- This homework is due at 11:59pm on February 23, 2022. **Please submit as a PDF** by email to `natalies@cs.unc.edu+comp790`.
- You may write up your solutions in a Jupyter notebook, but please send it as a PDF so that it can be easily read and commented on. :)
- Due to our current COVID situation, if for any reason you need more time on the homework, please try to let me know 24 hours before the deadline.
- There are 3 data files provided for the following questions, including, `Levine_matrix.csv`, `cell_graph.edgelist`, and `population_names_Levine_13dim.txt`. Instructions for how to use these data will be provided in each homework problem.
- You are welcome to any available libraries, like numpy and networkx.
- Please feel free consult with other colleagues, but please write up your own independent solution.
- You are welcome to use Python, Julia, or R here. All hints and sparse code are given for Python.
- You are welcome to write up your assignment using the `HW1_790-166.tex` template, or write up the solutions in the method of your choice.
- This homework is worth 70 points total.

Problem 1

Warm Up - Counting and Adjacency Matrix Math (9 Points Total) Consider an undirected, unweighted graph with N nodes and its corresponding adjacency matrix, $\mathbf{A}$.

1. (5 points) Consider a graph with N nodes and its corresponding adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$. Let $\mathbf{1} \in \mathbb{R}^N$ be the column vector of only 1s (e.g. N 1s.) Using $\mathbf{A}$ and $\mathbf{1}$, write an expression for a vector of node degrees, $\mathbf{k}$, such that the ith entry of $\mathbf{k}$, k_i, represents the number of neighbors of node i. (Hint, you should be writing this expression in terms of $\mathbf{A}$ and $\mathbf{1}$.)

   Sol: The vector of nodes degrees, $\mathbf{k}$ can be expressed as $\mathbf{A} \times \mathbf{1}$.

2. (3 points) Again, using $\mathbf{A}$ and $\mathbf{1}$ write an expression for the total number of edges in the graph.

   Sol: The total number of edges in the graph can be expressed as $\frac{1}{2}\mathbf{1}^T\mathbf{A}\mathbf{1}$. If you have trouble seeing his, we know that $\mathbf{A}\mathbf{1}$ gives us the vector of node degrees. Multiplying this by $\mathbf{1}^T$ will then add all of the node degrees, which we know is twice the number of edges.

3. (1 point) Verify the two expressions that you just defined by plugging the $\mathbf{A}$ defined below. Show that you get $\mathbf{k} = [1, 2, 1]$ and that the number of edges in the graph is 2. You can show this by drawing the graph. You can either verify this by writing out the algebra or by writing code to compute the expressions and showing the output.

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Sol:

Problem 1 continued on next page. . .

```
A = np.matrix([[0,1,0],[1,0,1],[0,1,0]])

Ones = np.matrix([[1],[1],[1]])

#Node degrees
A @ Ones

#total number of edges
0.5 * Ones.T @ A @ Ones
```

Problem 2

Nice Properties of the Graph Laplacian (14 Points Total)

- Here we will walk through computing the Graph Laplacian and exploring nice properties about it that relate to graph structure. You are welcome to paste the code that you write for each of these sub-problems.

Consider the following graph, $\mathcal{G}$ of 6 nodes. This graph has two components. We will see how what happens with our graph Laplacian in this situation.

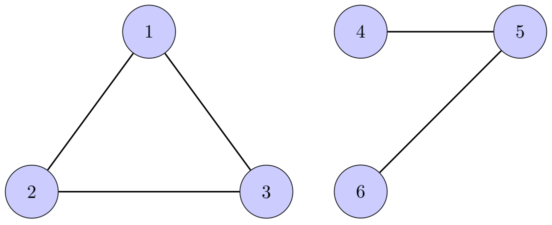


Figure 1: Graph, $\mathcal{G}$ with six nodes and two components.

1. (3 points) Create the adjacency matrix, $\mathbf{A}$, corresponding to $\mathcal{G}$. (Note that node i should correspond to the ith row of $\mathbf{A}$.

   Sol:

```
#adjacency matrix
A = np.matrix([[0,1,1,0,0,0],[1,0,1,0,0,0],[1,1,0,0,0,0],[0,0,0,0,1,0],[0,0,0,1,0,1],[0,0,0,0,1,0]])
```

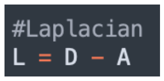
2. (2 points) We will go through some steps to write the code to compute the Graph Laplacian. The first matrix we need to define is the diagonal degree matrix, $\mathbf{D}$. Create the degree matrix ($\mathbf{D}$) for $\mathcal{G}$.

