CSCI 1052 Problem Set 3

January 23, 2024

Submission Instructions

Please upload your solutions by **5pm Friday January 26, 2024.**

- You are encouraged to discuss ideas and work with your classmates. However, you **must acknowledge** your collaborators at the top of each solution on which you collaborated with others and you **must write** your solutions and code independently.
- Your solutions to theory questions must be typeset in LaTeX or markdown. I strongly recommend uploading the source LaTeX (found here) to Overleaf for editing.
- I recommend that you write your solutions to coding question in a Jupyter notebook using Google Colab.
- You should submit your solutions as a **single PDF** via the assignment on Gradescope. You can enroll in the class using the code GPXX7N.
- Once you uploaded your solution, **mark where you answered each part of each question**.

Problem 1: Distance Reconstruction

Suppose you are given all pairwise distances between a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$. You can assume that $d \ll n$. Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the distance matrix with $\mathbf{D}_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$. You would like to recover the location of the original points, at least up to possible rotations and translations which do not change pairwise distances. Assume that $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$.

We can learn the sum of norms $\sum_{i=1}^n \|\mathbf{x}_i\|_2^2$ from $\mathbf{D}$. In particular,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{D}_{i,j} = \sum_i \sum_j \|\mathbf{x}_i\|_2^2 + \|\mathbf{x}_j\|_2^2 - 2\mathbf{x}_i^\top \mathbf{x}_j = \sum_i \left(\sum_j (\|\mathbf{x}_i\|_2^2 + \|\mathbf{x}_j\|_2^2) - 2\mathbf{x}_i^\top \sum_j \mathbf{x}_j \right).$$

By our assumption that the points are centered around the origin i.e., $\sum_j \mathbf{x}_j = \mathbf{0}$, we can conclude that

$$\sum_i \sum_j \mathbf{D}_{i,j} = \sum_i \sum_j \|\mathbf{x}_i\|_2^2 + \|\mathbf{x}_j\|_2^2 = 2n \sum_i \|\mathbf{x}_i\|_2^2.$$

Part 1 (2 points)

Inspired by the above approach, describe an efficient algorithm for learning $\|\mathbf{x}_i\|_2^2$ for each i.

Next, describe an algorithm for recovering a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ which realize the distances in $\mathbf{D}$. **Hint:** This is where you will use the SVD! It might help to prove that $\mathbf{D}$ has rank $\leq d + 2$.

Part 2 (1 point)

Implement your algorithm and run it on the U.S. cities dataset (details below). Note that the distances in the file are unsquared Euclidean distances, so you need to square them to obtain $\mathbf{D}$. Plot your estimated city locations on a 2D plot and label the cities to make it clear how the plot is oriented. Submit these images and your code with the problem set.

I recommend starting with Python code in `UScities.py`. Alternatively, you can directly access the TXT or CSV files.

Problem 2: Opinion Dynamics

We will analyze one model of how opinions evolve in a social network through the lens of the power method. Consider a social network represented by a graph $G = (V, E)$ where V is a set of n nodes and E is a set of edges. Let $\mathbf{z}^{(0)} \in \mathbb{R}^n$ be a vector of randomly initialized starting opinions. If $z_i \geq 0$, we say the ith node has an opinion right of the center and, if $z_i \leq 0$, we say the ith node has an opinion left of the center.

We will use the DeGroot model where the opinions are averaged among neighbors at every step. Formally,

$$z_i^{(t)} = \frac{1}{d_i} \sum_{j:(i,j)\in E} z_j^{(t-1)} \tag{1}$$

where the degree d_i is the number of nodes adjacent to i.

If we run this process for long enough, the opinions will converge to an average opinion. However, if we mean-center and normalize the opinions after each step, we will get interesting behavior. Formally, after the update process in Equation 1, we will mean-center

$$\mathbf{z}^{(t)} = \mathbf{z}^{(t)} - \text{mean}(\mathbf{z}^{(t)})$$

and normalize

$$\mathbf{z}^{(t)} = \mathbf{z}^{(t)} / \|\mathbf{z}^{(t)}\|_2.$$

Part 1 (2 points)

Consider the adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$. We define the adjacency matrix to encode which nodes are connected

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \text{ or } (j,i) \in E \\ 0 & \text{else} \end{cases}$$

and the degree matrix to encode the degree of each node

$$\mathbf{D}_{i,j} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{else} \end{cases}.$$

Write the opinion vector $\mathbf{z}^{(t)}$ after t iterations of DeGroot dynamic updates in terms of matrix multiplication using $\mathbf{A}$ and $\mathbf{D}$ and the starting opinion vector $\mathbf{z}^{(0)}$.

Apply the power method analysis from class to write $\mathbf{z}^{(t)}$ in terms of the eigenvector of some matrix. Remember to make this matrix symmetric so we can apply the power method.

Hint: You will need to use the second eigenvector because the (modified) top eigenvector of this matrix is the constant vector. This is one reason the unmodified DeGroot opinion dynamics converge to a constant opinion.

Part 2 (1 point)

Use the python library `networkx` to implement and visualize $\mathbf{z}^{(t)}$ using the direct computation and the power method computation. I recommend starting with Python code in `opinions.py`.

How do the final opinions $\mathbf{z}^{(t)}$ depend on the starting opinions $\mathbf{z}^{(0)}$? What does your conclusion imply about how people might form opinions?

Problem 3: Spectral Clique Finding

A common tasks in data mining is to identify large *cliques* in a graph. For example, in social networks, large cliques can be indicators of fraudulent accounts or networks of accounts designed to promote certain content. In this problem, we consider a spectral heuristic for finding a large clique based on the top eigenvector of the graph adjacency matrix $\mathbf{A}$:

- Compute the leading eigenvector $\mathbf{v}_1$ of $\mathbf{A}$.
- Let $i_1, \ldots, i_k \in \{1, \ldots, n\}$ be the indices of the k entries in $\mathbf{v}_1$ with largest absolute value.
- Check if nodes $i_1, \ldots, i_k$ form a k-clique.

We will analyze this heuristic on a natural random graph model. Specifically, let G be an Erdos-Renyi random graph: we start with n nodes, and for every pair of nodes (i, j), we add an edge between the pair with probability $p < 1$. To simplify the math, also assume that we add a self-loop at every vertex i with probability p. Then, choose a fixed subset S of k nodes to form a clique. Connect all nodes in S with edges and add self-loops. We will argue that, for sufficiently large k, we can expect the heuristic above to identify the nodes in the clique.

Part 1 (2 points)

Let $\mathbf{A}$ be the adjacency matrix of a random graph generated as above. What is $\mathbb{E}[\mathbf{A}]$? Then write $\mathbb{E}[\mathbf{A}]$ in terms of its eigenvalues and eigenvectors (you don't need to find the exact expression for these!). **Hint:** Argue that, up to multiplying by a constant, any eigenvector $\mathbf{v}$ must have $v[i] = 1$ for all $i \notin S$ and $v[i] = \alpha$ for all $i \in S$, where α is a constant. Then use some high school algebra 2!

Part 2 (1 point)

To prove the algorithm works, it is possible to use a matrix concentration inequality to argue that the top eigenvector of A is close to that of $E[A]$. Instead of doing that, let's verify things experimentally. Generate a graph G according to the prescribed model with $n = 900$, $k = |S| = 30$, and $p = .1$. Compute the top eigenvector of $\mathbf{A}$ and look at its 30 largest entries in magnitude. What fraction of nodes in the clique S are among these 30 entries? Repeat the experiment and report the average fraction recovered.