ELL888 Minor Exam

Navneel Singhal, 2018CS10360

Contents

1 Graph learning setting

You are given $\mathbf{X} \in \mathbb{R}^{n \times d}$ whose rows reside on the vertices of an unknown graph.

$$X = \begin{bmatrix} -\mathbf{x}_1 \in \mathbb{R}^d - \\ -\mathbf{x}_2 \in \mathbb{R}^d - \\ \vdots \\ -\mathbf{x}_n \in \mathbb{R}^d - \end{bmatrix}$$

Graph learning from data simply means finding an edge-weight matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ where its element $w_{ij} = [A]_{ij}$ captures the relation between any two pairs of vertices (i, j), i.e., data points $(\mathbf{x}_i, \mathbf{x}_j)$. We learn the graph weights $\mathbf{w} \in \mathbb{R}^{n(n-1)/2}$ under certain assumptions like smoothness, probabilistic distribution, nearest neighbourhood, etc. The focus here is an undirected graph which implies that the graph matrix is symmetric. Now consider the following problems.

2 Graph learning with missing data

2.1 Problem

$$X = \begin{bmatrix} \mathbf{x}_1 = [x_{11}, x_{12}, \bullet, x_{14}, \ldots, x_{1d}] \\ \mathbf{x}_2 = [x_{21}, \bullet, x_{23}, \bullet, x_{25}, \ldots, x_{1d}] \\ \vdots \\ \mathbf{x_n} = [\bullet, x_{n2}, \bullet, x_{n4}, \ldots, x_{nd}] \end{bmatrix}$$

where some parts of the data are missing at random and $\bullet$ indicates missing data. Explain in detail how we can learn the graph matrix with missing data (in descriptive detail).

2.2 Solution

I will mainly focus on solving this problem under smoothness constraints — i.e., I will assume that we want to learn a graph from a smooth signal.

2.2.1 Optimization problem formulation

Firstly, note that a rudimentary imputation scheme is to do the following: for each feature $i \in \{1, \ldots, d\}$, let D be the distribution of the values for the feature that are not missing. Then, for each x_i such that x_{id} is a missing feature, assign to it a value sampled from D. We could also assign it something different — for instance, the mean of D.

However, this is clearly not good enough. Let's look at the formulation of the optimization problem for the weights $\mathbf{w}$ (with $w(i, j)$ corresponding to the weight of the edge between i and j). It looks something like the following:

$$\min_{w} \sum_{1 \leq i < j \leq n} w(i,j)||\mathbf{x}_i - \mathbf{x}_j||_2^2$$
$$\text{subject to certain constraints on } w$$

Note that we are working under the assumption that the signal is smooth. Considering the fact that in graph regularization, such an "energy" term is used to penalize the objective function, it makes sense to consider the above problem as a minimization problem when we vary the missing data as well, so as to ensure that the graph signal is indeed smooth. That is, the problem transforms to the following:

$$\min_{w, x_{\text{unknown}}} \sum_{1 \leq i < j \leq n} w(i,j)||\mathbf{x}_i - \mathbf{x}_j||_2^2$$
$$\text{subject to certain constraints on } w$$

In a sense, alongside the weights $\mathbf{w}$, we are also learning the missing data under the smoothness assumption.

2.2.2 Solving the optimization problem

For solving this optimization problem, we can follow the following iterative method to solve for optimal w, x_{unknown} given that we have a solver for the original problem as a black-box (in certain special cases, it is possible to solve this directly rather than relying on the following distribution):

1. For each feature $j \in \{1, \ldots, d\}$, compute the multiset D_j of values that appear in $\mathbf{X}$, and for each $\mathbf{x}_i$ such that x_{ij} is missing, assign to x_{ij} a random sample from D_j (or the mean).
2. Until a stopping condition is reached, do the following:
   - Keeping x_{unknown} fixed, solve the optimization problem for $\mathbf{w}$ using the black-box algorithm.
   - Keeping $\mathbf{w}$ fixed, solve the quadratic programming problem for x_{unknown} to get an optimal solution x^*_{unknown}.
3. Return the learned weights $\mathbf{w}$.

Here, the first step tries to learn $\mathbf{w}$ using our prior beliefs about the unknown data, and later on, we try to refine the unknown data by performing some sort of regularization of the graph signal in order to fit it to a more regular signal.

Note that if a fraction k of the data was missing, then in the quadratic programming problem that we need to solve, there would be knd variables, which might be infeasible for a large number of variables. A variant of the above method would be to optimize over these unknown variables in batches (similar to stochastic gradient descent) while keeping the other variables constant, so as to make the size of this problem small enough.

