Programming Assignment 2

- Term $i$ appears in $n_i$ articles within the corpus
- Inverted Document Frequency
  - $IDF_i = \log_e(\frac{N}{n_i})$
  - where $N$ is the total number of articles
- TF-IDF
  - $TF_i \times IDF_i$

Topics covered in this lecture

- Large-scale Analytics 1. Web-Scale Link and Social Network Analysis
  - Challenges in PageRank Algorithm
  - Dead ends
  - Spider traps
  - Using PageRank in a search engine
This material is built based on,

• Chapter 5
• http://infolab.stanford.edu/~ullman/mmds.html

Problems we need to avoid

• Dead end
  • A page that has no links out
  • Surfers reaching such a page will disappear
  • In the limit, no page that can reach a dead end can have any PageRank at all

• Spider traps
  • Groups of pages that all have out links but they never link to any other pages

Avoiding Dead Ends

• What if we allow dead ends?
  • The transition matrix of the Web is no longer stochastic
    • Some of the columns will sum to 0 rather than 1

• If we compute $M^k$ for increasing powers of a substochastic matrix
  • Some of all of the components of the vector go to 0
  • substochastic matrix
    • A matrix whose column sums are at most 1
  • Importance “drains out” of the Web
    • No information about the relative importance of pages
• Remove the arc from C to A
  ➔ C becomes a dead end
If a random surfer reaches C, they disappear at the next round
\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
0 & 0 & 0 & 1/2 \\
0 & 0 & 1/3 & 1/3 \\
0 & 0 & 0 & 2/3 
\end{bmatrix}
\]

• Repeatedly multiplying the vector by M:

\[
\begin{bmatrix}
7/24 \\
5/24 \\
5/24 \\
5/24 
\end{bmatrix}
\]

\[
\begin{bmatrix}
7/48 \\
7/48 \\
7/48 \\
7/48 
\end{bmatrix}
\]

\[
\begin{bmatrix}
21/288 \\
31/288 \\
31/288 \\
31/288 
\end{bmatrix}
\]

• The probability of a surfer being anywhere goes to 0 as the number of steps increase

Approaches to dealing with dead ends

• Recursive deletion
  • Drop the dead ends from the graph
  • Drop their incoming arcs as well
  • Doing so may create more dead ends
  • Drop these new dead ends

• Taxation
  • Modify the process
  • Random surfers are assumed to move around on the Web without links

Example of recursive deletion (1/4)

Example of recursive deletion (2/4)

• We still need to compute deleted nodes (C and E)

• C was the last to be deleted
  • We know all its predecessors have PageRanks (A and D)
  • Therefore,
  • PageRank of C = 1/3 x 2/9 + 1/3 x 3/9 = 1/3

Example of recursive deletion (3/4)
Example of recursive deletion (4/4)

- Now, we can compute the PageRank for E
  - Only one predecessor, C
  - The PageRank of E is the same as that of C (13/54)
- The sums of the PageRanks exceeds 1
- It provides a good estimate

Spider Traps and Taxation

- Spider traps
  - A set of nodes with no dead ends but no arcs out (from the set of nodes to the outside of the set)
  - This can appear intentionally or unintentionally on the Web
  - Spider traps causes the PageRank calculation to place all the PageRank within the spider traps

Example 1

- There is a simple spider trap of 4 nodes

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 & 0 \\
0/3 & 0 & 0 & 1/2 & 0 \\
0/3 & 0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/2 & 0
\end{bmatrix}
\]

Example 2

- The arc out of C changed to point to C itself
- C is a simple spider trap of one node
  - It still has outgoing edge to itself
  - It is not a dead end

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 0 & 1 & 1/2 \\
1/3 & 1/2 & 0 & 0
\end{bmatrix}
\]
• If we perform the usual iteration to compute the PageRank of the nodes, we get:

<table>
<thead>
<tr>
<th>Page</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>5/24</td>
<td>7/48</td>
<td>1/8</td>
</tr>
<tr>
<td>Page 2</td>
<td>5/24</td>
<td>7/48</td>
<td>31/288</td>
</tr>
<tr>
<td>Page 3</td>
<td>5/24</td>
<td>7/48</td>
<td>31/288</td>
</tr>
</tbody>
</table>

• All the PageRank is at C
  • Once there, a random surfer can never leave

To avoid Spider traps,
- We modify the calculation of PageRank
  - Allowing each random surfer a small probability of **teleporting** to a random page
  - Rather than following an out-link from their current page

The iterative step, where we compute a new vector estimate of PageRanks \( v' \) from the current PageRank estimate \( v \) and the transition matrix \( M \) is:

\[
v' = \beta M v + \left( 1 - \beta \right) \frac{e}{n}
\]

- Where \( \beta \) is a chosen constant
  - Usually in the range 0.8 to 0.9
- \( e \) is a vector for all 1's with the appropriate number of components
- \( n \) is the number of nodes in the Web graph

If the graph has no dead ends:
- The probability of introducing a new random surfer is exactly equal to the probability that the random surfer will decide to follow a link from their current page
- Surfer decides either to follow a link or teleport to a random page

