CISC 372
HIT/PageRank

The Internet and the Web are large information structures, with many interesting patterns to be learned from:

* the content
* the connections
* and how humans interact with these large structures

These patterns are of interest to:

* ordinary users using search engines
* businesses and organizations that want effective web sites
* researchers who want to understand how
    -- to best organize information
    -- how humans interact with large information structures
    -- how humans learn

Example: clickstreams

When people are foolish enough to have cookies turned on when they browse, web sites can capture the sequence and timing of their visits to pages of an organization's web site.

Dataset:

* each row represents one visit
* rows are sequences of page ids (so they're not all the same length)

This data can be used to:

* figure out what information people have trouble finding
* find ways to make them visit a large number of pages, so that there are many chances to show them ads (!!!)
* figure out why customers visit item selection pages, appear to decide to buy items, and then fail to go to a checkout page.

Clickstream data is quite different from the datasets we have considered until now – it can't easily be represented as a matrix.

It's a special case of temporal data mining, where the relationship between entries in the same row is important (the order of the columns is significant).

Algorithms for clickstreams:

Association rules generalizes to this setting:

we want to find ordered itemsets that are frequent enough;

Pages A → C → F → H are visited (in this order) 25% of the time.

Pages A → C → F are visited 50% of the time.

So a visit to pages A, C, F has a 0.5 probability of continuing to page H.

We could also map the clickstreams onto the graph whose nodes are pages and whose edges are links.

We can compute the frequency of visiting each page and traversing each link (but this loses temporal information).

For each clickstream, we could label each graph edge by where it comes in the stream; and then compute the average, for each edge, over all of the clickstreams.

Edges that always come late in a clickstream will have large averages – it's a measure of edge availability to the world.

Edges with a large standard deviation might be those that some people have a hard time finding.

Similarly we could compute the `depth' of a page in a website in terms of how many clicks it usually takes to get to it.

Indexing the web.

Web indexes such as Google, Ask, Bing have a much harder problem to solve than standalone information retrieval systems:

* the web is huge – Google indexes ?20 billion pages; there are probably about twice as many pages whose content is generated on the fly.

* the word-document matrix is sparse because there are enough pages (and esp. technical pages) that really rare words actually get used; acronyms are plausible search terms; plus web pages are in many languages

workable English: 300 words

educated English: 60,000 words

total English: 1,000,000 words

acronyms: 40,000 strings

mis-spellings: 40,000 words

+ ~55 languages at Google

Web crawlers must look around often enough that their results are reasonably fresh

23% of web pages change daily

40% of commercial web pages change daily

Most changes are small

… but many front pages change a lot and often

Half-life of all web pages is 10 days (standard deviation very large – scholarly pages about 2 years)

8% of pages new each week, and two-thirds of these have new content

Links change too…

** Web page authors try to game the system so that their pages will rank highly **

The challenge for web search engines is to come up with a combination of: preprocessing, clever algorithms, and realtime result generation, that gives users what they want to see.

Hardly any users look beyond the first 20 results returned, so if you don’t get it almost exactly right, you’ll be perceived as performing very poorly.

The three main algorithms are:

1. HITS (Hypertext Induced Topic Search)

2. PageRank (Google)

3. SALSA (Stochastic Approach for Link Structure Analysis)

All depend on the existence of explicit links between pages (???).

When crawlers fund a page, they create index entries for the words that it contains.

All have agreed that they will not index documents that are not connected to the Web (although there are lots of other crawlers for which this guarantee does not apply).

Most pages are visited ~weekly, although this is modified by a number of factors.

It’s hard to keep up with the growth of the Web.

HITS.

Useful pages are:

Authorities, pages that are pointed to by many other pages

Hubs, pages that point to many other pages

Good hubs point to good authorities.

The simple version:

For each page compute an authority score and a hub score.

Initialize these to small values

Repeatedly:

New authority score ← sum of hub scores of pages that point to you

New hub score ← sum of authority scores of pages you point to

Let L be the adjacency matrix of the (directed) web; entry l_{ij} is 1 if page *i* has a link to page *j*.

Then

$$x^{(k)} = L' y^{(k-1)}$$

$$y^{(k)} = L\ x^{(k-1)}$$

where x is the vector of authority scores for each page and y is the vector of hub scores.

This gives an obvious algorithm:

Initialize y to all 1's

Repeat until convergence

$x^{(k)} = L' y^{(k-1)}$

$y^{(k)} = L\ x^{(k)}$

$k = k + 1$

normalize $x^{(k)}$ and $y^{(k)}$

In fact, we can optimize and only compute one:

$x^{(k)} = L' L x^{(k-1)}$ or $y^{(k)} = L L' y^{(k-1)}$

and get the other one via one more mat mult.

