Assignment 3: Query-Centric PageRank

Due: Nov 18th (Friday)

PageRannk is one of the most well known algorithms in information retrieval. Based on Wikipedia definition, *PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.*

In this assignment, we will investigate a different PageRank algorithm, personalized PageRank, a.k.a. query-centric PageRank, to address user information need when rank the documents or objects. We employ an important domain – Scholarly Search, for this assignment.

Task 1: Rank the authors in scholarly database

As Google Scholar shows, scholarly search can search and rank the publications based on user queries. For publication search, author ranking can be important (which can be used as a kind of paper prior). For this task, we will use PageRank to calculate the author importance. In the graph we provided, each node is an author, and each edge is a citation, i.e., $A_i \rightarrow A_j$. When an author accumulate more citations, this author can be more important, and his/her paper can be more important. We, then, will use PageRank to calculate the author importance given a graph.

$$\pi(v)^{i+1} = d\left(\sum_{u=1}^{d_{in}(v)} P(v|u)\,\pi^{i}(u)\right) + \frac{1-d}{N}$$

The PageRank theory holds that an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d. Various studies have tested different damping factors (d), but it is generally assumed that the damping factor will be set around 0.85. The damping factor is subtracted from 1 (and in some variations of the algorithm, the result is divided by the number of documents (N) in the collection) and this term is then added to the product of the damping factor and the sum of the incoming PageRank scores. In this project, each node, e.g., v and u, is an author, and each edge translation probability is calculated by $P(v|u)$, which can be used to characterize the probability that author u cite author v in the scholarly database (or to say the chance that author u vote author v is important).

After subtracting from 1

For implementation, you can use java JUNG package (http://jung.sourceforge.net/doc/api/index.html) and the PageRank algorithm is hosted in a java class: edu.uci.ics.jung.algorithms.scoring.PageRank<V,E>

For this task, you will need to submit the java code via: AuthorRank.java
And, please list the top 10 ranked authors here:

Task 2: Rank the authors given a query

While author ranking in the first task can be potentially important for scholarly search, one additional problem still exists – which author is more important in a domain? Or, given a query, which author(s) can be more important than others?

In order to address this problem, we will need to use a different algorithm – PageRank with prior:

$$\pi(v)^{i+1} = \left(1 - \beta_b\right)\left(\sum_{u=1}^{d_{in}(v)} P(v|u)\,\pi^i(u)\right) + \beta_b p_{prior}(v)$$

Unlike PageRank, for this formula a number of nodes can be used as the "seeds" on the graph, and they are more important to the target user given an information need (e.g., a query). In above formula, it is characterized as the prior probability $p_{prior}(v)$. If the node *v* is not a seed, the prior can be 0. So, for this PageRank function, the importance of the node can be decided by two different components, 1. The random walk probability from node *u* to *v* $\sum_{u=1}^{d_{in}(v)} P(v|u)\,\pi^i(u)$, and 2. The prior probability $p_{prior}(v)$. Note that, when a node is not directly or indirectly linked to the seed nodes, this node importance can be 0 or very low (even though this node has a large number of incoming edges). This is significantly different from PageRank.

Then, the next question will be how to generate node prior $p_{prior}(v)$. For this task, we will use Lucene. First, you will need to download the Lucene index we generated. Each Lucene document is a scholarly paper, which has four different fields: Paper ID, Paper Content (text), Author ID (mapping to the node ID on the graph) and Author Name.

When user input a query, we will:

1. Send the query to the Lucene index (paper content field) and find the top ranked 300 publications via BM25 algorithm.
2. For each top ranked publication, we will extract the ranking score, and add the ranking score to the target author prior probability. For instance, if an author published 3 papers in the top 300 publication list and the ranking scores are 0.1, 0.05, and 0.05, the author prior can be 0.1+0.05+0.05 = 0.2
3. Then, we need to normalize the author prior. Because, in step 2, the author prior is not probability yet. You can use:

$$p_{prior}(v) = \frac{p_{prior}(v)}{\sum p_{prior}(u)}$$

4. Next, we will rank the authors with the priors generated in step 3. The PageRank with prior algorithm can be found in edu.uci.ics.jung.algorithms.scoring.PageRankWithPriors<V,E>

For this task, you will need to submit the java code via: AuthorRankwithQuery.java
Please list the top 10 ranked authors given query "Data Mining" here:

Please list the top 10 ranked authors given query "Information Retrieval" here: