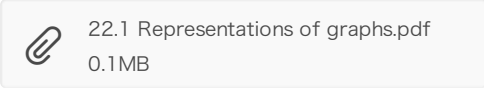
AI训练营-数据挖掘-第二周

Day 1

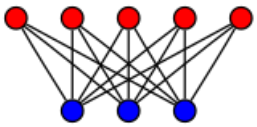
1. Learn Graph Data Structure
   a. Graph Data Structure Basics (对英文不习惯的同学可以看《算法导论》第22章第一节)



   b.
      i. How to store a graph in Python or C++
      ii. The limitation of storing a graph in a matrix (two dimensional array)
   c. Basic graph algorithm
      i. DFS (https://en.wikipedia.org/wiki/Depth-first_search)
      ii. BFS (https://en.wikipedia.org/wiki/Breadth-first_search)
   d. Bipartite graph (https://en.wikipedia.org/wiki/Bipartite_graph)
      i. Coding exercise at home: store a bipartite graph in python

Now, let's think about how to do recommendation by utilizing the graph structure. There are two major questions we need to answer here: (1) what is the graph? and (2) how to do recommendation. During all the following days, we will try to give answers to these two questions and implement our own graph based recommendation algorithms.

What is the graph?
Generally speaking, in the recommendation scenarios, we have two entities: users and items. Users tend to engage with items and our task is usually defined as *given a user, recommend top K items to this user.*
Obviously, this naturally fits in the graph. Here is an simple bipartite graph illustration.



In the movie recommendation task, we can represent users as the red nodes and movies as the blue nodes in above graph. If one user watches/likes/comments a movie, we put an edge between the user (red node) and the movie (blue node). Now, we have the movie recommendation graph.

Another example would be in the online shopping setting, we view users as the red nodes and products as the blue nodes. If one user buys a product, we add an edge into the graph. This will give us the production recommendation graph.

How do we do recommendation by using the graph?
Graphs store all the relations between users and items, which are very valuable for recommendation. Think about this simple case (feel free to draw this sample graph on paper): Jack bought Product A and B. Peter bought Product B, C, D. Because of B, C and D is accessible to Jack and we can suggest C and D to Jack as our recommendation. This is a very simple toy example to give you a sense how a simple graph based recommendation works. Along the course, we will learn some basic graph algorithms to make good recommendations.

Don't reinvent the wheel! There are lots of libraries to efficiently store the graph and run basic graph algorithms. In the following days, we are going to learn **SNAP** (http://snap.stanford.edu/) and build our own recommendation algorithms based on SNAP.

2. Use the pre-installed SNAP on Kesci.com or install SNAP on your own laptop.
   a. Install instruction: http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part2.pdf
3. SNAP has the standardized naming convention. You may feel strange at the beginning, no worries. Keep learning and you will find it is very conveinient.
   a. Reading (mandatory): http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part2.pdf (Don't need to finish all the reading on Day 1 but have to finish it in Day 2)
   b. C++ API: http://snap.stanford.edu/snap/doc/snapuser-ref/
   c. Python API: https://snap.stanford.edu/snappy/doc/index.html

Day 2

1. Continue learning SNAP.
   a. Reading (mandatory): http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part2.pdf (Finish it in Day 2)
   b. More exmaple about SNAP (optional): http://snap.stanford.edu/class/cs224w-2017/recitation/SNAP.PY_Recitation.pdf
   c. C++ API: http://snap.stanford.edu/snap/doc/snapuser-ref/
   d. Python API: https://snap.stanford.edu/snappy/doc/index.html
   e. Try the examples on the slides
      i. Graph Creation.
      ii. Graph saving and loading from text file
      iii. Preprocessing your previous dataset and load it by SNAP. (If the data is too big, do sampling)
      iv. C++ instruction is here: http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part5.pdf

Day 3

1. Plot degree disbribution by using SNAP.
   a. Reading (at slide 26): http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part2.pdf
   b. Reading (at slide 9): http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part3.pdf
2. Random walk on graph.
   a. There are multiple descriptions of random walk algorithms and lots of theories behind them. Here is a simple description to help you understand and implement it: every time, when you want to start the random walk, you specify (1) how many steps (say N steps in total) you want to walk, which is the stopping criteria; (2) and the starting point. Each iteration, we flip a coin at probability p, to decide whether to continue the random walk or to restart the random walk (go back to the staring point). When you choose to continue the random walk, you uniformly (if you have weights on the edges, you can do weighted sampling) randomly select a neighbor from current node. You jump to the selected neighbor and you finish one step out of the total N steps. Then we just repeat the above process. While we are doing the random walk, we count the number of times we jump to the same node and at the end, the most frequently visit nodes will be our recommendation.

      Here is a formal description of Basic Random Walk Algorithm.

      i. Reading: https://arxiv.org/pdf/1711.07601.pdf There are lots of information on this paper, you don't need to read all of them. (If you have time, please enjoy the entire paper.) Go to Section 3.1 and try to understand Algorithm 1. Then you can stop

reading it.

3. Use the same data set you picked in Week 1 and pre-process it and load it into a bipartite graph by SNAP. For example, in the movie recommendation setting, you will have all the movies on one side and all users on the other side.
   a. C++ API: https://snap.stanford.edu/snap/doc/snapuser-ref/d8/d97/classTBPGraph.html

Day 4

1. Implement the random walk algorithm on the graph you created yesterday.
2. Test the recommendation by using the random walk algorithm.
   a. Given a user, recommend top K items.
   b. Given a user, recommend top K users.
   c. Given an item, recommend top K items.
   d. Choose whatever setting and take a look of your results. Just look at the results of some cases and to see whether they make sense to you.
3. Fully evaluate the recommendation results by computing precision/recall and all other metrics you have in Week 1.
4. Compare the recommendation results with methods you developed in Week 1.

Day 5

1. Work on your own project/data by using various recommendation algorithms.
   a. Hint: think about how to improve the basic random walk algorithm; what if the graph is not a bipartite graph; what if you have extra features besides users and items, etc.

Day 6

1. Data profiling on TAL data.
2. Play with your refactored, clean recommendation algorithms on TAL data.
3. Benchmark both the recommendation accuracy and running time.
4. Prepare the report to summarize what you learned in this week
   a. Data statistics (feature distributions, etc.)
   b. Recommendation performance (recall, RMSE, MAE, etc)
   c. Running time analysis
   d. **Any interesting stuff you find**

Prepare the presentation slides