AIW & Information Retrieval (UE18CS322)
Unit 4

Aronya Baksy

May 2021

1 Web Advertising

1.1 Cost Per Impression Model

- The purpose of advertising is
  - Convey positive feelings about a brand or product
  - Generate awareness about the new product
- Advertising impact is measured in cost per 1000 impressions.
- Decision whether to display banner Ads depends on product. Yahoo does it, Google does it only selectively

1.2 Cost Per Click Model

- Advertiser is charged when their advertisement gets clicked
- Advertisers bid for "keywords". Ads for highest bidders displayed when user query contains a purchased keyw
- e.g.:
  - **YouTube Affiliate Marketing** : YouTube Affiliate will make videos with the aim of getting you to buy products from a 3rd party site
  - **Amazon Affiliate**: entirely free for website owners and bloggers to be Amazon affiliates, allow Amazon ads on their website/blog.

1.3 Cost Per Action Model

- Advertiser pays for an action undertaken, e.g. buying a product
- e.g.: **Amazon Associates** advertise products from Amazon.com on their websites by creating links.
- When customers click the links and buy products from Amazon, they earn referral fees

1.4 Comparison

- 1% of all viewers click on an ad. 1% of all clickers actually buy a product. Hence impression rate is largest, but action rate is lowest
- Suppose advertiser pays x for 1000 impressions:
  - As per **CPI** model, cost per impression = $\frac{x}{1000}$
  - As per **CPC** model, cost per click = $\frac{x}{10}$
  - As per **CPA** model, cost per action = $\frac{x}{0.1} = 10x$

- CPI is most risky, CPA is least risky. CPI is risky because money is being spent to show to people who are mostly not interested
- If the risk needs to be divided between advertiser and publisher, CPC is the chosen model
- Publisher would prefer CPI (more frequent). Advertiser prefers CPA (least risk).

1.5 Search Ads

- Closer connection between viewing a search ad and actually buying a product, as against randomly seeing the ad on TV/Newspaper
- Allows for a CPC model.
- Parties involved:
  - **Publisher**: gets revenue everytime someone clicks on an ad, and punish misleading or irrelevant ads. (e.g.: Google)
  - **Users** click on ads and get their needs fulfilled
  - **Advertisers** find new customers by paying publishers to host their ads.
- Issues with search Ads:
  - **Keyword Arbirtrage**: Buy a keyword at Google, redirect to a service that is more cost effective.
  - **Click Spam**: Publisher executes clicks for fake users, and extracts money from advertisers.
- First gen search engines like **Goto** ordered search results for a query **q**, by the money bidded by each advertiser for query **q**.
- Next gen search engines combined the pure algorithmic behaviour (Google etc.) with the sponsored behaviour. Show both search results side by side.
- Search engines claim not to let sponsorships affect the actual search ranking.

1.5.1 Ranking for Sponsored Ads

- **Auction system**: advertisers bid for keywords, anyone can participate in auction
- To determine the rank of the ad, a second price auction is held
- Types of ranking: based only on bid price, or based on both bid price and relevance. Relevance measures for ads are:
  - Click through rate (clicks per impressions)
  - Location, time of day
  - Lnding page quality, loading time for landing page
- An Ad rank for an ad is analogous to Pagerank for a normal webpage. The Adrank formula may consist of CTR, landing page experience, ad relevance to query, ad formats. etc.
- An auction is a NOT bargaining or negotiation, rather it is a price discovery mechanism

1.5.2 Vickrey Auction

- Also called the sealed-bid, second-price auction.
- All bidders submit their price simultaneously, without knowing anyone else's bids. The highest bidder is the winner of the auction, but the winner pays the price of the second highest bid.
- In Google's version of the second-price auction, the rank of a bidder is determined by the value of $bid\ value \times CTR$, instead of just the bid value.

- The advertiser pays the minimum amount necessary to maintain their position in the auction. This is calculated by:

$$p_n = p_{n-1} \times \frac{CTR_{n-1}}{CTR_n}$$

where n is the auction rank of a bidder.

- The actual formula may depend on more criteria than just the CTR, such as landing page quality, relevance of query and ad etc.

1.5.3 Google Display Network

- The GDN is Google's network of websites and apps where it displays advertisements to internet users.
- Google pays these websites and apps to host the banner ads.

