E536: Graph Analytics

Homework 2

Post date: Tuesday, Feb 6.

Due date: Monday, Feb 12. 11:59pm Eastern time

Total Points: 80

For all of the problems below you will be using the Cora data. The data is available as part of the homework. You can use all example codes provided on Canvas.

Q 1: Testing parameters in **DeepWalk** [Total 20 points]

In this problem you will experiment with different parameters as specified below and their effect on (a) node classification, (b) link-prediction, (c) clustering modularity score calculations. While testing a single parameter in DeepWalk, use the parameter values as given below while keeping other parameters at their default values (as provided in the sample notebook). Recall that after we obtain an embedding, we need to train a logistic regression to classify nodes or predict links. For these classification tasks, please use 70% data for training and 30% data for the testing. For the clustering task, use 10 clusters with the Kmeans clustering algorithm. For node classification and link prediction tasks just report the accuracy. For the clustering task just report the modularity score.

1. parameter **walk lengths:** 1, 5, 10, 20, and 50. With embeddings obtained with these parameters, perform the following tasks and create a table with three rows and 5 columns (one column for each parameter).
   (a) node classification [2 points]
   (b) link-prediction [2 points]
   (c) clustering modularity scores [2 points]
2. parameter **number of walks:** 1, 10, 40, 80, and 200. With embeddings obtained with these parameters, perform the following tasks and create a table with three rows and 5 columns (one column for each parameter).
   (a) node classification [2 points]
   (b) link-prediction [2 points]
   (c) clustering modularity scores [2 points]
3. parameter **embedding dimension:** 2, 10, 50, 120, and 500. With embeddings obtained with these parameters, perform the following tasks and create a table with three rows and 5 columns (one column for each parameter).
   (a) node classification [2 points]
   (b) link-prediction [2 points]
   (c) clustering modularity scores [2 points]

For these tasks, you can use the Jupyter notebook provided with the HW. Tabulate your results (the accuracy and modularity scores). You will report three tables, one for each set of parameters.

Please explain which parameters are more important for different tasks. For example, which parameter/parameters are more important for different tasks should be enough.

Also mention the trend (positive correlation or negative correlation) observed. [2 points]

Q 2: Testing parameters in **Node2Vec** [Total 20 points]

In this problem you will experiment with different parameters as specified below and their effect on (a) node classification, (b) link-prediction, (c) clustering modularity score calculations. While testing a single parameter, use the parameter values as given below while keeping other parameters at their default values. Please use the same training and testing ratio as mentioned in Q1. Similar to Q1, report test accuracy for node classification and link prediction and the modularity score for the clustering task.

1. parameter **p:** 0.01, 0.1, 0.25, 0.5, and 1. Perform -
   (a) node classification [3 points]
   (b) link-prediction [3 points]
   (c) clustering modularity scores [3 points]
2. parameter **q:** 0.1, 0.5, 1, 2, and 4. Perform -
   (a) node classification [3 points]
   (b) link-prediction [3 points]
   (c) clustering modularity scores [3 points]

Tabulate your results (the accuracy and modularity scores). You will report two tables, one for each set of parameters.

Please explain which parameters are more important for different tasks. [2 points]

Q 3: New embedding for shortcut random walks [Total 20 points]

Develop a new random walk strategy as follows. Suppose we are walking from a node u while generating random walks. Repeat the following steps l times where l is the length of a random path.

step 1. with probability p we randomly pick a neighbor v of u and add v to the path.

step 2. with probability $(1-p)$ we randomly pick any non-neighbor v of u and add v to the path.

step 3. Set v as the new current vertex and repeat step 1 and 2.