2.2.3 Experiments using some standard techniques

Firstly, a comment on the applicability of this method: note that in the mentioned algorithm, we assumed that we only have access to an oracle that learns weights from a graph (and it is called in the second part of the loop in our algorithm). Hence, this method is applicable for practically any scenario where we want to learn the weight matrix from data, no matter what the constraints on $\mathbf{w}$ that the oracle provides, or the form of the objective function (though if the dependence of the objective function on x is not quadratic, it becomes a different problem — however, the idea remains the same). However, it is not clear if the convergence is guaranteed.

To benchmark it, we shall use the approach in the following paper for the aforementioned oracle:

- Dong et al. "Laplacian Matrix Learning for Smooth Graph Signal Representation". ICASSP, 2015.

Since our objective at hand is to see how well the imputed data represents the original data had it not been missing, we shall take the following approach:

- Take a dataset X of complete data, and remove a certain fraction of values to get a new dataset X'.
- Using the oracle $\mathcal{O}$, compute the graph G, and using our algorithm, compute the graph G' and the dataset X'' consisting of new values.
- Compute the difference between X and X'' on the imputed parts for each feature, and find the average square of the difference between the actual data and imputed data for each feature.
- Compare the F-measures of edge existence.

Since we are working under the assumption that the graph signal is smooth, we will choose the following dataset:

- Randomly generated sensor networks: Since sensor networks aim to measure some physical property, which should be reasonably smooth, these graphs are good candidates for this approach. The graph signal is generated by embedding these sensors in 2D and applying smooth functions to their coordinates.

The implementation uses the base code found at https://github.com/rodrigo-pena/graph-learning, and it was retrieved from https://paperswithcode.com/paper/how-to-learn-a-graph-from-smooth-signals. Some of the metrics that we want are already implemented there, but we used new metrics and they are self-implemented. The code author has released it for public use, and the license is included with the code in the code directory.

For the following experiments, we have $n = 256, d = 4$. In the baseline, we assign the mean of the feature to the missing values. We use $k = 0.05$ and $k = 0.1$ (i.e., 5% and 10% of data missing respectively) in two independent sets of experiments to show the differences between this and the baseline.
Define

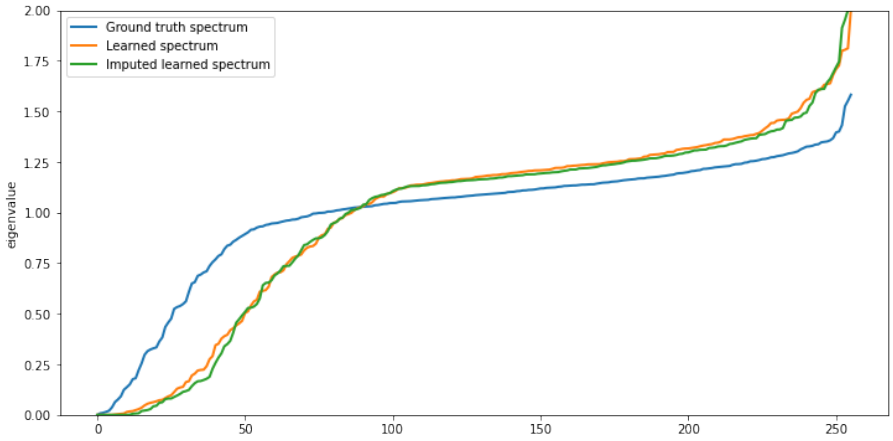
$$\ell_j = \sqrt{\frac{\sum_{i=1}^{n}(x_{ij} - x_{ij,\text{imputed}})^2}{\sum_{i=1}^{n} 1_{x_{ij} \text{ is missing}}}}$$