1.6 Google's Search Ad System

1.6.1 AdWords

- Based on search keywords.
- Businesses that use AdWords will pay Google a sum for advertisement based on keyword search
- **AdWords for Video** allows businesses to display video ads in YouTube search results, within a YouTube video or anywhere within the GDN.

1.6.2 AdSense

- For website publishers, they can monetize their websites by publishing AdWords ads.
- Publishers get paid for each click on an ad hosted on their website.
- Google uses its technology to serve advertisements based on website content, the user's location, and other factors

1.7 Programmatic Advertisement

- The process of buying and selling ads with software and publishing those ads contextually
- Context is handled by algorithms that account for user browsing history, IP Address, location, time of day etc.
- **Demand Side Platforms** are third-party software that allow one to purchase, analyze, manage ads across many networks from a single place
- **Supply Side Platforms** are third-party software that allow publishers to auction off their inventory, fill it with the winning buyer's material and earn revenue. They communicate with the DSP regarding impressions
- **Data Management Platforms** are used to store, compile and analyze data which is used by DSP and SSPs to optimize.

1.8 Understanding Search Users

- User queries may be **informational**, **navigational**, **transactional** or **others**
- Discering the query type is important for advertisers. Informational and transactional queries have good potential for sponsored results, while that of navigational is low.

2 Characteristics of the Web

- The web document collection has the following characteristics:
  - No global design or co-ordination rules
  - Distributed content creation and linking. Content may be dynamically generated
  - Various levels of truth: obsolete truths, complete falsehoods, contradictions etc
  - No universal notion of trust
  - May be unstructured, semi-structured, or structured
  - Very highly scalable, fast growing
- The web may be represented as a **static graph** where each HTML page is a node and each hyperlink is a directed link to another node.
- The **degree centrality** measure ranks nodes/pages with more connections higher in terms of centrality (ignoring direction or weight of connections)
- Drawback of degree centrality: Only local neighbourhood, cannot measure impact on the global graph network, and does not work with directed graphs
- The eigenvector approach attempts to find similarity patterns on a global scale, ignore local effects as much as possible.
- Bow-Tie structure: every strongly connected component of the web graph has an out-component and an in-component.
- Corollary: if a node belongs to both in and out components of a SCC, then that node is part of the SCC.

3 SEM and SEO

3.1 Search Engine Marketing (SEM)

- Understand how search engines rank ads, develop bidding strategy for keywords
- Goal is secure top rank in ads shown by search engine, with optimum amount of money spent

3.2 Search Engine Optimization (SEO)

- Understand how search engines rank pages
- Goal is to have the web pages of the organization secure top rank in the pages shown by search engine for a given query (having certain keywords)
- "Tuning" your web page to rank highly in the algorithmic search results for select keywords
- Alternative to paying for ad placement, and thus, intrinsically a marketing function

3.3 Spam

- Motives: political/religious/commercial, funded by advertising budget
- Operators: contractors (SEO), web masters or hosting services
- Forums: webmasterworld, academic discussions

3.3.1 Keyword Stuffing

- Defeat Tf-IDF bys stuffing pages with repeated terms (hide the repeated terms in the background of the page maybe)
- Such repetitions not visible to website users, but since the crawler can see them, they increase the TF-IDF relevance score
- Implemented as meta-tags (hidden metadata tags in HTML page) or hidden text (using style tricks)

3.3.2 Cloaking

- DNS cloaking: Switch IP address. Impersonate
- Serve fake content to search engine spider with misleading keywords
- For the actual user, the content will be different !

3.3.3 Other Spamming Techniques

- **Doorway pages**: pages optimized for a single keyword that redirect to the actual page.
- **Clickbait**: actual content may not be valuable or interesting, but headline or title is attractive enough to generate impressions and clicks
- **Link Spamming**: Mutual admiration (in the form of hyperlinks), or domain flooding (multiple domains that point to a single webpage)

3.3.4 Countering Spam

- Alternate metrics for page quality: user votes or author votes
- Policing of URL submissions
- Link analysis detects spammers (guilt by association)
- Using machine-learning techniques (text mining)
- Linguistic analysis, image analysis to detect family-friendly content
- Editorial (human) intervention using blacklisting, top queries audited, pattern detection

