DSCI 553: Foundations and Applications of Data Mining

Fall 2020

Assignment 4

Deadline: 11/20/2020 11:59 PM PST

1. Assignment Overview

The goal of this assignment is to let you be familiar with matrix representation of graph, graph Laplacian, and spectral methods. They are covered in Week7 slides, Page 59.

2. Requirements

2.1 Programming Environment

Python 3.6. You could use spark, numpy, scipy, pandas, sklearn, matplotlib.

You are NOT supposed to use networkx for any purpose in this homework. The goal of the homework is let you familiar with matrix representation of graph, so please build these matrix yourself.

If you need to use additional libraries, you must have approval from TA and declare it in requirements.txt

There will be a 20% penalty if we cannot run your code due to Python version inconsistency or undeclared 3rd party library.

2.2 Write your own code

Do not share code with other students!

For this assignment to be an effective learning experience, you must write your own code! We emphasize this point because you will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code!

TAs will combine all the code we can find from the web (e.g., Github) as well as other students' code from this and other (previous) sections for plagiarism detection. We will report all detected plagiarism.

2.3 What you need to turn in

The submission format will be the same as homework 2. Your submission must be a zip file with name: **firstname_lastname_hw4.zip** (all lowercase). You need to pack the following files in the zip:

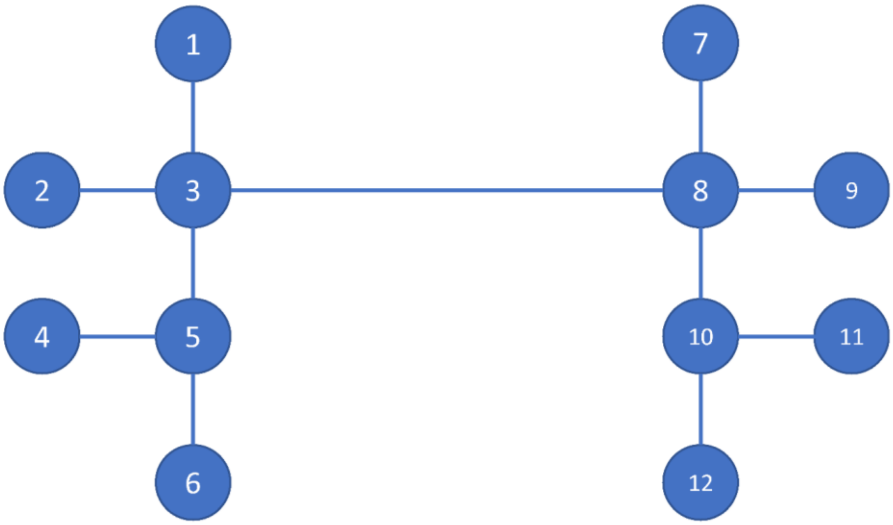
a. Six files, named: (all lowercase)
`firstname_lastname_task1.txt, firstname_lastname_task2.txt,`
`firstname_lastname_task3.py, firstname_lastname_task4.py,`
`firstname_lastname_task5.py, firstname_lastname_task6.py,`

Note the first 2 are txt files, the last 4 are python scripts.

3. Tasks

3.1 Task 1: 2-way spectral graph partition on a small graph (5 pts)

Study slides, especially page 80. Then perform 2-way graph partition on the unweighted, undirected graph below:



Task:
1. Compute its Laplacian matrix (nodes sorted by node index, in increasing order)
2. Calculate the second smallest eigenvector
3. Do graph partition.

Output format: Create `firstname_lastname_task1.txt`
First 12 rows: Laplacian matrix. Numbers are separated by comma.
13th row: second smallest eigenvector. Numbers are separated by comma
14th row: partition result. Written in two brackets, separated by comma (see page 80). Sorted by node index in each bracket and between brackets.

Note:
1. The partition result is obvious in visual, but you should be able to get it from eigenvector.

2. It could be computed by *numpy.linalg.eig*. Please refer to the document for its usage. Note its return values are NOT sorted by eigenvalue.
3. Don't worry about rounding or scaling of eigenvectors.

3.2 Task 2: 4-way spectral graph partition on a small graph (5 pts)

Study slides, especially page 91, 83. If we represent the eigenvectors in the layout of page 83, each row is an embedding of a node. We found similar nodes (eg, node 2 and 5) have similar embedding! So, the complex graph topology has been represented in a simple vector space, and algorithms like clustering, classification could be used on them easily.

Practically, columns corresponding to zero or negative eigenvalue should not be included in the embedding, as they will be a constant for all nodes. (Think about why). The column corresponding to smaller positive eigenvalue contains more useful information, so usually, only the left few columns are used as embedding.

Task:
1. Compute node embedding for the same graph as in task 1. Only keep the first 3 dimensions.
2. Use k-means on the embedding. Nodes belong to the same cluster in the embedding space also belong to the same cluster in the original graph.

Output format: Create `firstname_lastname_task2.txt`
First 12 rows: Node embedding. Numbers are separated by comma. 3 numbers in each row. (nodes sorted by node index, in increasing order)
13th-16th row: Clustering centers. Numbers are separated by comma. 3 numbers in each row. Row order doesn't matter
17th row: Partition result. Written in 4 brackets, separated by comma (see page 80). Sorted by node index in each bracket and between brackets.

Note:
1. The partition result is obvious in visual, but you should be able to get it from k-means.
2. You could use *sklearn.cluster.KMeans* or write your own, in all tasks.

3.3 Task 3: k-way spectral graph partition on a large graph (15pts)

Now, let's apply the spectral graph partition algorithm on some real-world data, for example, a graph of email communication. (https://snap.stanford.edu/data/email-Eu-core.html). Open the link for more information about the dataset.

The dataset is included in the *data* folder. It includes two files, *email-Eu-core.txt* contains edge information, each line represents an edge between two nodes. File *email-*

Eu-core-department-labels.txt contains node label information, each line specifies the label of a node, 42 labels in total. They are ground truth cluster labels.

Task: Clustering a large graph into k clusters.

Input format:
python firstname_lastname_task3.py <edge_filename> <output_filename> <k>

edge_filename: Path to edge file, e.g.: *data/email-Eu-core.txt*

output_filename: Path to the the output file

k: Number of clusters, e.g.: 42

Output format: Similar to *email-Eu-core-department-labels.txt*, the second column should be predicted cluster label. The absolute value of the labels doesn't matter. See the note #5 below.

Note:

1. You should read the input and output path from the command line argument. We may test your code on a different dataset.
2. The dataset is directed, but this simple spectral method requires the graph to be undirected! Add edge in both directions when you construct the graph. Spectral methods for directed graph is possible but much more complicated.
3. If computed eigen value/vector are complex number, you should convert it to real number via *np.real*.
4. You need to decide which eigenvectors should be included in the node embedding. Bigger embedding is not necessarily better, it may be noisier. Hint: Those corresponding to smaller positive eigenvalue contains more useful information.
5. Clustering is unsupervised learning: you should NOT read ground truth cluster label in your submitted program!
6. The evaluation should be done in another program (no need to submit). Quality of clustering could be evaluated by Adjusted Rand index (*sklearn.metrics.adjusted_rand_score)*. This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way. Its range is [-1, 1]. A score of 1 means perfect match, a positive score means two clustering results are similar, a 0 means they are irrelevant.

Grading:

1. 0 point if your spectral graph partition is provided by a library. But you could use *sklearn* to perform k-means, and/or other auxiliary steps.

2. As long as your Adjusted Rand index >= 0.06, you get full points. Since k-means algorithm has randomness in nature, we will run your program multiple times and take the best result.