How does a query work?

Given a set of query words,

1. retrieve all of the documents that contain them;

2. add documents that point to or are pointed to by the first set of documents (not too many)

3. apply the HITS algorithm to the resulting set

Step 2 helps to find some of the latent connections of the query words.

Step 3 produces two lists of documents ordered by authority and hubbiness and returns them to the user.

Note that the eigenvector computation is applied to a matrix that is fairly dense and fairly small.

Good things about HITS:

1. Users may find one of the ranked lists more useful than the other

2. The algorithm runs on a small dataset

3. The only precomputation is building the inverted indexes, which can be done incrementally

Bad things about HITS:

1. Each query requires a lot of new work: finding the pages, doing an eigenvector computation

2. It's easy for a web page author to increase his/her hub score (by adding links to good pages) which tends to feedback to an increased authority score.

3. A strong, but more or less irrelevant, authority can get picked up as a neighbour and skew the whole set of results

PageRank – the algorithm behind Google

The basic idea: each page gets a rank that determines how much total `vote' it has. Its `vote' is split evenly across all of its outgoing links.

A link from a powerful page gives the destination page a greater rank.

The definition is recursive, so we need to find the fixed point.

So

rank of page P = sum (rank of Q / number of links from Q)

summed over all of the pages that point at P.

If $\pi^{(0)}$ is initialized to $1/n$, then we can compute

$$\pi'^{(k)} = \pi'^{(k-1)} P$$

where P is the matrix containing 0 if page i doesn't link to page j, and 1/(number of outgoing links) if it does.

P is a nonnegative matrix whose rows sum either to 1 or to 0. The second case represents pages with no outgoing links.

π is the dominant left eigenvector of the matrix P and it contains the page rank of every page.

“the world’s largest matrix computation”.

One interesting way to interpret a page’s rank is that it is the proportion of time spent at a web page by a web surfer who clicks on links at random for ever.

There are a number of practical problems with computing the required eigenvector.

First, we need to do something about the rows of all 0's. This is usually done by replacing such rows by rows all of whose entries are 1/n.

A matrix that can be permuted (rows and columns) so that it contains a block of zeroes in the bottom left hand corner is called reducible.

A reducible matrix doesn't necessarily have a single dominant eigenvector – so the pages couldn't be organized into a single ordered list.

A reducible matrix also means that a random clickstream gets trapped in some regions of the Web graph.

Unfortunately, the Web graph is almost certainly reducible.

Some fix is needed to ensure an irreducible matrix.

Google generates a new matrix of the form

$$newP = \alpha P + (1 - \alpha) E$$

where E models users' ability to teleport to other (unlinked pages), and $(1 - \alpha)$ is the probability of this happening.

Originally E described a uniform probability, but now Google allocates each page a probability of being teleported to.

We don’t know exactly how Google chooses to set α or how they set the entries of the teleport vector.

Choosing slightly different values for α gives very different page ranks.

Choosing different teleport vectors also produces different page ranks.

So what Google computes is not the page rank but a page rank – and we don’t know to what extent they knowingly adjust the answers.

The actual recurrence computed is

$$\pi'^{(k)} = \alpha \pi'^{(k-1)} P + (1 - \alpha) v'$$

which:

-- preserves the inherent sparsity

-- spends most time doing $\pi'^{(k-1)} P$ which can be easily parallelized

-- converges after 100 – 150 iterations

Computing the page ranks is a precomputation.

To handle a query, documents containing the query words are retrieved, and then presented in descending order of page rank.

Notice that page rank is an importance ranking, so responses to a query are not the most relevant documents, but the most important relevant documents.

There’s not much work to do for each query, but the amount of work to update the page ranks is huge.

At present, it is only updated every week.

There doesn’t seem to be any easy way to make the algorithm incremental – it’s faster to recompute from scratch than to start with the current page ranks.

Good things about PageRank:

1. importance rather than relevance seems to be a good property from the user perspective

2. it's impervious to manipulation by page authors – it's hard to get other people to create links to your page

3. response time is small because only a simple calculation is needed for each query

4. choosing the teleport vectors gives Google some extra power (e.g. to punish manipulators)

but do they use this power for good or evil?

What's bad about PageRank:

1. it's query independent, so it can't distinguish between a globally authoritative page and a page that's authoritative on the topic of the query

2. the precomputation is an algorithmic bottleneck that limits the freshness of the results

3. choosing the teleport vectors gives Google some extra power

It’s been claimed that PageRank and HITS are quite closely related, but this seems to be only partly true.

Summary:

Web information retrieval is a difficult problem because of scale, access to data, and rate of change.

All algorithms use the idea of an eigenvector as the direction along which to rank pages – but the details differ.