Data Mining and Web Algorithms

- Why web search algorithms(information Retrieval systems)
- Different Retrieval Systems
  - Boolean Retrieval
  - Ranked Retrieval
  - Link Analysis
  - Ranking algorithms
- Web Crawlers
- Web Caching Algorithms
- Recommendation Systems

Data Mining and Web Algorithms

→ Web search engines are actually a speific type of information retrieval systems.

→ "Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections."

→ In general, an IR can be

- → E-mail Search
- → Searching your laptop
- → Legal knowledge retrieval
- → Audio and Visual Retrieval Systems
- → Web Search Systems

→ Usually IR considers unstructured data / semi-structured data.

Data

Structured (Data base Tables) | Un-structured (twitter, comments, e-mail messages) | Semi-structured (JSON, XML)

→ An IR Problem :-

Eg We want to search the plays of shakespeare's which contains the words Brutu and Caesar and not Calpurnia.

IR Models :-

Option 1 :- Apply "grep" command of unix and you will get the answer.

Problem: but for larger set of documents and different kind of query such as "near countrymen", it is unpractical to do the job with grep.

Basic Terms :-

(i) Corpus : a large repository of documents

(ii) Information need (query) : A topic about which I want to collect the data (information)

(iii) Relevance :- Some of the documents in corpus that may contain what I want to search.

Option 2 :- Boolean Retrieval Model :-

→ Each document or query is treated as a "bag" of words or "terms". Word sequence is not considered.

→ Given a collection of documents D, let
(3) $V = \{t_1, t_2, \ldots t_{|V|}\}$ be the distincle words/
terms in the collection. V is called
the vocabulary.

→ A weight $w_{ij} > 0$ is associated with each
term t_i of document $d_j \in D$. For a term that
does not appear in document d_j, $w_{ij} = 0$.

$$d_j = (w_{1j}, w_{2j}, \ldots w_{|V|j})$$

usually $w_{ij} = 1$, if it is there in the document.

→ Result of above step is term-incidence matrix. Eg

↓ Terms	Antony & Cleopatra (D1)	Julius Caesar (D2)	The Tempest (D3)	Hamlet (D4)	Orthello --- (D5)
Antony	1	1	0	0	0
Brutus	1	1	0	1	0
Caesar	1	1	0	1	1
Calpurnia	0	1	0	0	0
mercy	1	0	1	1	1

Query :- Brutus AND CAESAR AND NOT CALPURNIA
solution : 11010 AND 11011 AND 10111
⇒ 100100 [This implies D1 and D4)

Problem 1 :- Building a term-document matrix in above fashion has too many zero and 1's. Hence matrix is extremely sparse as it has few non-zero entries. (Memory Wastage Problem)

Problem 2 :- Retrieval results are usually poor because term-frequency is not considered.

Option 3 :- Inverted Index :-

→ An index always maps back from terms to the parts of the document where they occur.
→ It is a standard term in IR or web search.
→ An inverted index has two things
  (i) Dictionary (Vocabulary) : set of terms
  (ii) Posting list :- for each term, we have a list that records which documents the term occur in. (which is called posting). This list is called posting list (inverted list).

Antony → | 1 | 2 | — |
Brutus → | 1 | 2 | 4 | — — |
Caesar → | 1 | 2 | 4 |
Calpurnia → | 2 | — — — |
Caesar → | 1 | 2 | 4 |

Dictionary   Postings

Dictionary is kept in memory ~~disk~~ and Postings are kept on disk.

① An important step is terms should be in sorted fashion that makes the searching faster.

② We ~~ato~~ can also record the document frequency ⑤ of each term.

Example :-

Doc 1
I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Doc 2
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

⟹

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

⟹

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Processing Boolean Queries :- Q: $\boxed{\text{Brutus AND Calpurnia}}$

(1) locate Brutus in Dictionary

(2) Retrieve its postings

(3) locate Calpurnia in the Dictionary

(4) Retrieve its postings

(5) Intersect them [(2) & (4)]

Brutus : 1, 2, 4

Calpurnia : 2

∴ query result is 2 (Document 2 has Both the token)

Problems with Boolean retrieval models

1. Works for Boolean queries that match or do not to match.

2. Most of the users are not capable to write Boolean queries.

3. In the case of large document collections, the resulting number of matching documents can far exceed the number a human user could possibly shift through.