3.4 Web Search Index Size

- With increasing index size, recall of search engine will suffer, and search engine may become less efficient at document retrieval.
- Indexes vary in scope across search engines, hence there is no single measure of index size.
  - Some search engines may return content not within the index, but based on rules such as anti-spam rules, max url depth, max count/host etc.
  - Search Engines index different things under the same URL and all of them may not get used for all queries

3.4.1 Capture-Recapture Method

- Initial hypothesis: size of web is finite, each search engine is indexing an uniform, independent and randomly chosen subset of the web
- Given 2 indices E_1 and E_2. Symmetrically test whether a page from E_1 is in E_2, and similarly if a page from E_2 is in E_1.
- The result is that we get x fraction of pages in E_1 are in E_2, and that a fraction y of pages in E_2 are in E_1.
- Then, according to the initial hypothesis, we have

$$x|E_1| \approx y|E_2|$$
$$\frac{|E_1|}{|E_2|} \approx \frac{y}{x}$$

3.4.2 URL Sampling Approaches

- Can be done randomly, from the index of one of the engines, or from a collection of documents entirely outside both indices.

1. **Random Searches**:
   - Starting with a search log (generated from a single user group maybe), send a random search from the log to E_1 and a random page from the results.
   - The bias in this approach comes from the types of searches made by a particular user group
2. **Random IP Addresses**:
   - Generate random IP, send request to that IP and get all pages stored at that server.
   - Biases here include the types of pages stored on the server, the fact that many servers share a single IP (virtual hosting)
3. **Random Walk**:
   - Start from arbitraryt page in the page graph, and start a random walk. The random walk converges to a steady state where probability of picking any page becomes constant
   - Since the web graph is not strongly connected, steady state may not be reached. Even if it is, time taken to reach steady state is not known at the beginning.
4. **Random Queries**:
   - Pick a set of random terms from a web dictionary (put together by a web crawler), then build random conjunctive queries from these.
   - For each query, get a random page p from the top 100 results on E_1. From p, get some low-frequency terms, build a query from them and send to E_2. This tests presence of p in E_2.
   - Biases:
     - Biased towards longer documents
     - Picking top 100 of E_1 introduces ranking bias of E_1
     - E_2 may not handle the queries properly
     - Operational issues like connection timeout.

4 Handling Duplicates in Web Search

- Aim is to compute syntactic similarity (not semantic, too hard). Similarity threshold θ denotes whether 2 documents are duplicates or not.

1. **Naive Approach 1**:
   - Convert both documents to sets, then compute Jaccard Similarity. Document pairs with Jaccard coefficient above θ are flagged as duplicate
   - For a corpus of N documents, number of comparisons is $C(N, 2)$ which is $O(N^2)$ comparisons.
2. **Naive Approach 2**:
   - Compute, for each web page, a fingerprint that is a 64-bit summary of the characters on that page. Compare fingerprints for each pair
   - Complexity is $O(N^2 d)$ where d is number of words on document. Impractical, and also fails to detect *near duplicates*.
3. **Shingles**:
   - Also called $n - grams$. Build sequences of n consecutive terms from the document.
   - Let $S(d_i)$ be the set of shingles built from document d_i. Find Jaccard similarity between sets of shingles, and flag as duplicate if score exceeds threshold θ

- Shingle size:
  - Small shingles appear in too many documents. Large shingles appear in too few.
  - Shingle size should be proportional to average document length in corpus.
  - Comparing all pairs of documents for shingle set in each document will not scale – still $O(N^2)$

4. **Sketches**:
   - Select a random number of shingles from each document, this set is now called a sketch.
   - Reduced number of comparisons, accuracy suffers but good enough.
5. **Hashing**
   - Represent each shingle as a unique number. A 64-bit hash function hashes large shingles, also handles collisions due to hashing
   - Space saving due to storing just one integer instead of a sequence of characters
   - Efficient due to just number comparision, instead of comparing strings.

