Document retrieval:
A short overview of some old and recent techniques

Marco Saerens (UCL), with
Christine Decaestecker (ULB)
Contents

- General introduction
- Information retrieval: Basic standard techniques (content-based methods)
  - Documents pre-processing
  - Vector-space model
  - Probabilistic model
  - Assessment of performances
- Information retrieval: More recent techniques
  - Exploiting links between documents (web link analysis)
    - The PageRank algorithm
    - The HITS algorithm
  - Exploiting the relational structure in order to improve retrieval
General introduction
Introduction

- We have a collection of documents (mainly text or html-based)
- We have a set of users
- A user wants to retrieve the documents related to a given concept
- He consequently submits a query expressed through words or terms
- An information retrieval system returns the documents most related to this concept
Introduction

- One major problem:
  - We want to express a concept
  - With words
  - There is no one-to-one mapping (e.g. marché)
Documents preprocessing
Documents preprocessing

  - Modern Information Retrieval
  - Addison Wesley
Documents pre-processing

- We have a collection of documents
- Here are the standard pre-processing steps
  - Extract text and structure (eg. from Microsoft Word or LaTeX to XML)
  - Remove stop words (eg. remove "the", "at", "all", etc)
  - Named entity recognition (eg. find proper names)
  - Stemming (eg. extract "process" from "processing"
Documents pre-processing

- It is a tedious job

- But some tools are readily available
  - Galilei project developed at the ULB
Documents pre-processing

- Stemming aims to extract the « root » of the words
- Stemming can be based on
  - A dictionary (for instance Mmorph developed at the University of Geneva)
  - A set of rules developed by linguists (like Porter's stemming algorithm for English)
Documents pre-processing

Example of stemming rules in french:

\( (m > 0) \ aux \rightarrow al \)
\( (m > 0) \ ouse \rightarrow ou \)
\( (m > 0) \ eille \rightarrow eil \)
\( (m > 0) \ nne \rightarrow n \)
\( (m > 0) \ fs \rightarrow v \)
Basic Methods
The vector space model
The vector space method

- M. W. Berry & M. Browne (1999)
  - Understanding Search Engines
  - SIAM
The vector space model: Introduction

- In its basic form, each **document** is represented by a vector
  - A **query** is also represented by a vector
  - A **user profile** may be represented by a vector as well
- The coordinates of the vector are **words**
  - Each element of the vector represents the **frequency of the word** in the document or the query
  - In the **space of words**
The vector space model: Basics

- Thus a **document** is represented by a vector
  - Document $j$ is characterized by $d_j$
  - $f_{ij}$ is the frequency of word $w_i$ in document $j$
  - The total number of words is $n_w$

- The dimension of the vector is $n_w$
The vector space model: Basics

- Thus each document is represented by

\[ \mathbf{d}_j \triangleq \begin{bmatrix} f_{1j} \\ f_{2j} \\ \vdots \\ f_{n_w,j} \end{bmatrix} \]

- This is called the « bag of words » representation in the words space
  - The order of the words is not taken into account
  - This vector is usually sparse
  - This vector is very large
The vector space model: Basics

- The total number of documents is $n_d$
- The term-document matrix is

$$D \triangleq \begin{bmatrix}
  f_{11} & f_{12} & \cdots & f_{1n_d} \\
  f_{21} & f_{22} & \cdots & f_{2n_d} \\
  \vdots & \vdots & \ddots & \vdots \\
  f_{n_w1} & f_{n_w2} & \cdots & f_{n_wn_d}
\end{bmatrix}$$

\{ documents \} \{ words \}
The vector space model: Basics

- A query is also represented by a vector
  - Here is a query $q$
  - Each element is 0 or 1 (presence or absence of a word)

$$q \triangleq \begin{bmatrix}
0 \\
\vdots \\
0 \\
1 \\
0 \\
\vdots \\
0 
\end{bmatrix}$$

word $w_i$ is present in the query
The vector space model: Basics

- The purpose is of course to retrieve documents $d_i$ based on a query $q$
- We have to define a notion of similarity between a query and a document
The vector space model: Basics