- Most users don't want to wade through 1000s of results.
- This is particularly true of web search.

Accordingly, it is essential for a search engine to **rank** the documents matching a query. To do this, the search engine computes, for each matching document, a score with respect to the query at hand.

4. Boolean queries using AND gives very few and OR gives a lot.With a ranked list of documents it does not matter how large the retrieved set is.

5. Queries in web search is usually free text.

Ranked retrieval

Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

It doesn't consider term frequency (how many times a term occurs) in document.

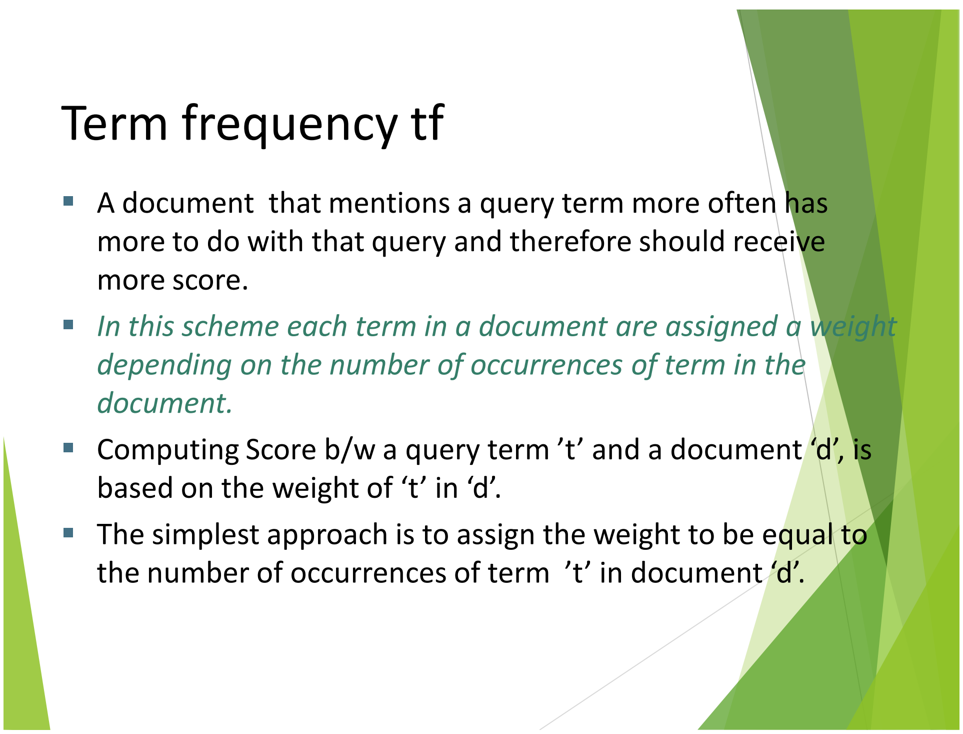
Term-document count matrices

- Consider the number of occurrences of a term in a document:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Vector Space Model

1. Documents are treated as bag of words model.
   - *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
   - This is called the bag of words model.
2. Each document is represented as a vector. Let the vector derived from document *d*, with one component in the vector for each dictionary term is denoted by *V* (*d*). The set of documents in a collection then may be viewed as a set of vectors in a vector space, in which there is one axis for each term.
3. However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of **term-frequency(tf) or tf-idf (inverse document frequency)** scheme.



Term frequency tf

- We want to use tf when computing query-document match scores. But how?

- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.

- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:

$$\text{score} = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

Document frequency (df)

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- For frequent terms, we want high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df_t is the document frequency of *t*: the number of documents that contain *t*
  - df_t is an inverse measure of the informativeness of *t*
  - $\mathrm{df}_t \leq N$
- We define the idf (inverse document frequency) of *t* by

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

tf-idf weighting

- The tf-idf weighting scheme assigns to term ‘t’ a weight in document ‘d’ given by

$$\text{tf-idf}_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t) \quad [1]$$

- Highest when ‘t’ occurs many times within a small number of documents.
- Lower when the term occurs fewer times in a document, or occurs in many documents
- Lowest when the term occurs in virtually all documents.

The vectors space model for scoring

- To compensate for the effect of document length, the standard way of quantifying the similarity between two documents *d*1 and *d*2 is to compute the *cosine similarity* of their vector representations *V* (*d*1) and *V* (*d*2)

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \quad \text{-----(3)}$$

- where the numerator represents the *dot product* of the vectors *V* (*d*1) and *V* (*d*2) and,
- The denominator is the product of their *Euclidean lengths.*

The vectors space model for scoring

- The dot product $x \cdot y$ of two vectors is defined as

$$\sum_{i=1}^{M} x_i y_i.$$

- Let $V(d)$ denote the document vector for d, with M components $V_1(d) \ldots V_M(d)$. The Euclidean length of d is defined to be

$$\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}.$$

- We can then rewrite equation (3) as

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \qquad \text{----(4)}$$

Queries as vectors

- we can also view a *query as a vector.*
- we can use the cosine similarity between the query vector and a document vector as a measure of the score of the document for that query.
- The resulting scores can then be used to select the top-scoring documents for a query. Thus we have

$$\text{score}(q, d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}.$$

Example

TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$

Query, Q: “gold silver truck”
D_1: “Shipment of gold damaged in a fire”
D_2: “Delivery of silver arrived in a silver truck”
D_3: “Shipment of gold arrived in a truck”
D = 3; IDF = $\log(D/df_i)$

		Counts, tf_i						Weights, $w_i = tf_i*IDF_i$			
Terms	**Q**	**D_1**	**D_2**	**D_3**	**df_i**	**D/df_i**	**IDF_i**	**Q**	**D_1**	**D_2**	**D_3**
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Similarity Analysis

$$|D_1| = \sqrt{0.4771^2 + 0.4771^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.5173} = 0.7192$$
$$|D_2| = \sqrt{0.1761^2 + 0.4771^2 + 0.9542^2 + 0.1761^2} = \sqrt{1.2001} = 1.0955$$
$$|D_3| = \sqrt{0.1761^2 + 0.1761^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.1240} = 0.3522$$

$$\therefore\ |D_i| = \sqrt{\sum_i w_{i,j}^2}$$

$$|Q| = \sqrt{0.1761^2 + 0.4771^2 + 0.1761^2} = \sqrt{0.2896} = 0.5382$$

$$\therefore\ |Q| = \sqrt{\sum_i w_{Q,j}^2}$$

- Next, we compute all dot products (zero products ignored)

$$Q \bullet D_1 = 0.1761 * 0.1761 = 0.0310$$
$$Q \bullet D_2 = 0.4771 * 0.9542 + 0.1761 * 0.1761 = 0.4862$$
$$Q \bullet D_3 = 0.1761 * 0.1761 + 0.1761 * 0.1761 = 0.0620$$

$$\therefore\ Q \bullet D_i = \sum_i w_{Q,j} w_{i,j}$$

- Now we calculate the similarity values

$$\text{Cosine } \theta_{D_1} = \frac{Q \bullet D_1}{|Q| * |D_1|} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$\text{Cosine } \theta_{D_2} = \frac{Q \bullet D_2}{|Q| * |D_2|} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$\text{Cosine } \theta_{D_3} = \frac{Q \bullet D_3}{|Q| * |D_3|} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

$$\therefore\ \text{Cosine } \theta_{D_i} = \text{Sim}(Q, D_i)$$

$$\therefore\ \text{Sim}(Q, D_i) = \frac{\sum_i w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2}\sqrt{\sum_i w_{i,j}^2}}$$