4.1 Locality Sensitive Hashing

- Every time a new document enters the collection, it has to be compared to all N documents in the corpus again, not efficient as N scales up.
- LSH: an algorithm such that if we generate hashes of 2 documents, it tells us if their similarity is greater than a threshold θ in a non-brute force method.
- Key idea, $similarity(D_1, D_2) \propto Prob(H(D_1) == H(D_2))$. Choice of hash function $H(D)$ is linked to choice of similarity function. (for Jaccard, we use min-hashing)
- The min-hash function generates a fixed-length signature for each document.
- The algorithm for getting Min-Hash similarity is:
  - Input: matrix with shingles across the rows, documents across columns
  - Generate k permutations of the matrix (row-wise, i.e. move the rows around only). For each permutation π, generate the signature using the function

    $$H_\pi(C) = min_\pi C$$

    meaning, the index of the first (in the permuted order) row in which column C has value 1
  - For each permutation, now we have a row vector of length $|D|$. Stack these rows on top of one another to get the signature matrix M.
  - Each column of M is the signature or hash of the corresponding document. THe length of each signature is k.
  - Similarity between 2 documents is basically Jaccard similarity of the corresponding columns.
- With larger corpora, creating permutations becomes expensive. Hence in the real-life minhash implementation, this permutation is represented as combinations of multiple hash functions.

4.1.1 Band partitioning

- Take signature matrix, divide it horizontally into b **bands**, each band having r rows.
- For each band, hash its portion of each column to a hash table with k buckets
- Candidate column pairs are those that hash to the same bucket for at least 1 band
- Requirement: If 2 documents have similarity greater than θ then probability of them sharing the same bucket in atleast one of the bands should be 1

- Calculations:

$$P(d_1 \text{ and } d_2 \text{ are identical in one band }) = \theta^r$$
$$P(d_1 \text{ and } d_2 \text{ not similar in all } b \text{ bands }) = (1-\theta^r)^b$$
$$P(d_1 \text{ and } d_2 \text{ similar in atleast 1 band }) = 1-(1-\theta^r)^b$$

- The second equation is the **false negative rate** (if we want d_1 and d_2 to be hashed to the same bucket).
- The third equation is the **false positive rate** (if we don't want d_1 and d_2 to be hashed to the same bucket).

4.1.2 Real-life Implementation

- Shuffling too slow, hence people simply rehash the shingle to get a new signature for it and then store smallest one (from set of shingles) in the first signature slot.
- Typically universal hash function is used with separate parameters for each slot so that we get effectively a different

$$H(X) = ((ax+b) \text{ modulo } p) \text{ modulo } N$$

5 Web Crawler

5.1 Basic Operation

- Initialize queue with seed URLs. Remove one from queue, parse the page, extract the URLs from it and add them to queue.
- This is effectively a BFS traversal of the web graph.
- **URL frontier** consists of the set of URLs identified for crawling but not yet crawled
- Requirements for a crawler:
  - **Polite**: respect implicit and explicit politeness criteria (timeouts, well spaced out requests to avoid DoS, other restrictions)
  - **Robust**: Immune to spider traps and other malicious behaviour
  - **Distributed**: Run on multiple machines, no need to fetch all pages to a single machine
  - **Scalable** with number of machines
  - **Efficient**: in terms of resource utilization (memory and network bandwidths)
  - **Priority**: Fetch high quality pages first
  - **Continuous**: Always keep fetching pages and their updated fresh copies
  - **Extensible** to new protocols and data formats
- Implicit politeness: no specifications on allowed URLs, but do not hit any one server too often
- Explicit politeness: webserver specifies blocked URLs for crawling in a `robots.txt` file.
- DNS cache or Batch DNS resolvers reduce latencies associated with DNS lookup.
- Normalize relative URLs into absolute URLs.
- Do not crawl duplicate pages (verify using shingles/fingerprints)
- Cache all `robots.txt` files, avoid repeatedly retrieving them.
- Processing steps:
  - Pick URL from frontier. Fetch document at that URL
  - Extract links to other documents from the page
  - If URL not already seen, add to index.
  - For each extracted URL, test for filters, test whether it already exists in frontier.