- The similarity between a query $q$ and a document $d_i$ can be defined as
  - The cosinus of the angle between these two vectors:
    \[ \text{sim}(q, d_i) \triangleq \cos(q, d_i) = \frac{q^T d_i}{\|q\| \|d_i\|} \]
  - Euclidean distance does not work well because queries contain much lesser words than documents
The vector space model: Basics

- The similarity between the query and all documents can be computed by using the term-document matrix

\[
\cos(q, D) = \frac{q^T}{\|q\|} D \text{diag} \left[ \frac{1}{\|d_i\|} \right] = \frac{q^T}{\|q\|} \left[ \begin{array}{ccc} d_1 & \ldots & d_i & \ldots & d_{n_d} \end{array} \right] \text{diag} \left[ \frac{1}{\|d_i\|} \right] = \left[ \begin{array}{ccc} \frac{q^T d_1}{\|q\|} & \ldots & \frac{q^T d_i}{\|q\|} & \ldots & \frac{q^T d_{n_d}}{\|q\|} \end{array} \right] \text{diag} \left[ \frac{1}{\|d_i\|} \right] = \left[ \begin{array}{ccc} \frac{q^T d_1}{\|q\| \|d_1\|} & \ldots & \frac{q^T d_i}{\|q\| \|d_i\|} & \ldots & \frac{q^T d_{n_d}}{\|q\| \|d_{n_d}\|} \end{array} \right].
\]
The vector space model: Refinements

- Two refinements of the basic model:
  - Term weighting
  - Latent semantic models
The vector space model: Term weighting

- We introduce **term weighting**
  - Of course, each word does not have the same « weight »
  - We would like to take account of the "discriminative power" of every word
  - For instance, if a word is present in every document, it is useless
  - \( P(w_i) \) is the a priori probability that word \( w_i \) appears in a document
The vector space model: Term weighting

- We redefine the query vector $q$ as

$$q = \begin{bmatrix}
0 \\
\vdots \\
-\log_2[P(w_i)] \\
0 \\
\vdots \\
0
\end{bmatrix}$$

- Each word $w_i$ is weighted by the information provided by knowing the presence of the word.
The vector space model: Latent semantic models

- Latent semantic models
  - These models try to capture some semantic information
  - For instance, if we introduce a query with "newborn", it would be nice if documents containing "baby" but not "newborn" are also retrieved
  - We say that words are semantically related when they are used in the same context
The vector space model: Latent semantic models

- This way, we can capture some « semantic similarity » between words
  - In the present case, we will say that two words are semantically related
  - When they often occur in the same document
The vector space model: Latent semantic models

- One solution to this problem is to use "sub-space projection methods" like
  - "Singular Value Decomposition" (SVD) or
  - "factor analysis"

- The rank $m$ SVD of a matrix of rank $n$ is the « best approximation » to this matrix having rank $m < n$
  - In the present case, we use a SVD in order to reduce the rank of the term-document matrix
The vector space model: Latent semantic models

- This allows to reduce the dimensionality of the space by clustering the words that are semantically "similar",
- That is, used in the same documents
- This allows us to build a kind of concept space
The vector space model: Latent semantic models

- Every matrix has a "singular value decomposition":

$$D = U \Sigma V$$

where $U^T U = I$ and $V^T V = I$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_n \end{bmatrix}$$

with $\sigma_1 > \sigma_2 > \ldots > \sigma_n > 0$
The vector space model: Latent semantic models

- If we want the best rank-$m$ approximation to $\mathbf{D}$, we put

$$\sigma_{m+1} = 0, \sigma_{m+2} = 0, \ldots, \sigma_n = 0$$
The vector space model: Latent semantic models