2.4 Task 4: Node classification based on spectral embedding (10pts)

Task: Perform node classification based on learned spectral embedding. A sample train/test split is provided as *labels_train.csv/labels_test.csv*

Input format:
python firstname_lastname_task4.py <edge_filename> <label_train_filename> <label_test_filename> <output_filename>

edge_filename: Path to edge file, e.g.: *data/email-Eu-core.txt*

label_train_filename: Path to train label file. e.g.: *data/labels_train.csv*

label_test_filename: Path to test label file. e.g.: *data/labels_test.csv*

output_filename: Path to the output file

In *labels_test.csv*, the second column is filled with 0. If you want to evaluate your model's performance, you need to compare your prediction with *labels_test_truth.csv*

Output format: Similar to *labels_test_truth.csv*, the second column should be your model's prediction. This time the absolute value of the labels DO matter. You can't permutate labels.

Note:

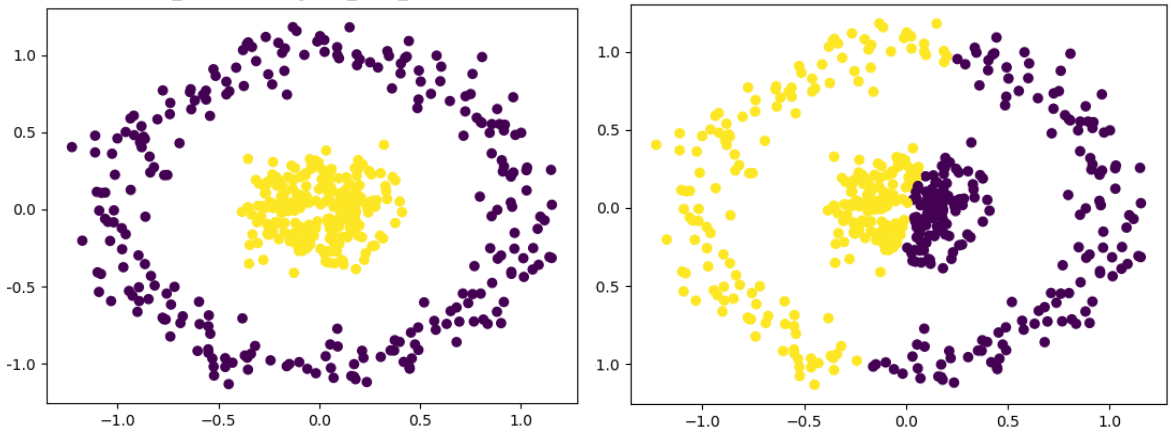
1. Same as task 3, Note 1-2
2. We have performed node classification in homework3, that is based on the external features on each node, like tweets they tweeted. Here the node classification is solely based on graph topology represented in spectral embedding.
3. You could compute the spectral embedding for all nodes at once, then train the classifier only with nodes in the training set. You don't have to (and actually impossible to) compute embedding for training nodes and test nodes separately. Graph is not like other data that each entry could be processed independently, graph should considered as an indivisible object, that removing one node will change embedding for all other nodes too.
4. You could use k-nearest neighbors classifier. (*sklearn.neighbors.KNeighborsClassifier*) Other classifiers like logistic regression are also fine.

5. Think about the difference between supervised and unsupervised learning, k-nearest neighbors, and k-means before proceeds.
6. Node classification is supervised learning: you could read ground truth label information from the training set, but NOT from test set.
7. The optimal embedding dimension may be different for classification task and clustering task.
8. The evaluation should be done in another program (no need to submit). You could use accuracy (sklearn.metrics.accuracy_score).

Grading:

1. 0 point if your spectral embedding is provided by a library. But you could use *sklearn*'s KNN or other classifiers, and/or other auxiliary steps.
2. As long as your Test Accuracy >= 0.6, you get full points.

3.5 Task 5: Spectral graph partition on non-graph data (15pts)



See the above example of non-spherical data (provide in *data/non-spherical.json*). K-means method couldn't partition them correctly.
However, if construct a graph based on these data, and partition the graph nodes using spectral methods, they could be partitioned correctly.
The idea is constructing a k-nearest neighbor graph: Each data point corresponding to a node and is connected to k closest nodes measured in Euclidean distance. Construct an adjacency matrix based on the graph. Then perform spectral graph partition. For nodes in the same graph partition, their original data points should be in the same partition.