5.2 Distributed Crawler Implementation

- Every node completes one step in the processing, then send the processed URL to another node (decided using some hash function).
- Every node receives as well as sends URLs.
- Politeness and freshness are conflicting goals in a simple FIFO traversal.
- In a distributed scheme, freshness is denoted by a priority. Priority is assigned using heuristics (website popularity, refresh rate from previous crawls, application-specific)
- According to the priority assigned, each URL goes into a separate **front queue**.
- A front queue is selected by the **front queue selector** (round robin fashion) with an added bias that favours the high priority front queues. From the selected FQ, a URL is picked.
- The back queue router routes the selected URL from the selected queue, to a **back queue**. Each back queue is on one host machine.
- Back queues are organized either by URL source (one back queue contains only URLs from one web host), or by emptiness (make sure no back queue is empty at any time)
- The **back queue heap** contains one entry for each back queue. The nodes are sorted in a min-heap fashion by the earliest time the coresponding host can be hit again (t_e)
- A crawler thread gets the heap root, picks a URL from the corresponding back queue q.
- If q is empty, pull an URL v from the front queue chosen. If there's already a back queue q for v's host, append v to q and pull another URL from front queues, repeat till the back queue is filled.
- Else add v to q if v is a new host.
- Recommended: number of back queues = 3 x number of crawlers

6 Page Rank

6.1 Background

- Given a square, non-singular matrix A, and a column vector $\overrightarrow{x}$, the values λ that satisfy the equation
$$A\overrightarrow{x} = \lambda\overrightarrow{x}$$
are called the **right Eigenvalues** or simply the **Eigenvalues** of the matrix A
- For each Eigenvalue λ, the corresponding vector $\overrightarrow{x}$ is called the **Eigenvector**.
- **Left Eigenvectors** (row vectors) and **Left eigenvalues** satisfy the following:
$$\overrightarrow{x_L}A = \lambda\overrightarrow{x_L}$$
- A Markov Chain is called **Ergodic** if there exists a positive integer T_0 such that for all pairs of states i and j, if chain starts at $t = 0$ in state i then probability of being in state j at time $t > T_0$ is > 0.
- The conditions for Ergodicity are:
  - **Irreducible**: there exists a sequence of transitions of non-zero probability from any state to any other
  - **Aperiodic**: states are not partitioned into sets such that all state transitions occur cyclically from one set to another
- **Steady State Theorem** : For any ergodic Markov Chain, there is a unique steady state probability vector π that is the **principal left Eigenvector** of transition Matrix P
- The **PageRank Vector** is simply the steady state distribution of a random walk process over the web graph.

6.2 Eigenvector Centrality

- The centrality (aka popularity, aka PageRank) of a node is dependent on the centrality of its neighbours, instead of just the degree of that node in the graph.
- Assume that all pages have the same initial score and then iteratively update until the steady state distribution is reached

6.3 Hotelling's Power Method for Eigenvalue Centrality

- Let x_0 be the initial pagerank vector for all nodes, and A be the transition probability matrix
- We define the PageRank at time t, x_t as

$$\begin{aligned} x_t &= Ax_{t-1} \\ x_t &= A.Ax_{t-2} \\ \implies x_t &= A^2x_{t-2} \\ \implies x_t &= A^tx_0 \end{aligned}$$

- Let x_0 be the linear combination of the eigenvectors of A, hence

$$\begin{aligned} x_0 &= \sum c_iv_i \\ x_t &= A^t \sum c_iv_i \end{aligned}$$

- Since we have $Av = \lambda v$ for all v_i, hence we have $A^tv = \lambda^t v$, and hence

$$x_t = \sum \lambda_i^t c_i v_i$$

- Divide both sides by λ_1^t where λ_1 is the dominant i.e. largest eigenvalue

$$\frac{x_i}{\lambda_1^t} = \sum \left(\frac{\lambda_i}{\lambda_1}\right)^t c_iv_i$$

- As $t \to \infty$, we have

$$\lim_{t\to\infty} \frac{x_i}{\lambda_1^t} = c_1v_1$$

  as all other terms become 0. Thus the steady state distribution is essentially the **dominant eigenvector** of matrix A.

