
    ...
return labels
```

Your code will take in a normalized graph Laplacian `L`, the desired number of clusters `k`, and optionally the Laplacian eigenbasis `psi`. If `psi` is not provided, you will compute the first k eigenvectors of the graph Laplacian. You can use `scipy.linalg.eigh` for this.

As with k-means before, we will repeat the clustering `nrep` times to get a more robust clustering. You can just pass this parameter and `itermax` through to `kmeans`. The `sklearn` flag can be used to check your clustering in the absence of your k-means implementation. Consider it a nice performance metric and a sanity check.

Next, let's repeat our previous k-means experiment but using spectral clustering:

1. Run spectral clustering on your SBM with various values of sigma and p_{ij}.

2. Next, generate a concentric spherical dataset by sampling 1000 points from a 3d unit normal distribution. Normalize all points using L2 distance. This should form a hollow sphere. Multiply the first 500 points by 10. This should create concentric spheres. Each sphere will be a separate cluster. Use a Gaussian kernel to generate a graph for these spheres. Run spectral clustering on this.

3. Generate multiple instantiations of Gaussian noise on the concentric spheres and filter them using an ideal low-pass filter.

4. Finally, use your SBM to generate coordinates (as in regular k-means), but generate an adjacency matrix for these points using the Gaussian kernel. Perform spectral clustering on this at various bandwidths.

5. Compare these results to the ones you found in our initial k-means experiment.

Question 6.1. *Compare the labels you obtain on the SBM to the ground truth clusters. At what values of sigma and p_{ij} does spectral clustering start to fail?*

Question 6.2. *How does spectral clustering compare to k-means on the SBM? How does using the SBM adjacency matrix compare to creating one from a Gaussian kernel?*

Question 6.3. *How does spectral clustering compare to k-means on the concentric spheres?*

Question 6.4. *How does the kernel width affect the output of the clustering?*

Question 6.5. *Compare the cluster labels of spectral clustering to the filtered noise on the concentric spheres. What is the frequency content of the label vector? Discuss the connection between Spectral clustering and the Graph Fourier Transform.*

Question 6.6. *In the case that you don't have some pathological geometry (i.e. the concentric spheres), when would you use spectral clustering?*

7 Visualizing with PHATE and tSNE, Comparing to Louvain

We've now performed several clustering methods, but unless you have an extraordinary imagination you may not have a good idea of what these clusters really look like. In this section, we'll use tSNE and PHATE to visualize our clusters. We'll also experiment with one last clustering method — Louvain — and will use tSNE and PHATE visualizations to directly observe the effects of tweaking Louvain's hyperparameters.

1. First, run Louvain on each of the graphs you previously built with the SBM, concentric spheres, and Gaussian Mixture. You can use the `python-louvain` package available from PyPI (e.g. with `pip install python-louvain`). The documentation is available here: `https://python-louvain.readthedocs.io/en/latest/api.html`. For a theoretical overview of Louvain, you can read the paper which introduced it: `https://arxiv.org/abs/0803.0476`.

2. Using tSNE, visualize each of the clusters provided by Louvain. You can use the `scikit-learn` implementation of tSNE. Create at least six visualizations by first varying the parameters of tSNE to find the optimal representation of Louvain's clusters, and then tweaking the parameters of Louvain and rerunning tSNE. You can find a nice explanation of tSNE's parameters here: `https://distill.pub/2016/misread-tsne/`.

3. Now, visualize the Louvain clusters using PHATE. The PHATE package is also available on PyPI (use `pip install phate`), and its usage instructions are here: `https://phate.readthedocs.io/en/stable/`. As before, create at least six visualizations of Louvain's clusters by tweaking the parameters of each algorithm.

4. Finally, resurrect those clusters you previously generated with Spectral Clustering, and use both tSNE and PHATE to visualize them. You may wish to tweak the parameters of tSNE of PHATE to find the optimal visualization.

Question 7.1. *How does Louvain compare to the output of the two previous clustering methods we tried (spectral and regular k-means)?*

Question 7.2. *What is the goal of Louvain?*

Question 7.3. *How does Louvain perform on SBM at various p_{ij} and p_{ii}?*

Question 7.4. *What effects did you observe when varying the parameters of tSNE? Why do you think these changes occurred? Did you find an optimal set of parameters for visualization, or did this depend on the specific clusters?*

Question 7.5. *What effects did you observe when varying the parameters of PHATE? Why do you think these changes occurred? Was there an optimal set of parameters for all visualizations?*

Question 7.6. *What differences did you notice between tSNE and PHATE? Theoretically, what do you think accounts for them?*

Question 7.7. *Why is the Gaussian mixture model a good data set to use to test the clustering algorithm?*

Question 7.8. *What modifications to the mixture model would you make to further test the clustering algorithms?*

Question 7.9. *What are other synthetic data sets you would use to test clustering algorithms?*

8 Retinal Bipolar dataset

In this final installment, we'll turn away from synthetic datasets and enter the much messier world of single-cell RNA sequencing. Our subject is the *Retinal Bipolar* dataset of Shekhar et al. (2015), which includes the markers of 21,000 bipolar cells of the mouse retina. These retinal bipolar cells receive input from the rod and cone photoreceptors of the retina, process it, and pass it onward to the brain. These neurons have been categorized according to their location (whether they receive input from rods or cones), their function (whether their activation increases or decreases with increasing light), and their molecular properties, but these distinctions do not always agree with other. Indeed, significant debate exists in the biological community over *what* qualifies as a class — which provides a perfect entrance for unsupervised learning.

Using a variant of the Louvain method combined with prior knowledge of known clusters, Shekhar et al. identified 15 different classes of retinal bipolar cells. In this section, you'll apply your newly honed clustering skills to verify this classification.

(More about the Retinal Bipolar dataset may be found in Shekhar's paper at `https://www.cell.com/cell/fulltext/S0092-8674(16)31007-8`)

1. Load the Retinal Bipolar dataset and its metadata from the corresponding files `retinal-bipolar-data.pickle` and `retinal-bipolar-metadata.pickle` in the data folder. You should import the `pandas` data storage library to do this, as you can then use the command `pandas.read_pickle('file-path')`.

2. The unaltered dataset has over 15,000 columns corresponding to 15,000 markers for each of the 21,000 cells. To make the computations feasible on your laptop, we'll apply PCA to the columns of the dataset, and project each cell onto just the first 100 PCA components. You may wish to use the command `scprep.reduce.pca` from the `scprep` library.

3. 21,000 cells is still quite a few, so we'll subsample down to 3000 cells. You can use the function `scprep.select.subsample` to randomly select 3000 cells and their corresponding markers from from the dataset and its metadata.

4. Build a graph from the dimensionally-reduced data using an adaptive Gaussian kernel with $k = 10$. You can use the functions you defined in Assignment 1.

5. Cluster the data using k-means, Spectral Clustering and Louvain.

6. Visualize the dimensionally-reduced data using PHATE, coloring the points according to each of the above clusterings. Produce similar visualizations using tSNE. What differences do you notice between the visualization techniques with this dataset?

7. Try varying the kernel parameter k to obtain different cluster assignments. Additionally vary the Louvain parameters. Visualize each of these new clusterings with a recolored PHATE plot.

8. Plot the second and third eigenvectors of the Graph Laplacian. Color the plot by your cluster assignments.

9. Try coloring the PHATE plot using different channels in the (dimensionally-reduced) data. Can you find any channels that seem to correspond to your cluster assignments?

10. Apply an ideal low-pass filter to some of the channels you found above, treating the gene expression as a signal over the graph. Rerun your visualizations using these new, filtered channels to color the PHATE plot. How much has changed? Do any of the denoised channels better represent the true clustering?

11. Binarize the kernel such that it is no longer a weighted graph (i.e., if a value is greater than some threshold, it is 1, otherwise, it is 0). Try rerunning the above clustering methods with this binarized kernel.

12. Finally, compare your clusters to those of Shekhar et al. by coloring your PHATE visualizations with the cluster number provided in the metadata file. Describe any relationships you notice between these clusters and the clusters you've obtained with k-means, Louvain, and Spectral Clustering.

Question 8.1. *How many clusters did you produce? To what extent did different clustering methods parameters affect this number?*

Question 8.2. *In a biological setting, how would you interpret the different clusters? More specifically, how would you interpret any variation you noticed from the 15 clusters described by Shekhar et al?*

Question 8.3. *What did you notice when you plotted the eigenvectors of the Graph Laplacian?*

Question 8.4. *What is the effect of low-pass filtering a feature on the graph? In what context could this be useful for data analysis?*

Question 8.5. *How does binarization of the kernel affect your clustering? Why might this be the case?*

9 Grading Rubric

9.1 Implementation – 20 points

Your implementation will be graded by running your submitted `ps2_functions.py` on several test cases. It is **very important** that you do not change the function input / outputs provided in the skeleton file. If you do, then your functions will not properly handle the test cases and you may receive no credit for the implementation portion of the assignment. Please adequately comment your code so that partial credit may be assigned in the event that your code does not pass all test cases.

9.2 Report – 80 points

You must write a detailed report about the experiments that you have run for this homework, including all plots and visualizations. Your report should address the questions raised here, but can include extra discussion beyond the scope of these questions. Feel free to add any interesting insights that you had or extra experiments / validations you ran.

10 Rules and Guidelines

10.1 Allowable code libraries

You are encouraged to use routines available in `numpy`, `scipy` and `matplotlib` for computation and plotting.

You are required to use `scikit-learn` for tSNE and for the `sklearn=True` implementation of k-means, and both `networkx` and `python-louvain` for Louvain clustering. You may also use the `scprep` library for data preprocessing. Please do not use `scikit-learn` outside of the specific use cases denoted above. Use of other external libraries is prohibited unless otherwise stated, though if there is a library you would like to use you should ask.

10.2 Extensions

Extensions for graded work will not be granted any later than 24 hours before the assigned deadline. Any unexcused late submission will be penalized 10% per day or partial day, with the penalty incrementing at 12:00 midnight each day after the deadline.

10.3 Contesting a grade

Students wishing to contest a grade should submit a request in writing no less than 24 hours after the grade is submitted. Students should first request an explanation of the grade on a particular question from the TF. If, after explanation, a student still wishes to contest, the question will be regraded by the primary instructor.

10.4 Academic Integrity

Please familiarize yourself with the "Definitions of Plagiarism, Cheating and Documentation of Sources" section of the Yale College Undergraduate Regulations. Plagiarism of any kind will not be tolerated in this course. Students are encouraged to discuss general ideas with each other, but must write code and carry out experiments individually. However, the course staff (professor, TA, etc.) is happy to provide assistance in all aspects of the assignment, from coding difficulties to interpretation of experiments. Any plagiarism – whether copying code or sharing experimental analyses – will not be tolerated.

11 Submission Instructions

Each student should submit a zip file titled `[last name]_[first name]_ps2.zip` to Canvas by **Monday, November 2nd, 11:59pm** containing

1. A detailed write-up in pdf form, titled `[last name]_[first name]_ps2_report.pdf`, containing figures, responses to questions, and optionally the code used to produce the figures.
2. A subdirectory titled `code` containing your complete `ps2_functions.py`, and optionally any other code you used in the assignment.

References

[1] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[2] Nathanael Perraudin, Benjamin Ricaud, David Shuman, and Pierre Vandergheynst. Global and local uncertainty principles for signals on graphs. *arXiv preprint arXiv:1603.03030*, 2016.

[3] David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.

[4] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[5] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.