- Finally we sort and rank the documents in descending order according to the similarity values

Rank 1: Doc 2 = 0.8246

Rank 2: Doc 3 = 0.3271

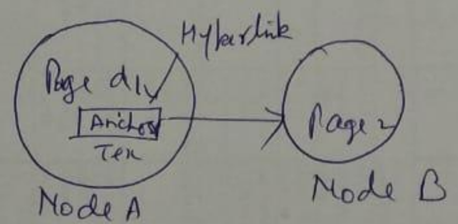
Rank 3: Doc 1 = 0.0801

· Traditional,
↳ Information Retrieval (document Retrieval) investigates relevant document in a small and trusted set. (Eg newspaper articles, patents etc)

→ But, in Web Search, Web is huge and full of untrusted documents, random things, web spam etc

→ Web can be thought of as a graph consisting of

- Nodes :- Webpages
- Edges :- Hyperlinks.



→ Web challenges :-

–(i) Web contains many sources of information. Who to trust ?

Trick :- Trustworthy pages may point to each other. [Hyperlinks]

–(ii) What is "best" answer to query "newspaper"

→ No single answer

→ Hence, by looking at web-structure we can rank the webpages.

Link analysis :-

→ To ~~analst~~ analyze the web structure, we do link analysis.

→ Different Approaches in this course for link analysis are -

- Page Rank
- Hubs and authorities [HITS]

PageRank :- [

→ PageRank algorithm determines the importance of web pages based on link structure

→ It is a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.

→ Central part of Google's search engine is PageRank developed by Larry Page and Sergey Brin at Standford University.

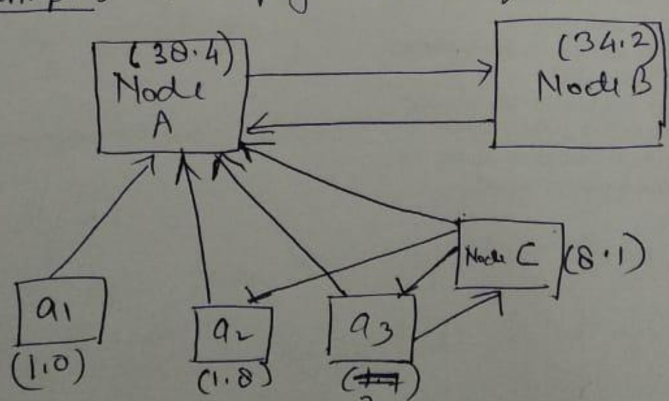
(3)

Basic Idea for Page Rank

→ Consider links as votes

- Page is more important if it has more links.
- Whether incoming | outgoing links.
- Incoming links are actually more important and conveys that that webpage is more trusted.
- Are all the links are same? It depends on the importance of other pages incoming to it.

Example :- Web pages as a graph with page Ranks:



(1) Node A has high Page Rank as it has more incoming links.

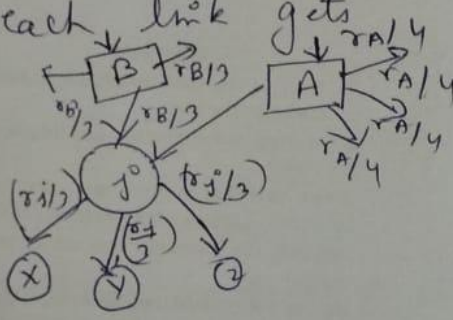
(2) But Node B has also higher rank as compared to Node C which also has single incoming link (from a_3) as it has getting link (incoming) from Node A.

(3) Hence, it is recursive.

Simple Recursive formulation can be done by making these statements -

→ Each link's vote (Rank) is proportional to importance of its source page.

→ If page j with importance r_j has n outlinks, each link gets $\boxed{r_j/n}$ votes



→ Hence, Page j's own

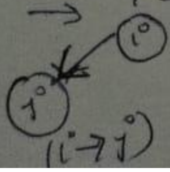
$$r_j = \frac{r_A}{4} + \frac{r_B}{3}$$

importance is the sum of the votes on its links.