6.4 Modification for Directed Graphs

- Now the page rank for each node is dependent only on the **incoming links into that node**.
- Hence a page with many outgoing links but no incoming links will still have popularity 0, which is not realistic.
- To solve this, two constants α and β are introduced.
- α is a damping constant on the actual page rank value. β is the initial page rank added to the rank of all nodes.
- The modified calculation is now:

$$x_i = \alpha \left(\sum_{j=1}^{n} A_{ji}x_j \right) + \beta$$

6.5 PageRank Modification

- Issue: Everyone known by a well-known person is assumed to be well-known.
- To mitigate this problem, divide the value of passed centrality by the number of outgoing links
- The modified calculation is now:

$$x_i = \alpha \left(\sum_{j=1}^{n} A_{ji} \frac{x_j}{d_j^{out}} \right) + \beta$$

6.6 PageRank Formula using Random Walk

- The walk is currently at node j with probability b_j.
- For each node j that the target node i is linked to, the proability of transitioning to i is $\frac{1}{l_j}$ where l_j is the number of outward links at node j
- Hence we get the total probability of landing up at node i in the next step as

$$P = \sum_{\text{incoming nodes to } i} \frac{b_j}{l_j}$$

- In matrix form this is simply written as $A^T v$ where A is the transition probability matrix, and v is the current state probability distribution. This is essentially the standard pagerank update rule
- *Modified walk*: with probability s ,the walk follows a random edge as before and with probability $1 - s$, it jumps to a node chosen randomly
- Hence the new revised formula is

$$P = s \left(\sum_{\text{incoming nodes to } i} \frac{b_j}{l_j} \right) + \frac{1-s}{N}$$

- *Google's approximation*: for a sparse Google matrix, the 1/n values in matrix B can be approximated with 1. Hence the revised formula is

$$P = s \left(\sum_{\text{incoming nodes to } i} \frac{b_j}{l_j} \right) + (1-s)$$

6.7 Limitations of pagerank

- **Rank Sink** occurs when a page has no outgoing links, only incoming links, hence it contributes nothing to the page rank of other pages
- **Hoarding** or circular reference is the situation when a group of pages link only among themselves, hence monopolizing the pagerank
- **Dangling nodes** which prevent random walk from functioning normally.
- **SOLUTION**: The random surfer spends proportion of the time following links at random and a proportion of the time 'teleporting' to a new random URL.

6.8 Possible Enhancements to PageRank

- Build **topic specific pagerank distributions**. For users interested in one topic, only that specific pagerank distribution is invoked
- THis is possible when the search engine knows the user's interests
- Users having multiple interests can be modelled as a linear combination of each topic's pagerank.
- Choose the proportion of each topic, build a linear combination which is essentially an Ergodic Markov chain that is now personalized to the user

6.9 Real-World Impact

- In practice: rank is assigned according to weighted combination of raw text match, anchor text match, PageRank & other factors.
- PageRank continues to be an important part of the page importance score that is assigned by the search engines.

7 Hyperlink Induced Topic Search (HITS)

- Search Engine does not have much context to assess the intent of all links that the search engine gives as query result.
- One possible approach: collect all web pages that mention the query, find that maximum number of links from all these pages point to which website, and then push that website as the best result
- This is done using the following:
  1. Collect a large sample of web pages that relevant to the query by a text only retrieval system
  2. Let these pages vote through their links
  3. Find out the page on the web that receives largest number of in-links from the sample pages collected
- For each sample page, it's score is the sum of number of votes received by each link it voted for.
- Once each sample page has a score, the new score for each link is computed as the sum of scores for all the pages that voted for it.
- There are 2 main types of pages:
  - **Authorities** are pages containing useful information i.e, the prominent, highly endorsed answers to the queries
  - **Hubs** are pages that link to authorities
- Typically weights are normalized such that sum of squared hub weights and authority weights are 1.
- It can be proved that after N iterations ($N = 5$ typically) the hub and authority scores converge to a steady state.

7.1 Implementation of HITS

- Executed in two steps:
  1. **Sampling Step** : to collect a set of relevant web pages given a topic
  2. **Iterative Step** : To find the hubs and authorities using the information collected during sampling
- Sampling is done by querying a search engine and then using the most highly ranked web pages retrieved for determining hubs and authorities

7.2 Drawbacks

- No clear cut distinction between hubs and authorities, some pages may be both hubs and authorities
- Topic drift: the sampled documents in fact may not be very relevant to query being posted
- Auto generated links represent no human judgements but HITS still gives them equal importance

HITS	**PageRank**
Introduced in 1997	Introduced in 1998
Used on result set of query	Used on all pages in the Web
Normalization after every iteration	No normalization involved