If the graph has dead ends:
- The surfer goes nowhere
- The term \( (1-\beta)e/n \) does not depend on the sum of the components of the vector \( v \), there will be some fraction of a surfer operating on Web
- When there are dead ends, the sum of the components of \( v \) may be less than 1
  - But it will never reach 0
Example of Taxation (1/2)

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 1/2 & 0 \\
1/3 & 0 & 1/2 & 0 \\
1/3 & 1/2 & 0 & 0 \\
\end{bmatrix}
\]

\[
v' = \beta M v + (1 - \beta) e / n
\]

\[
\beta = 0.8
\]

Example of Taxation (2/2)

• First few iterations:

\[
\begin{bmatrix}
9/26 \\
5/60 \\
3/60 \\
\end{bmatrix}
\begin{bmatrix}
43/4500 \\
707/4500 \\
707/4500 \\
\end{bmatrix}
\begin{bmatrix}
15/148 \\
19/148 \\
95/148 \\
\end{bmatrix}
\]

• By being a spider trap, C gets more than \% of the PageRank for itself
• Effect is limited
• Each of the nodes gets some of the PageRank

Part 1. Large Scale Data Analytics

1. Web-Scale Link Analysis and Social Network Analysis

Challenges in PageRank Algorithm for the real Web: Examples

Example 1

• Compute the PageRank of each page assuming \( \beta = 0.8 \)

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 1/2 & 0 \\
1/3 & 0 & 1/2 & 0 \\
1/3 & 1/2 & 0 & 0 \\
\end{bmatrix}
\]

\[
v = \begin{bmatrix}
1/3 \\
1/3 \\
1/3 \\
\end{bmatrix}
\]

\[
\beta M v + (1 - \beta) e / n
\]

\[
\beta = 0.8
\]

Example 2

• clique
  • Set of nodes with all possible arcs from one to another
  • Suppose the Web consists of a clique of \( n \) nodes and a single additional node that is the successor of each of the \( n \) nodes in the clique

Example 2

• Determine the PageRank of each page assuming \( \beta = 0.8 \)
  • Is there a dead end?
  • Is there a spider trap?
Example 2

• Determine the PageRank of each page assuming $\beta=0.8$
• Is there a dead end?  
  • Yes, E
• Is there a spider trap?  
  • No. Having clique does not mean that there is a spider trap

Example 3

• Suppose that we recursively eliminate dead ends from the Web graph to solve the remaining graph
• Suppose that the graph is a chain of dead ends, headed by a node with a self-loop
• What would be the PageRank assigned to each of the nodes?
Link Analysis
Using PageRank in a search engine

Searching pages
• Each search engine has a secret formula that decides the order in which to show pages to the user in response to a search query consisting of one or more search terms
• Google uses more than 250 different properties of pages

Generating the final lists
• Selecting candidate pages
  • A page has to have at least one of the search terms in the query
  • Applying weight
  • Presence or absence of search terms in prominent places
    • e.g. headers or the links to the page itself
• Among the qualified pages, a score is computed for each
  • PageRank score

Problems in performing PageRank
• To compute the PageRank for a Web graph
  • We should perform a matrix-vector multiplication of the order of 50 times
  • Until the vector is close to unchanged at one iteration
• The transition matrix of the Web $M$ is very sparse
  • Representing it by all its elements is highly inefficient
  • We want to represent the matrix by only its nonzero elements
  • We want to reduce the amount of data that must be passed from the Map tasks to Reduce tasks

Representing Transition Matrices (1/2)
• The average Web page has about 10 out-links
  • We are analyzing a graph of 1.4 billion pages
  • Only one in 0.14 billion (140 million) entries is not 0
• Can we list the location of the nonzero entries and their values?
• If we use two 4-byte integers for coordinates (row#, col#) of an element and an 8-byte double-precision number for the probability value
  • 16-bytes per nonzero entry
  • The space needed is linear of nonzero entries
Representing Transition Matrices (2/2)

• For the Web graph
  • The value will be 1 divided by the out-degree of the page

\[
M = \begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 \\
\frac{1}{3} & 0 & \frac{1}{2} & 0 \\
\frac{1}{3} & 0 & 1 & \frac{1}{2} \\
\frac{1}{3} & \frac{1}{2} & 0 & 0 \\
\end{bmatrix}
\]

PageRank Iteration Using MapReduce

• One iteration of the PageRank algorithm involves,
  \[v' = \beta M v + (1 - \beta) e \] / \[n\]

  First round of MapReduce
  • Calculate \( Mv \) and store the result to \( v' \)

  Second round of MapReduce
  • For each component, multiply \( \beta \) and add \((1 - \beta)e\) / \(n\)

PageRank Iteration Using MapReduce

\[v' = \beta M v + (1 - \beta) e \] / \[n\]

• If \( n \) is small enough that each Map task can store the full vector \( v \) in main memory
  • And \( v' \)

  For the Web, \( v \) is much too large to fit in main memory

  We need striping
  • \( M \) into vertical stripes and break \( v \) into corresponding horizontal strips

Questions?