So that we obtain

\[
\tilde{\Sigma} = \begin{bmatrix}
\sigma_1 & 0 & \ldots & \ldots & \ldots & \ldots & 0 \\
0 & \sigma_2 & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & \sigma_m & 0 & \ddots & \vdots \\
\vdots & \vdots & \vdots & \ddots & 0 & 0 & \ddots & \vdots \\
0 & 0 & \ldots & \ldots & \ldots & \ldots & 0 \\
0 & 0 & \ldots & \ldots & \ldots & \ldots & 0
\end{bmatrix}
\]
The vector space model: Latent semantic models

- \( \tilde{D} \) is the best rank-\( m \) approximation to \( D \)
  \[ \tilde{D} = U\Sigma V \]
  - That is, there is no rank-\( m \) matrix closer to \( D \) in terms of the Frobenius norm
- The queries are now adressed to \( \tilde{D} \) instead of \( D \)
  \[ \text{sim}(q, \tilde{d}_i) = \cos(q, \tilde{d}_i) = \frac{q^T\tilde{d}_i}{\|q\|\|\tilde{d}_i\|} \]
The vector space model: Latent semantic models

- But how does it work?
The vector space model: Conclusion

- The vector-space method relies on linear algebra concepts
- The SVD approach allows to work in a latent space representing concepts
- The main problem: How many dimensions of the subspace do we keep?
Basic Methods
Probabilistic methods
Probabilistic methods

- K. Sparck Jones & P. Willett (Editors) (1997)
  - Readings in Information Retrieval
  - Morgan Kaufmann
- Collection of papers
Probabilistic methods: Introduction

- The probabilistic methods rely on statistical models
  - Each user profile is represented by a statistical model

- A document can be relevant or not to a user
  - $R = 1$ if it is relevant; $R = 0$ if it is not relevant
Probabilistic methods: Introduction

- Based on
  - Relevance feedback from the user
  - Or simply the ranking of a vector space model

- We can build a probabilistic model
  - It will estimate the probability that a document is relevant

- We will introduce the **binary independence retrieval model**
Probabilistic methods: Introduction

- We introduced a query
  - Based on a vector space model, we obtain

![Diagram showing relevance and results of a vector model]
Probabilistic methods: Basic model

- Each document $d_i$ is represented by a binary vector

$$d_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

This word is present in the document

- $[d_i]_j = 1$ if word $w_j$ is in document $d_i$
- $[d_i]_j = 0$ if word $w_j$ is not in document $d_i$
Probabilistic methods: Basic model

- Based on ranking, some documents are considered as relevant ($R = 1$)
- And some documents are considered as not relevant ($R = 0$)
- To a user $u_k$
Probabilistic methods: Basic model

- We define \( P(d = x| R = 1, u_k) \)
  - as the probability of observing a document \( d = x \) given that this document is relevant for user \( u_k \)

- We will see that it is easy to estimate these probabilities for the binary independence model
Probabilistic methods: Basic model

- However, during the document retrieval phase, we are mainly interested in:
  \[ P(R = 1|d = x, u_k) \]
- The larger this value, the more likely the document \( x \) is relevant
- This probability has to be computed for each document in the database
Probabilistic methods: Basic model

- Now, instead of computing
  \[ P(R = 1 | d = x, u_k) \]

- It is easier to compute the odds
  \[
  \lambda = \frac{P(R = 1 | d = x, u_k)}{P(R = 0 | d = x, u_k)}
  \]
  \[
  = \frac{P(R = 1 | d = x, u_k)}{1 - P(R = 1 | d = x, u_k)}
  \]
Probabilistic methods: Basic model

- It is a monotonic increasing function of
  $$P(R = 1|d = x, u_k)$$
- It therefore provides the same ranking
- The larger this value $\lambda$, the more likely the document is relevant
Probabilistic methods: Basic model

- Remember Bayes' law!

\[
P(R = 1 | d = x, u_k) = \frac{P(d = x | R = 1, u_k) P(R = 1 | u_k)}{P(d = x | u_k)}
\]

\[
P(R = 0 | d = x, u_k) = \frac{P(d = x | R = 0, u_k) P(R = 0 | u_k)}{P(d = x | u_k)}
\]
Probabilistic methods: Basic model

- We can easily compute $\lambda$ by assuming conditional independence between the words ($d_n$ is element $n$ of vector $d$)

$$
\lambda = \frac{P(R = 1 | d = x, u_k)}{P(R = 0 | d = x, u_k)} = \frac{P(d = x | R = 1, u_k)}{P(d = x | R = 0, u_k)} \times \frac{P(R = 1 | u_k)}{P(R = 0 | u_k)}
$$

$$
= \frac{\prod_{n=1}^{n_w} P(d_n = x_n | R = 1, u_k)}{\prod_{n=1}^{n_w} P(d_n = x_n | R = 0, u_k)} \times \frac{P(R = 1 | u_k)}{P(R = 0 | u_k)}
$$
Probabilistic methods: Basic model

- And finally $\lambda$ is proportional to

$$
\lambda \propto \frac{\prod_{n=1}^{n_w} P(d_n = x_n | R = 1, u_k)}{\prod_{n=1}^{n_w} P(d_n = x_n | R = 0, u_k)}
$$

- This is really a **naive Bayes classifier**
- The $P(d_n = x_n | R = 1, u_k), P(d_n = x_n | R = 0, u_k)$ are **easy to compute**
  - = Likelihoods estimated by frequencies
Probabilistic methods: Basic model

- The $P(d_n = x_n | R = 1, u_k), P(d_n = x_n | R = 0, u_k)$ are easy to compute
  - Likelihoods estimated by frequencies

- They are estimated by the proportion of documents containing the word $w_n$ among relevant/irrelevant documents
Probabilistic methods: Conclusion

- The binary independence probabilistic retrieval model makes strong assumptions about independence of word occurrence.
- More sophisticated models are available:
  - For instance, Poisson models can be used in order to take account of the number of words appearing in the document.
  - We can also take account of second-order interactions between words (correlations).
Assessment of documents retrieval systems

- In general, we compute two measures:
  - The precision
  - The recall
Assessment of documents retrieval systems

- The precision measure estimates the percentage of relevant retrieved documents in the set of all retrieved documents
  - Precision indicates to which extent the retrieved documents are indeed relevant
Assessment of documents retrieval systems

- The recall measure estimates the percentage of relevant retrieved documents in the set of all relevant documents
  - Recall indicates to which extend the relevant documents are indeed retrieved
Cutting a graph in small pieces and exploring it
Cutting a graph in small pieces and exploring it

- A nice reference
  - Sedgewick
  - Algorithms in Java
  - Addison-Wesley
Cutting a graph in small pieces and exploring it

- Many **software solutions** are readily available for cutting a graph
  - See www.insna.org/INSNA/soft_inf.html
  - It provides a number of pointers to softwares
Cutting a graph in small pieces and exploring it

- **JUNG**
  - the Java Universal Network/Graph Framework
  - Open source and written in Java
  - Mainly a **toolbox** of methods
  - [http://jung.sourceforge.net](http://jung.sourceforge.net)
Cutting a graph in small pieces and exploring it

- Pajek
  - A software for large network analysis
  - There is a companion book
  - Quite powerful
  - Free software
Cutting a graph in small pieces and exploring it

- UCINET
  - Social Network Analysis Software
  - Written by active researchers in the social network community
  - Quite complete commercial software
  - http://www.analytictech.com
Cutting a graph in small pieces and exploring it

- The first step in analysing a graph is often to look at its connected components
  - In an undirected graph, a connected component is a maximal connected subgraph
  - A strongly connected component is the similar concept defined for directed graphs
Cutting a graph in small pieces and exploring it

- Here is a graph with two connected components
Cutting a graph in small pieces and exploring it

- There exists linear-time algorithms for solving this problem
  - Using, for instance, depth-first or breadth-first search
Cutting a graph in small pieces and exploring it

- An **articulation point** or **vertex-cut** is a node (vertex) of a graph such that
  - removal of the node causes an increase in the number of **connected components**
Cutting a graph in small pieces and exploring it

- Standard linear-time algorithms are available for this problem, using, for instance
  - depth-first or breadth-first search
Cutting a graph in small pieces and exploring it

- A bridge or an edge-cut is an arc (edge) of a graph such that
  - removal of the arc causes an increase in the number of connected components
Cutting a graph in small pieces and exploring it

- Standard linear-time algorithms are available for this problem, using, for instance
  - depth-first or breadth-first search
Identifying central or prestigious nodes by link analysis
Some link analysis books

  - Modeling the Internet and the Web
  - John Wiley & Sons
Some link analysis books

- S. Chakrabarti (2003)
  - Mining the Web
  - Morgan Kaufmann
Some link analysis books

  - Google’s PageRank and beyond
  - Princeton university
Some link analysis books

- B. Liu (2006)
  - Web data mining
  - Springer
Identifying central or prestigious nodes by link analysis

- The PageRank algorithm
- The HITS algorithm
- The SALSA algorithm
The PageRank algorithm
The basic PageRank algorithm

- Introduced by Page, Brin, Motwani & Winograd in 1998
- Partly implemented in Google
- Corresponds to a measure of «prestige» in a directed graph
Web link analysis

- A set of techniques
  - Applied to: Hyperlink document repositories
  - Typically web pages

- Objective:
  - To exploit the link structure of the documents
  - In order to extract interesting information
  - Viewing the document repository as a graph where
    - Nodes are documents
    - Edges are directed links between documents
  - It does not exploit the content of the pages !!
Web link analysis

- Suppose we performed a search with a search engine
- **Objective**: to improve the (content-based) ranking of the search engine
  - Based on the graph structure of the web hyperlinks
  - PageRank is computed off-line
The basic PageRank algorithm

- To each web page we associate a score, $x_i$
  - The score of page $i$, $x_i$, is proportional to the weighted averaged score of the pages pointing to page $i$
The basic PageRank algorithm

- Let $w_{ij}$ be the weight of the link connecting page $i$ to page $j$
  - Usually, it is simply 0 or 1
  - Thus, $w_{ij} = 1$ if page $i$ has a link to page $j$; $w_{ij} = 0$ otherwise

- Let $W$ be the matrix made of the elements $w_{ij}$
  - Notice that this matrix is not symmetric
  - We suppose that the graph is strongly connected
The basic PageRank algorithm

- In other words

\[ x_i \propto \sum_{j=1}^{n} \frac{w_{ji} x_j}{w_j} \]

\[ w_j = \sum_{i=1}^{n} w_{ji} \]

- Where \( w_j \) is the outdegree of page \( j \)
The basic PageRank algorithm

- In other words,
  - A page with a high score is a page that is pointed by
    - Many pages
    - Having each a high score
  - Thus a page is an important page if
    - It is pointed by many, important, pages
The basic PageRank algorithm

- These equations can be updated iteratively until convergence
- In order to obtain the scores, $x_i$
  - We normalize the vector $x$ at each iteration
- The pages are then ranked according to their score
The basic PageRank algorithm

- This definition has a nice interpretation in terms of random surfing
- If we define the probability of following the link from page \( j \) to page \( i \) as

\[
P(\text{page}(k+1) = i | \text{page}(k) = j) = \frac{w_{ji}}{w_j}.
\]

\[
w_j. = \sum_{i=1}^{n} w_{ji}
\]
The basic PageRank algorithm

- We can write the updating equation as

\[ x_i(k + 1) = P(page(k + 1) = i) \]
\[ = \sum_{j=1}^{n} P(page(k + 1) = i | page(k) = j) x_j(k) \]

- And thus we can define a random surfer following the links according to the transition probabilities

\[ P(page(k + 1) = i | page(k) = j) = \frac{w_{ji}}{w_j} \]
The basic PageRank algorithm

- This is the equation of a Markov model of random surf through the web.
- This is exactly the same equation as before:

\[ x_i \propto \sum_{j=1}^{n} \frac{w_{ji} x_j}{w_j}. \]

\[ w_j. = \sum_{i=1}^{n} w_{ji} \]
The basic PageRank algorithm

- If we denote element $i, j$ of the transition probability matrix $P$ as $p_{ij}$
- We thus have

$$[P]_{ij} = p_{ij} = P(\text{page}(k + 1) = j | \text{page}(k) = i)$$

- And the equation can be rewritten as

$$x_i(k + 1) = \sum_{j=1}^{n} p_{ji} x_j(k)$$
The basic PageRank algorithm

- In matrix form, if the vector $\mathbf{x}$ has elements $x_i$

  $$\mathbf{x}(k + 1) = \mathbf{P}^T \mathbf{x}(k)$$

- The stationary distribution is given by

  $$\mathbf{x}(k+1) = \mathbf{x}(k), \text{ and thus}$$

  $$\mathbf{x} = \mathbf{P}^T \mathbf{x}$$
The basic PageRank algorithm

- $x_i$ can then be viewed as the probability of being at page $i$
  - The solution to these equations is the stationary distribution of the random surf
  - Which is the probability of finding the surfer on page $i$ on the long-term behaviour
- The « most probable page » is the best ranked
The basic PageRank algorithm

- The PageRank scores can be obtained
  - By computing the left eigenvector of the matrix $P$ corresponding to eigenvalue 1
  - Which is the right eigenvector of $P^T$
  - Where $P$ is the transition probabilities matrix of the Markov process
  - Containing the transition probabilities

- If the graph is undirected, the scores are simply the indegrees of the nodes
Adjustments to the basic model

- However, there is a problem with
  - Dangling nodes
  - That is, nodes without any outgoing link
- In this case, the $\mathbf{P}$ matrix is no more stochastic
  - Rows do not sum to one
- Moreover, the graph could have separate components
Adjustments to the basic model

- One potential solution is to allow to jump to any node of the graph
  - With some non-zero probability (teleportation)
- Thus,

\[
G = \alpha P + (1 - \alpha) \frac{ee^T}{n}
\]

where \( G \) is called the Google matrix, \( e \) is a column vector full of 1’s and \( 0 < \alpha < 1 \)
Adjustments to the basic model

- In this case,
  - The matrix is stochastic
  - The matrix is irreducible (no separate component)
  - The matrix is aperiodic

- Then, there is a unique eigenvector associated to eigenvalue 1

- However, $G$ is no more sparse!
Computing PageRank

- The problem is thus to find the **left eigenvector** of $G$
  - corresponding to the eigenvalue 1
  - instead of $P$

$$x^T G = x^T$$

- with the normalization $\|x\|_1 = 1$ or $x^T e = 1$

- One can use the standard **power method**
Computing PageRank

Fortunately, the power method results in sparse matrix multiplication only:

$$x^T(k+1) = x^T(k)G$$

$$= \alpha x^T(k)P + \frac{(1-\alpha)}{n}x^T(k)e e^T$$

$$= \alpha x^T(k)P + \frac{(1-\alpha)}{n}e^T$$

– where $x$ is normalized after each iteration
Personalization in PageRank

- How can we favour some pages in a natural way (advertising, etc)?

- Rather than using $ee^T/n$,

- use $ev^T$ where
  - $v > 0$ is a probability vector ($v^Te = 1$)
  - Which is called the personalization vector

- It contains the a priori probability of jumping to any page
Personalization in PageRank

- The Google matrix thus becomes

\[ G = \alpha P + (1 - \alpha) e \mathbf{v}^T \]

- Where \( \mathbf{v} \) is provided a priori
The PageRank problem as a sparse linear system

Here is an alternative formulation of the PageRank problem

- As a sparse linear system

\[ \mathbf{x}^T \mathbf{G} = \mathbf{x}^T \]
\[ \Rightarrow \mathbf{x}^T (\alpha \mathbf{P} + (1 - \alpha) \mathbf{e} \mathbf{v}^T) = \mathbf{x}^T \]
\[ \Rightarrow \alpha \mathbf{x}^T \mathbf{P} + (1 - \alpha) \mathbf{v}^T = \mathbf{x}^T \]
\[ \Rightarrow \mathbf{x}^T (\mathbf{I} - \alpha \mathbf{P}) = (1 - \alpha) \mathbf{v}^T \]
\[ \Rightarrow (\mathbf{I} - \alpha \mathbf{P})^T \mathbf{x} = (1 - \alpha) \mathbf{v} \]
The PageRank problem as a sparse linear system

- In other words, the problem is

\[ \text{Solve } (\mathbf{I} - \alpha \mathbf{P})^T \mathbf{x} = (1 - \alpha) \mathbf{v} \text{ with } \mathbf{x}^T \mathbf{e} = 1 \]

- Which has been shown (Del Corso, Gulli & Romani, 2005; Langville & Meyer, 2006) to be equivalent to

\[ \text{Solve } (\mathbf{I} - \alpha \mathbf{P})^T \mathbf{x}' = \mathbf{v} \text{ and compute } \mathbf{x} = \mathbf{x}' / \|\mathbf{x}'\|_1 \]
The HITS algorithm
The HITS algorithm

- Introduced by Kleinberg in 1998/1999
Web link analysis

- Suppose we performed a search with a search engine
  - Compute the neighborhood graph from the retrieved documents
  - Associated to the particular query

- **Objective**: to improve the ranking provided by the search engine
  - Based on the graph structure of the web hyperlinks
The HITS algorithm

- The model proposed by Kleinberg is based on two concepts
  - **Hub** pages
  - **Authorities** pages

- These are two categories of web pages

- These two concepts are strongly connected
The HITS algorithm

- Example:
  - Suppose we introduced the query “Car constructors”
The HITS algorithm

- **Hubs**
  - Link heavily to authorities
  - A good hub points to many good authorities
  - Hubs have very few incoming links

![Diagram showing Hubs and Authorities](image-url)
The HITS algorithm

- **Authorities**
  - Do not link to other authorities
  - A good authority is pointed by many good hubs
  - The main authorities on a topic are often in competition with one another
The HITS algorithm

- The objective is to detect **good hubs** and **good authorities**
  - from the results of the search engine
- We therefore assign two numbers to each returned page $i$:
  - A **hub score**, $x^h_i$
  - An **authority score**, $x^a_i$
The HITS algorithm

- Let $w_{ij}$ be the weight of the link connecting page $i$ to page $j$
  - Usually, it is simply 0 or 1
  - Thus, $w_{ij} = 1$ if page $i$ has a link to page $j$; $w_{ij} = 0$ otherwise

- Let $W$ be the matrix made of elements $w_{ij}$
  - Notice that this matrix is not symmetric
  - We suppose that the graph is strongly connected
The HITS algorithm

- A possible procedure for computing hub/authorities scores (Kleinberg)

  - A page's **authority score** is proportional to the sum of the hub scores that link to it
    \[
    x_j^a = \eta \sum_{i=1}^{n} w_{ij} x_i^h
    \]

  - A page's **hub score** is itself proportional to the sum of the authority scores that it links to
    \[
    x_i^h = \mu \sum_{j=1}^{n} w_{ij} x_j^a
    \]
The HITS algorithm

- In matrix form,

\[
\begin{align*}
\mathbf{x}^a &= \eta \mathbf{W}^T \mathbf{x}^h \\
\mathbf{x}^h &= \mu \mathbf{W} \mathbf{x}^a
\end{align*}
\]