   Sol:

```
#diagonal degree matrix
D = np.zeros((6,6),int)
np.fill_diagonal(D,[2,2,2,1,2,1])
```

3. (2 points) Define the Laplacian matrix, $\mathbf{L}$ as $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

   Sol:



```
#Laplacian
L = D - A
```

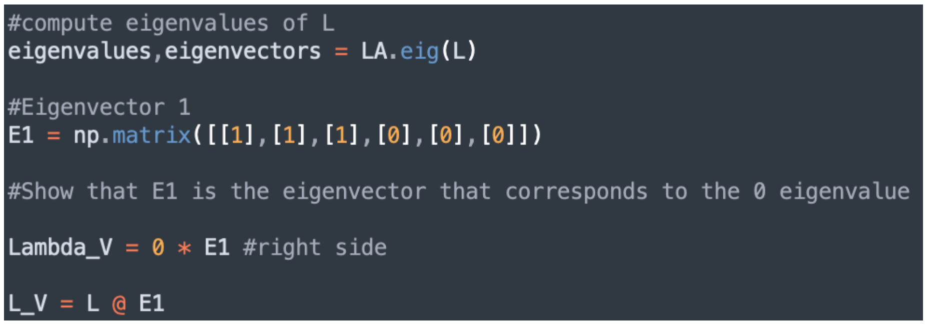
4. (2 points) An eigenvector ($\mathbf{v} \in \mathbb{R}^6$) and its corresponding eigenvalue (λ) of the graph Laplacian, $\mathbf{L}$, must satisfy $\mathbf{L}\mathbf{v} = \lambda\mathbf{v}$. As this graph has two components, we will be able to find two unique eigenvectors ($\mathbf{v}_1$ and $\mathbf{v}_2$) that correspond to the zero eigenvalue, such that $\mathbf{L} \times \mathbf{v}_1 = 0 \times \mathbf{v}_1$ and $\mathbf{L} \times \mathbf{v}_2 = 0 \times \mathbf{v}_2$. Consider $\mathbf{v}_1 = [1, 1, 1, 0, 0, 0]^T$. You will notice that there are 1s in the first three entries, corresponding to the nodes that are together in a component. Check that $\mathbf{v}_1$ does indeed correspond to eigenvalue 0 and satisfies $\mathbf{L} \times \mathbf{v}_1 = 0 \times \mathbf{v}_1$.

   If you are using numpy, you can check by computing,

   ```
   L @ np.matrix([[1],[1],[1],[0],[0],[0]])
   ```

   Discuss your findings.

   Sol:

   ```
   #compute eigenvalues of L
   eigenvalues,eigenvectors = LA.eig(L)

   #Eigenvector 1
   E1 = np.matrix([[1],[1],[1],[0],[0],[0]])

   #Show that E1 is the eigenvector that corresponds to the 0 eigenvalue

   Lambda_V = 0 * E1 #right side

   L_V = L @ E1
   ```

   Yes, this eigenvector satisfies the 0 eigenvalue.

5. (3 points) Given this pattern you just observed with $\mathbf{v}_1$, find $\mathbf{v}_2$. Show that your $\mathbf{v}_2$ satisfies $\mathbf{L} \times \mathbf{v}_2 = 0 \times \mathbf{v}_2$.

   You can fill in the following with your entries of $\mathbf{v}_2$.

   ```
   L @ np.matrix([[ ],[ ],[ ],[ ],[ ],[ ]])
   ```

   Sol:

   ```
   L @ np.matrix([[0],[0],[0],[1],[1],[1]])
   ```

6. (2 points) As a further numerical experiment, find the eigenvalues of $\mathbf{L}$ (you can use `https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html#numpy.linalg.eig`). What do you notice about the smallest two eigenvalues? What about the third smallest eigenvalue?

Problem 3

(**A Single-Cell Graph in the Wild**) (3 points) A graph, $\mathcal{G}_{cell}$ was constructed between 5000 cells contained in `cell_graph.edgelist`. Make sure for the subsequent problems you interpret this as an undirected graph and the adjacency matrix is symmetric.

Problem 3 continued on next page. . .

- (3 points) Use all of your hard work from Problem 2 to compute the graph Laplacian, **L** for $\mathcal{G}_{cell}$. Make a histogram to visualize the distribution of eigenvalues of **L**. What is the smallest eigenvalue of **L**? What are the smallest and second smallest eigenvalues of **L** and how does the observation relate to the structure of the graph?