The probabilities (p) is given to make a choice between randomly picking a neighbor or randomly picking a non-neighbor. Once this choice is made using the probabilities, we just pick a vertex at random from the target list which can either be a list of neighboring vertices or a list of non-neighboring vertices. Here $0 \leq p \leq 1$. If p is set to 1, we will get DeepWalk embedding. If p is set to 0, we get a random collection of nodes.
You have to do the following:

1. Write a function named `shortcut_walk`. You can follow the structure of the function `deepwalk_walk` shown in the class. The function `shortcut_walk` should implement the walking logic discussed above. [6 points]

2. parameter p: 0.01, 0.1, 0.25, 0.5, and 1. For these five values of the parameter p and your algorithm `shortcut_walk` perform the same analysis (as done in Q1 done for DeepWalk):

   (a) node classification [3 points]

   (b) link-prediction [3 points]

   (c) clustering modularity scores [3 points]

   NOTE: as before you will keep other parameters (walk length, number of walks, etc) with default values. (mention these values in your report). Use the experimental setting considered in Q1. You will obtain a table with three rows and five columns.

3. Explain your observations for different values of p. For example, should we expect better accuracy with a small number of shortcuts? [5 points]

Q 4: New embedding for long-distance similarity [Total 20 points]

Develop a new random walk strategy as follow. Suppose we are walking from a node u while generating random walks. Repeat the following steps l times where l is the length of a random path.

step 1. with probability p we randomly pick a neighbor v of u and add v to the path.

step 2. with probability $(1 - p)$ we randomly pick a vertex v such that u and v have "similar" local clustering coefficient. For example, two vertices have similar clustering coefficient if the difference between their clustering coefficients is less than 0.1.

step 3. Set v as the new current vertex and repeat step 1 and 2.

Here $0 \leq p \leq 1$. If p is set to 1, we will get DeepWalk embedding.
You have to do the following:

1. Write a function named `similarity_walk` in the walker.py file in the GraphEmbedding library. You can follow the structure of the function `deepwalk_walk` in the same file. The function `similarity_walk` should implement the walking logic discussed above. You will need to find the clustering coefficient of all vertices before you generate random walks. [6 points]

2. parameter p: 1, 0.8, 0.5, 0.2, and 0. For these five values of the parameter p and your algorithm `similarity_walk` perform the same analysis (as done in Q3 above for `shortcut_walk`):

   (a) node classification [3 points]

   (b) link-prediction [3 points]

   (c) clustering modularity scores [3 points]

   NOTE: As with problem 3, the probabilities (p) is given to make a choice between randomly picking a neighbor or randomly picking another vertex with similar clustering coefficient. As before you will keep other parameters (walk length, number of walks, etc) with default values. (mention these values in your report)

3. Explain your observations for different values of p. For example, should we expect better accuracy with a small number of long-distance similarity walks? [5 points]

Submission Instructions

Since this assignment involves lot of tabulated results, it is better to have some organization. Please submit your code that can be executed. Submit all files (except the data file) needed to execute your notebook. We also ask you to submit separate notebooks for different questions. Write your explanations in cells inside your notebooks. Notebooks must be executable with your helper functions.

1. file `hw02-q01.ipynb` as a source code file of your Jupyter Notebook for question 1.
2. file `hw02-q02.ipynb` as a source code file of your Jupyter Notebook for question 2.
3. file `hw02-q03.ipynb` as a source code file of your Jupyter Notebook for question 3.
4. file `hw02-q04.ipynb` as a source code file of your Jupyter Notebook for question 4.
5. any other python code files given to you, as part of this HW, that you have changed.
6. any other python code files that you may have written for any of the questions.

Since there is lot of common functionality in all four questions, you could put all of that in a single python module in a file `hw02-helper.py` and use that (and do not forget to submit this file too).

And as usual:
You are welcome to discuss HW questions with others. I am always happy to help you. Please don't copy answers or code from the internet or from another person.

Late policy: For every late day you will lose 20% of the assigned grade for this HW. That means, if you submit 5 days after the deadline, you will not get any point. Only exceptions are medical or family emergencies. In emergency, you can submit after the due date without any penalty. Please ask if you have questions.