Task: Perform spectral graph partition on non-spherical data.

Input format:
python firstname_lastname_task5.py <data_filename> <output_filename>

edge_filename: Path to edge file, e.g.: *data/non-spherical.json*

output_filename: Path to output file. It will have *.png* as its suffix.

Output format:

1. A plot (in png format) showing the data points and coloring the clustering result. Other settings like size, color used, labels don't matter. Please refer to the "Correct partition" figure showing above. Make sure your program saves a figure rather than displaying it (via *plt.savefig*)

Note:

1. You should read the input and output path from the command line argument. We may test your code on a different dataset, but they will always contain 2 clusters.
2. The constructed adjacency matrix should satisfy the property listed in slide page 75: Zero on diagonal and symmetric.
3. You could find k-nearest nodes using a double loop, or use *sklearn.neighbors.NearestNeighbors* or *sklearn.neighbors.kneighbors_graph*. Think about the connection between *NearestNeighbors* and KNeighborsClassifier
4. The optimal embedding dimension may be different in this task. Since we are doing 2-way clustering, maybe we don't need to construct embedding at all? (Recall task 1) If you go this path, you couldn't simply set the threshold to 0. Check the detail from the announcement in slack #homework4 channel.

3.6 Task 6: Identify important nodes in a graph via page rank (10pts)

Page rank is initially used to find important webpages, but it generalizes to find important nodes in any type of graph, for example, important persons in an email communication graph. (reusing the dataset in task 3)
The same hypothesis in page rank could be used in email communication: if a lot of emails are sending to a person, then that person is very likely to be important. Getting an email from an important person makes you more likely to be an important person.

Task: Find the 20 most important nodes in a graph via page rank. Use random teleport with $\beta = 0.8$ and always teleport for dead ends.

Input format:

python firstname_lastname_task6.py <edge_filename> <output_filename>

edge_filename: Path to edge file, e.g.: *data/email-Eu-core.txt*

output_filename: Path to output file

Output format: A text file contains node index of the 20 most important nodes, sorted by their importance in decreasing order, one node index per line.

Note:

1. You should read the input and output path from the command line argument. We may test your code on a different dataset.
2. You MUST ignore self-loop. In general, adding self-loop or not are both fine in page rank. But here, the graph is not strongly connected, couldn't use page rank algorithm. But by ignoring self-loop, components will get connected via random teleport.
3. The graph here is very small, we could compute eigenvectors directly, and don't have to use a random walk-based method. The web graph is so large that it's impractical to perform eigen decomposition, so a random walk-based method is necessary.
4. The dataset is directed, you should construct a directed graph, and the transition matrix is NOT symmetry.

Grading:

1. 0 point if your page rank is provided by a library.
2. Your answer should exactly match the solution. If not, 5 points deducted, and we will inspect your code to determine the remaining points. Hint: The two most important nodes are node #160 and #62.

4. Grading Criteria

The perfect score for this assignment is 60 points.

Assignment Submission Policy

Homework assignments are due at 11:59 pm on the due date and should be submitted in Blackboard. Every student has ***FIVE free late days*** for the homework assignments. You can use these five days for any reason separately or together to avoid the late penalty. There will be no other extensions for any reason. You cannot use the free late days after the last day of the class. You can submit homework up to one week late, but you will ***lose 20% of the possible points*** for the assignment. After one week, the assignment cannot be submitted.

(% penalty = % penalty of possible points you get)

1. You can use your free 5-day extension separately or together.
2. If we cannot run your programs with the command we specified, there will be 80% penalty.
3. If your program cannot run with the required Python/Spark versions, there will be 20% penalty.
4. If our grading program cannot find a specified tag, there will be no point for this question.
5. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
6. If the header of the output file is missing, there will be 10% penalty.
7. We can regrade on your assignments within seven days once the scores are released. No argue after one week. There will be 20% penalty if our grading is correct.

8. There will be 20% penalty for late submission within a week and no point after a week.
9. There will be no point if the total execution time exceeds 15 minutes.