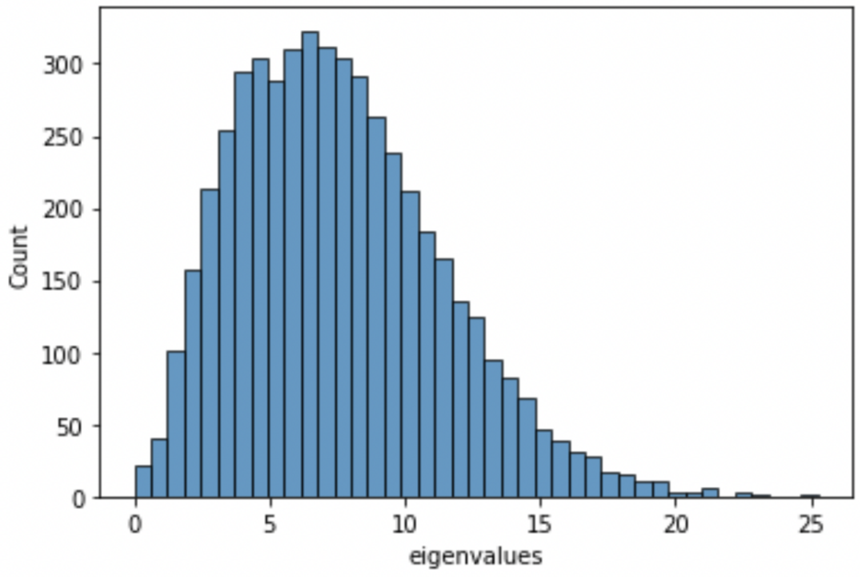
Sol (Chi-Jane's solution):
The smallest eigenvalue is effectively 0. This makes sense, given that the graph has one connected component.

```
fh = open("cell_graph.edgelist", "rb")
G = nx.read_edgelist(fh)
G=G.to_undirected()

A=nx.adjacency_matrix(G).todense()

D=list(G.degree())
D=np.array([tuple_num[1] for tuple_num in D ])
D=np.matrix(np.diag(D))

L=D-A
eigenvalues, eigenvectors = np.linalg.eig(L)
# print(eigenvalues)
```



```
the smallest eigenvalue is  -3.544773791565315e-14
the second smallest eigenvalue is  0.03304627639920743
```

Problem 4

(32 Points Total) (typo in homework template) **Playing with Single Cell Data**

Problem 4 continued on next page. . .

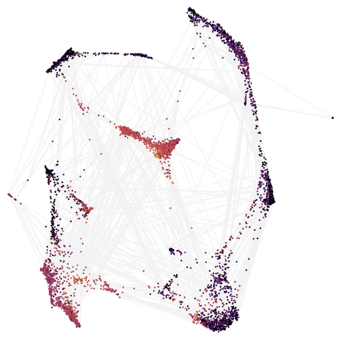


Figure 2: A visualization of the graph in homework problem 3.

We will consider data from a CyTOF experiment obtained from `http://flowrepository.org/id/FR-FCM-ZZPH`. Here, we are considering the expression of 13 different protein markers across a set of cells. It has already been pre-processed for you. From the entire dataset, 5,000 cells were sampled for further analysis. You can use the following accompanying data as follows.

- `Levine_matrix.csv` is the cell $\times$ marker matrix. Note that the last column is labels for the cells. Let's call this matrix, $\mathbf{X}$. **You should not use the last column (named label) for any kind of clustering.** Some cells are not labeled (hence are called NaN).
- `population_names_Levine_13dim.txt` maps the cell labels from the last column of $\mathbf{X}$ (number labels) to biologically-interpretable cell-type names.
- `cell_graph.edgelist` is an edgelist for a between-cell graph. We will call this, $\mathbf{G}$. Note that the nodes in $\mathbf{G}$ correspond to the rows in $\mathbf{X}$. So, node i maps to row i of $\mathbf{X}$, etc.

1) **Clustering on cell $\times$ marker data** (7 points): Use a clustering algorithm of your choice to generate a cell-to-cluster partition for the cells, using the matrix, $\mathbf{X}$. Use normalized mutual information (NMI) to compute overlap between the true and predicted cell labels. **Note that because not all cells are labeled, you can compute this only based on the labeled cells.** Feel free to use an available implementation, such as, `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html`.

Sol (Omar's Solution):

```
X_raw = np.genfromtxt("Levine_matrix.csv", delimiter=",")
Labels = X_raw[1:, -1]
X = X_raw[1:, :-1]

from sklearn.cluster import KMeans
print(f"num of unique labels = {np.unique(Labels[~np.isnan(Labels)]).shape[0]}")
kmeans = KMeans(n_clusters=23, init='k-means++', random_state=0).fit(X)

from sklearn.metrics.cluster import normalized_mutual_info_score
print(f"NMI = {normalized_mutual_info_score(Labels[~np.isnan(Labels)], kmeans.labels_[~np.
    isnan(Labels)])}")
```

```
num of unique labels = 23
NMI = 0.8299804261757121
```

Problem 4 continued on next page. . .

2) **Graph Partitioning** (7 points): Use a graph clustering algorithm to partition **G** into clusters. Similar to part 1 (1) compute NMI between the labels obtained in graph partitioning and the true cell labels.