and

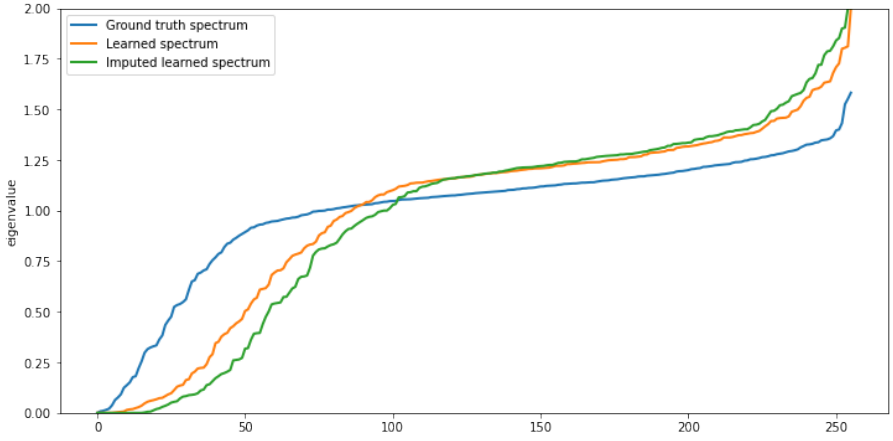
$$\omega = \frac{||W_{full} - W_{imputed}||_F^2}{n \times d}$$

Then we have the following

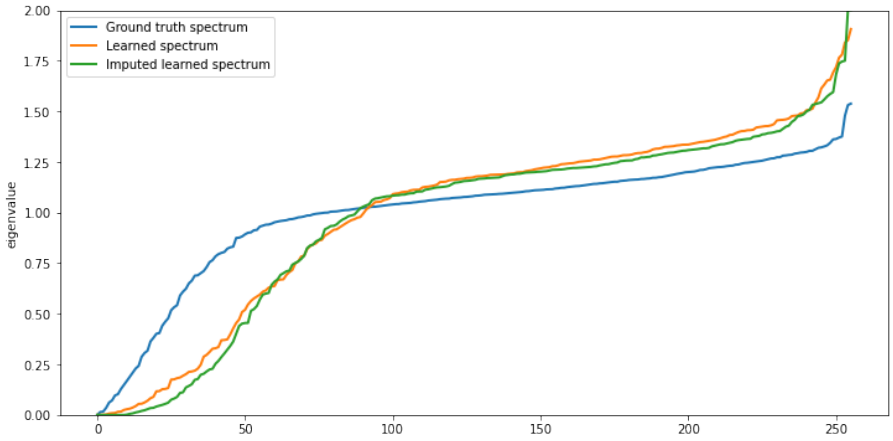
	Baseline	Our algorithm	No missing data
F2-score at $k = 0.05$	0.60	0.67	0.70
F2-score at $k = 0.1$	0.49	0.61	0.69
ω at $k = 0.05$	0.0165	0.0118	—
ω at $k = 0.1$	0.0357	0.0223	—
ℓ_1 at $k = 0.05$	1.156	0.833	—
ℓ_2 at $k = 0.05$	4.033	3.894	—
ℓ_3 at $k = 0.05$	1.875	1.715	—
ℓ_4 at $k = 0.05$	0.971	0.5712	—
ℓ_1 at $k = 0.1$	1.359	0.956	—
ℓ_2 at $k = 0.1$	2.558	2.1034	—
ℓ_3 at $k = 0.1$	1.778	1.745	—
ℓ_4 at $k = 0.1$	0.910	0.770	—



