Graph Mining - PageRank

Mert Terziihan-Zhixiong Chen
Content

1. Web as a Graph
2. Why is PageRank important?
3. Markov Chains
4. PageRank Computation
5. Hadoop Review
6. Hadoop PageRank Implementation
7. Pregel Review
8. Pregel PageRank Implementation
1 Web as a Graph

- Directed graph
  - Nodes: Web pages
  - Directed edges: Hyperlinks
2 Why is PageRank important?

- Can be used for
  - Rating nodes in the graph based on their incoming edges
- We can rate websites as well
  - Web is a graph!
- Developed at Stanford InfoLab
  - Its patent is at Stanford
- Heavily used by Google
  - for ranking web pages
2.1 The Idea behind PageRank

- Simulation of a random surfer
  - begins at a web page and executes a random walk on the Web
  - With $\alpha$ probability: teleport operation
    - Type an address into the URL bar of his browser
  - With $1-\alpha$ probability
    - Jump to a web page that the current page links to
  - No out-links: perform only teleport operation
2.1 The Idea behind PageRank

- As the surfer proceeds this random walk:
  - He visits some nodes more often than others
  - These are the nodes with many links coming in from other nodes
- Idea: Pages visited more often in this walk are more important
### 3 Markov Chains

- **Discrete-time stochastic process**
  - a process that occurs in a series of time-steps in each of which a random choice is made
- **Characterized by a transition probability matrix** $P$
  - Stochastic matrix
  - Its principal left eigenvector has largest eigenvalue

\[
\forall i, j, P_{ij} \in [0, 1] \\
\forall i, \sum_{j=1}^{N} P_{ij} = 1.
\]
3 Markov Chains

- Probability distribution of next states for a Markov chain
  - depends only on current state
  - not on how Markov chain arrived at the current state

\[ P = \begin{pmatrix}
0 & 0.5 & 0.5 \\
1 & 0 & 0 \\
1 & 0 & 0 
\end{pmatrix} \]
3.1 Probability Vector

- **N-dimensional probability vector**
  - Each entry corresponds to one of the states
  - Entries are in the interval [0,1]
  - Entries add up to 1

- \( \bar{x} \): the probability distribution of the surfer’s position at any time
  - At \( t=0 \) current state is 1, others are 0

- At \( t=1 \), surfer’s distribution = \( \bar{x}P \)
- At \( t=2 \), surfer’s distribution = \( (\bar{x}P)P = \bar{x}P^2 \)
3.1 Probability Vector

- If a Markov chain is allowed to run for many steps
  - Each state is visited at a frequency that depends on the structure of the Markov chain
  - The surfer visits certain web pages more often
- The visit frequency converges to fixed, steady-state quantity
  - PageRank of each node $v$ is the corresponding entry in this steady-state visit frequency
3.2 Ergodic Markov Chain

- Markov chain is ergodic if
  - There exists a positive int $T_0$
  - For all $t > T_0$, the probability being in any state $j$ at time $t$ is greater than 0
- Irreducibility
  - There is a sequence of transitions of non-zero probability from any state to any other
- Aperiodicity
  - States are not partitioned into sets
3.2 Ergodic Markov Chain

- For any ergodic Markov chain, there is a unique steady-state probability vector
  - Principal left eigenvector of $P$

$$\lim_{t\to\infty} \frac{\eta(i,t)}{t} = \pi(i),$$

- $\eta(i,t)$ is the number of visits to state $i$ in $t$ steps
- $\pi(i) > 0$ is the steady-state probability for state $i$
- Random walk with teleporting ensures a steady-state probabilities
4 PageRank Computation

- Compute left eigenvectors of transition probability $P$
  - $\vec{\pi} P = \lambda \vec{\pi}$.
- For computing PageRank values
  - Find the eigenvector corresponds to eigenvalue 1
    - $\pi P = 1\pi$,
- There are many efficient algorithms to compute the principal left eigenvector
4.1 PageRank Example

- Consider the following web graph with $\alpha=0.5$

- Transition matrix:

\[
P = \begin{pmatrix}
\frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\
\frac{5}{12} & \frac{1}{6} & \frac{5}{12} \\
\frac{1}{6} & \frac{2}{3} & \frac{1}{6}
\end{pmatrix}.
\]

- Initial probability distribution matrix:

\[
\vec{x}_0 = (1 \ 0 \ 0).
\]
4.1 PageRank Example

- After one step:
  \[ \tilde{x}_0 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} = \tilde{x}_1. \]

- After two steps:
  \[ \tilde{x}_1 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} = \tilde{x}_2. \]

- Convergence:

<table>
<thead>
<tr>
<th>( \tilde{x}_i )</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tilde{x}_0 )</td>
<td>1/6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \tilde{x}_1 )</td>
<td>1/6</td>
<td>2/3</td>
<td>0</td>
</tr>
<tr>
<td>( \tilde{x}_2 )</td>
<td>1/3</td>
<td>1/3</td>
<td>0</td>
</tr>
<tr>
<td>( \tilde{x}_3 )</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
</tr>
<tr>
<td>( \tilde{x}_4 )</td>
<td>7/24</td>
<td>5/12</td>
<td>7/24</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( \tilde{x} )</td>
<td>5/18</td>
<td>4/9</td>
<td>5/18</td>
</tr>
</tbody>
</table>
References

- Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*. 1999
5 Hadoop Review

- A MapReduce implementation
- Decompose algorithms into two stages
  - A **map** stage that maps a key/value pair into intermediate sets of key/value pairs
  - A **reduce** stage that merges all of the values associated with the same key
- Each stage is implemented as a separate function call for each key (running on a different thread, processor, or computer)
6 Hadoop PageRank Implementation

- Parse documents (web pages) for links
- Iteratively compute PageRank
- Sort the documents by PageRank
6.1 Parse Documents

- **Map**
  - **Input**
    - &lt;html&gt;&lt;body&gt;Blah blah blah...
    - &lt;a href="2.html">A link&lt;/a&gt;....
    - &lt;/html&gt;
  - **Output**
    - key: index.html
    - value: 2.html
    - &lt;doc, child&gt;

- **Reduce**
  - **Input**
    - &lt;doc, child&gt;
    - &lt;index.html, 2.html&gt;
    - &lt;index.html, 3.html&gt;
  - **Output**
    - key: index.html
    - value: 1.0 2.html 3.html
    - &lt;doc, doc_rank children&gt;
6.2.1 Iteration-Map

- **Map**
  - **Input**  
    - `<doc, doc_rank children>`
      - `<index.html, 1.0 2.html 3.html>`
  - **Output**  
    - `<child, doc doc_rank doc_children_size>`
      - `<2.html, index.html 1.0 2>`
      - `<3.html, index.html 1.0 2>`
6.2.2 Iteration-Reduce

● **Reduce**
  ○ **Input**  `<child, doc doc_rank doc_children_size >`
    ■ `<2.html, index.html 1.0 2>`
    ■ `<3.html, index.html 1.0 2>`
    ■ `<2.html, 1.html 1.0 2>`
  ○ **Output**  `<doc, doc_rank children>`
    ■ `<index.html, new_rank 2.html 3.html>`
    ■ `<2.html, 2.0>`
    ■ `<3.html, 1.5>`
6.2.3 Iteration-Convergence

- **Reduce**
  - **Input**
    - `<child, doc doc_rank doc_children_size >`
      - `<2.html, index.html 1.0 2>`
  - **Output**
    - `<doc, doc_rank children>`
      - `<index.html, new_rank 2.html 3.html>`

- **Map**
  - **Input**
    - `<doc, doc_rank children>`
      - `<index.html, new_rank 2.html 3.html>`
  - **Output**
    - `<child, doc doc_rank doc_children_size >`

- **Reduce**
- **Map**
- **Map**
- ....
6.3 Sort Documents

- **Map**
  - **Input**  \(<doc, doc\_rank children>\)
    -  \(<index.html, new\_rank 2.html 3.html>\)
  - **Output**  \(<doc\_rank, doc>\)
    - Distributed Merge Sort
7.1 Pregel Review

- A Framework for distributed processing of large scale graphs
- Components
  - Vertex
    - Has a **User-Defined, Modifiable** value
    - Manages its out-going Edges (**UDM** value, next vertex identifier)
    - Hashed into a **worker machine**
  - Master machine
    - Manages synchronization between supersteps (iterations)
7.2 Vertex-Centric Computing

- **Master**
  - Tell workers to start superstep $S_i$

- **Vertices (of worker machines)**
  - Parallely executes the same User-Defined Function that expresses the logic of a given graph processing algorithm
    - Modify its state or that of its edges, receive messages sent to it, send messages to other vertices
    - Vote to halt if reaches maximum supersteps

- **Master**
  - If all workers are done, $i++$
  - If all workers halt, we are done!
public class PageRankVertex{
    Double value;
    List edges; //neighbors
    public void compute(Queue<Message> msgs){
        if (superstep() >= 1){
            Double sum = 0;
            for(Message msg: msgs)
                Sum += msg.val;
            Value = 0.15/edges.size() + 0.85 * sum;
        }
        if (superstep() < 30)
            sendMessageToAllNeighbors(value/edges.size());
        else
            voteToHalt();
    }
}
References

- Jasper Snoek: Computing PageRank using MapReduce, CS Department of Toronto, 2008
- Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski: Pregel: A System for Large-Scale Graph Processing, In the Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 2010
Q&A
Thank You