Sol (Chi-Jane's Solution):

```python
#q2
fh = open("cell_graph.edgelist", "rb")
G = nx.read_edgelist(fh)
G=G.to_undirected()
df_nan_idx=list(set(df.index.tolist())-set(df.index[df['label'].notna()].tolist()))
df_nan_idx = [str(x) for x in df_nan_idx]
G.remove_nodes_from(df_nan_idx)
result= community_louvain.best_partition(G)
result = {int(k):int(v) for k,v in result.items()}
partition_by_graph=[v for k, v in sorted(result.items())]
```

```python
score_by_graph=normalized_mutual_info_score(partition_by_graph, label_idx)
print("NMI by original graph",score_by_graph)
```

3) (3 points) Comment on any observations you observe between the quality of the partitions obtained clustering on **X** in comparison to partitioning **G**. Which approach do you think works better, using the original data, or the graph?

Sol: Most people found that the graph partitioning approach produces results that are a bit better than standard clustering.

4) **Rare Cell-types** (5 points) `Plasmacytoid_DC_cells`, or pDCs (label 21) are a popular rare cell type, meaning many clustering algorithms will not be able to reliably find them. Report the number of distinct clusters where you found pDCs in both the clustering of **X** and in the partitioning of **G**.

Sol: Simply check which clusters contain cells with label 21. Will depend on your clustering results.

5) **Cell Classification** (10 points) Select cells from **X** with the following labels, $\{11, 12, 17, 18\}$ and $\{1, 2, 3\}$. In general, cells with labels $\{11, 12, 17, 18\}$ are T-cells and cells with labels $\{1, 2, 3\}$ are monocytes. Convert this to a binary classification problem by labeling T-cells with 0 and monocytes with 1. Use your favorite classifier to predict the labels of these cells. Use an ROC curve to visualize the performance. If the performance was not good, explain what could have gone wrong. If your performance is very good, can you identify features from **X** that were helpful in predicting labels?
Sol: While this is an easy classification problem, I was looking for the following,

- Splitting cells into training and test sets and only reporting accuracy on the test set
- Use of a standard classifier, such as SVM, random forest, logistic regression, etc.
- An ROC curve to visualize the performance.
- Some notion of feature importance, depending on the classifier that you used. Top features include, CD3, CD33, CD11b.

Problem 5

(30 points total) **node2vec**

We will use the implementation of node2vec available in github, `https://github.com/aditya-grover/node2vec` to create vector representations for each node in **G** encoded in `cell_graph.edgelist`.

1) (**Clustering on Node2Vec Features** (10 points)) First, use default parameters and follow the instructions in the README on the graph in `cell_graph.edgelist`. This will create a 128-dimensional vector for each node. Cluster the nodes based on these vectors and compare to the ground truth labels in the last column of `Levine_matrix.csv` using NMI. Compare your results to Problem 4, question 3. Does an embedding of the graph offer any apparent advantages in classifying cells?

Sol: Here I was looking for the following:

- Did you run node2vec successfully and get an embedding for each node?
- Did you cluster cells according to the embeddings and get an NMI?
- Did you discuss a bit what you found?

2) (**Parameters, part 1** (5 points)) Try a few different values for the number of dimensions `--dimensions`, such that some of them are less than 128, and some of them are more than 128. Cluster cells again with the embeddings obtained in different dimensions. Again, you can compute the NMI between the cluster assignments and the ground truth labels. Comment on some observations, and show a plot of NMI plotted against the number of dimensions used.

Sol: Open-ended, but was generally looking for if you tried a few parameters and talked about or plotted what it did to NMI.

3) (**Parameters, part 2** (5 points)) Recall that the parameters p and q control the 'breadth' vs 'depth of the walk'. Choose one of these parameters to vary, and repeat the previous question using the default 128 dimensions, but varying values for either p or q. Comment on some observations, and show a plot of NMI against p or q (whichever one you chose).

Sol: Open-ended, but was generally looking for if you tried a few different values of p and or q and talked about or plotted what it did to NMI.

4) (**Cell Classification, Part II** (10 points)) Repeat Problem 4, question (5). However, instead of using only $\mathbf{X}$ as the feature matrix, we are going to combine the marker expressions with node2vec features. Let $\mathbf{N}$ be your matrix generated through node2Vec. Create a new matrix called $\mathbf{X} = [\mathbf{X}|\mathbf{N}]$. That is, you will simply concatenate $\mathbf{X}$ and $\mathbf{N}$. Formulate the same classification problem from Problem 4, question (5) to classify T-cells from monocytes. Again, report your ROC curve. Comment on the performance, especially in comparison to the results obtained in Problem 4, question (5).

Sol: Was generally looking for the same things as in question 4, part 5. You should concatenate the node2vec features with the original cell $\times$ protein features to use as input for your classifier.