→ A "vote" from an important page is more worth.

→ A page is important if it is pointed by other important pages.

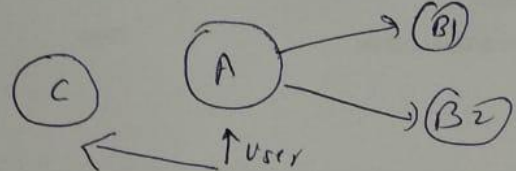
→ Hence rank of page "j" can be defined as



$$r_j = \sum_{i \to j} \left(\frac{r_i}{d_i} \right)$$, where d_i => outdegree of node i

(5)

→ PageRank is considered an model of user (surfer) where a user can click randomly on any page other ~~that~~ than going to page that it points to



User can go directly to "C" rather than going to B1 & B2 pointed by webpage A.

→ Google uses this random surfer's model and hence include some probability α going to page B1 and B2 and not going to that page (going to page c) as $(1-\alpha)$. and hence we can formulate the following equation.

Original PageRank algorithm :-

→ It is an iterative algorithm that starts with given .

$$PR(A) = \underbrace{(1-d)} + d\left[\frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} \cdots \frac{PR(T_n)}{C(T_n)}\right]$$

where, the probability for random surfer jumping to a page is always $(1-d)$.

→ $PR(A)$ = PageRank of page A

→ $PR(T_i)$ = PageRank of Pages T_i which link to page A (inbounds links)

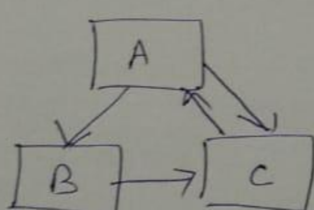
→ $C(T_i)$ = No of outbound links on page T_i

→ d = damping factor lies B/w 0 & 1.

→ According to Page and Brin (who have proposed)

$d = 0.85$.

Eg



No-of Nodes = 3

Let us assume initial are 1.

$$PR(A) = (1-d) + d\left[\frac{PR(C)}{C(C)}\right]$$

$$= (0.15) + (0.85)\left[\frac{PR(C)}{1}\right] \quad \text{—— (1)}$$

$$PR(B) = (1-d) + d\left[\frac{PR(A)}{C(A)}\right]$$

$$= (1-0.85) + (0.85)\left[\frac{PR(A)}{2}\right] \quad \text{—— (2)}$$

(*)

$$PR(C) = (0.5) + (0.5)\left[\frac{PR(A)}{C(A)} + \frac{PR(B)}{C(B)}\right]$$

$$= 0.5 + 0.5\left[\frac{PR(A)}{2} + \frac{PR(B)}{1}\right] \quad \text{— (3)}$$

Let $\boxed{PR(A) = P(B) = P(C) = 1}$ initial $\boxed{0^{th} \text{ iteration}}$

$\boxed{I^{st} \text{ iteration}}$

$$\left.\begin{aligned} PR(A) &= 0.5 + 0.5 = 1 \\ PR(B) &= 0.75 \\ PR(C) &= 0.5 + 0.5[0.5 + 0.75] \\ &= 0.5 + 0.5 * 1.25 \\ &= 0.5 + \cancel{0.75}.625 = 1.125 \end{aligned}\right\}$$

Sum of PageRank of all the pages = No. of web pages

$$= 1 + .75 + 1.125 = 2.875 \approx 3$$

It has to repeated (iteratively) until we get the same (or almost same) value of page Rank.

Another notation of pagerank eq^n :-

$$PR(A) = \frac{(1-d)}{N} + d\left[\frac{PR(T1)}{C(T1)} + \frac{PR(T2)}{C(T2)} \cdots \frac{PR(Tn)}{C(Tn)}\right]$$

Then sum of all pages will be One.

References:

[1] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. Cambridge university press, 2008.

[2] Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive data sets. Cambridge university press, 2020.