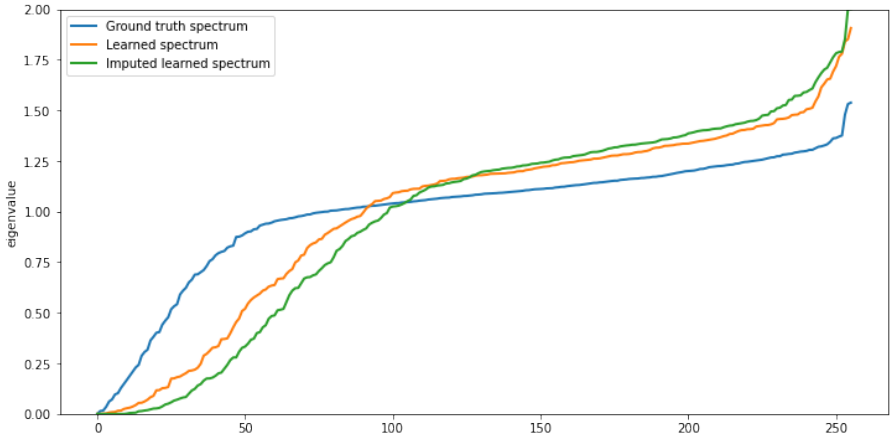
Spectrum for $k = 0.05$



Baseline spectrum for $k = 0.05$

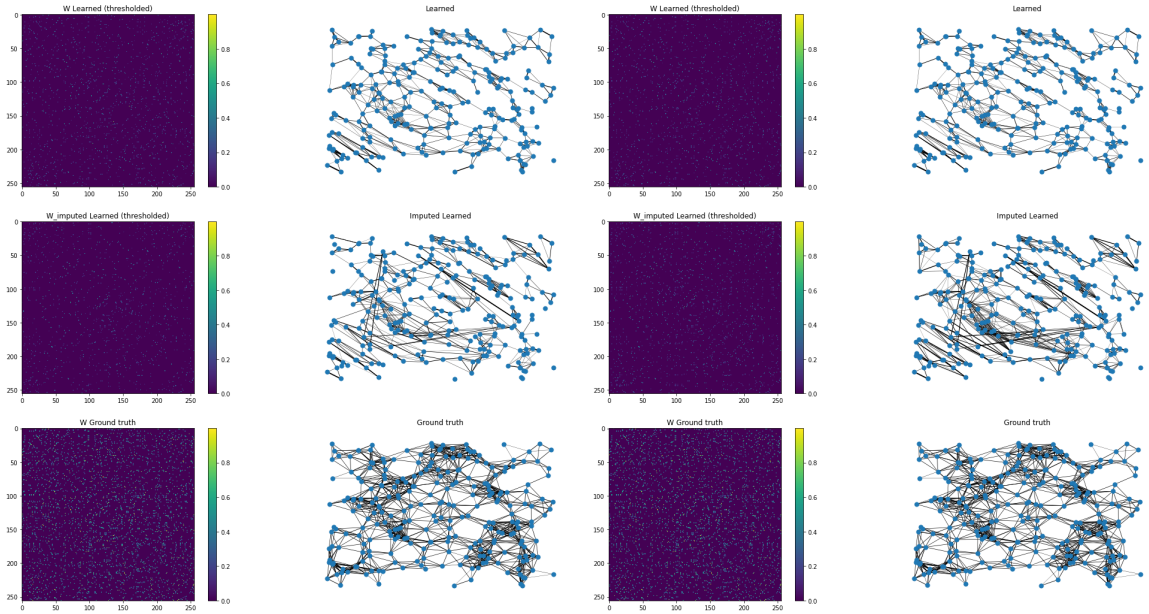


Spectrum for $k = 0.1$



Baseline spectrum for $k = 0.1$

As we can see in the above plots of the eigenvalues, our algorithm matches the eigenvalues fairly well. Now we will show how the learned graphs look like (for $k = 0.1$, baseline and our algorithm respectively):



3 Semi-supervised setting

3.1 Problem

Now consider a semi-supervised graph learning problem. With the data $\mathbf{X}$ sitting at n vertices, we also have the label information for $k < n$ vertices. Simply, we have $\mathbf{X} \in \mathbb{R}^{n \times d}$ with label information for k vertices $\mathbf{y} \in \{0, 1\}^k$. Explain in detail how can we learn the graph matrix integrating the label information. The dataset available is $\left(\{\mathbf{x}_i, y_i\}_{i=1}^k, \{\mathbf{x}_j\}_{j=k+1}^n\right)$.

3.2 Solution

4 Massive data setting

4.1 Problem

Consider a massive data scenario, where n and d both are very large, and typically $n \gg d$. It is not possible to process the entire data every iteration. Explain how we can approach the graph learning problem under such a setting. (Hint: Stochastic gradient descent based approaches try to address such problems, where each iteration is performed considering only a small subset of the dataset, or a *minibatch*).

4.2 Solution

5 Heterogeneous data scenario

5.1 Problem

Consider a heterogeneous data scenario, where the dataset $\mathbf{X}$ associated with vertices may belong to the set of reals $\mathbb{R}$, the set of integers $\mathbb{I}$, and categorical $\mathbb{C} = \{c_1, \ldots, c_k\}$. More concretely, $\mathbf{X} \in \mathbb{R}^{n \times d_r} \times \mathbb{I}^{n \times d_i} \times \mathbb{C}^{n \times d_c}$ and $d = d_r + d_i + d_c$. For example, $\mathbf{x}_1 = [x_{11}, x_{12}, \ldots, x_{1d_r} \in \mathbb{R}, x_{1(d_r+1)}, x_{1(d_r+2)}, \ldots, x_{1(d_r+d_i)} \in \mathbb{I}, x_{1(d_r+d_i+1)}, d_{1(d_r+d_i+2)}, \ldots, x_{1(d_r+d_i+d_c)} \in \mathbb{C}]$. Explain in detail how we can learn a graph matrix with heterogeneous data setting (in descriptive detail).

5.2 Solution