- And thus,

\[
\begin{align*}
\mathbf{x}^a &= \eta \mu \mathbf{W}^T \mathbf{W} \mathbf{x}^a \\
\mathbf{x}^h &= \eta \mu \mathbf{W} \mathbf{W}^T \mathbf{x}^h
\end{align*}
\]
The HITS algorithm

- Kleinberg used this iterative procedure in order to estimate the scores
  - with a normalization at each step
  - This is equivalent to computing the eigenvectors of the following matrices

\[ \mathbf{WW}^T \quad \mathbf{WTW} \]

- To obtain respectively the vector of hubs scores and the vector of authorities scores
Links with principal components analysis

- This is exactly uncentered principal components analysis (PCA; Saerens et al., 2005)
  - The proof is based on the dual view of PCA

- As for multidimensional scaling
  - View the set of rows of $W$ (authorities) as a cloud of points in the columns space
  - View the set of columns of $W$ (hubs) as a cloud of points in the rows space
Links with principal components analysis

- Let us consider a data matrix $X$
- The first PCA axis on which the data will be projected is given by the eigensystem
  \[ X^T X u_1 = \lambda_1 u_1 \]
- Thus, the first projection axis correspond to the dominant eigenvector of
  \[ X^T X \]
Links with principal components analysis

- Then, the first coordinate of the projected data is given by $Xu_1$
  - Which corresponds to the data vectors projected on the first principal axis, $u_1$

- These are the PCA scores for the first principal axis
Links with principal components analysis

- Here is a sketch of the proof that HITS is equivalent to uncentered PCA
- Let us consider the adjacency matrix $W$ as a data matrix
  \[ X = W \]
- We simply substitute $X$ by $W$ for computing the first principal axis (PCA), $u_1$
  \[ W^T W u_1 = \lambda_1 u_1 \]
Links with principal components analysis

- We pre-multiply this equation by $W$

$$WW^T(Wu_1) = \lambda_1(Wu_1)$$

- $Wu_1$ is an eigenvector of $WW^T$ and thus contains the hubs scores

- Since $Wu_1$ is the projection of the data on the first principal axis (= PCA scores)

- The hubs scores are equal to the uncentered PCA scores, up to a proportionality factor, computed from the data matrix $W$
Links with principal components analysis

- The same result holds for the authorities scores
- We now consider the transposed adjacency matrix $W^T$ as a data matrix

$$X = W^T$$

- And proceed as before
- The authorities scores are equal to the uncentered PCA scores, up to a proportionality factor, computed from the data matrix $W^T$
Links with principal components analysis

- Thus, the situation is exactly the same as for multidimensional scaling
  - The first eigenvector of $\mathbf{W}\mathbf{W}^T$ represents the projection of the row vectors on the first principal component (hubs scores)
  - The first eigenvector of $\mathbf{W}^T\mathbf{W}$ represents the projection of the column vectors on the first principal component (authority scores)

- This procedure is also related to both
  - Correspondence analysis
  - A random walk (Markov) model through the graph
HITS’ relationships to bibliometrics

- The HITS algorithm has also strong connections to bibliometrics research:
  - Cocitation
  - Coreference

- **Cocitation** occurs when two documents are both cited by the same third document

- **Coreference** occurs when two documents both refer to the same third document
HITS’ relationships to bibliometrics

C. Ding (2002) showed that

\[ W^T W = D_{in} + C_{cit} \]
\[ WW^T = D_{out} + C_{ref} \]

- where \( D_{in} \) is a diagonal matrix containing the indegree of each node \( D_{in} = \text{Diag}(w_{i,j}) \)
- \( D_{out} \) is a diagonal matrix containing the outdegree of each node \( D_{out} = \text{Diag}(w_{i,j}) \)
- \( C_{cit} \) and \( C_{ref} \) are the cocitation and coreference matrices
HITS’ relationships to bibliometrics

Thus,

– The **hub matrix** is closely related to the **coreference matrix**

– The **authority matrix** is closely related to the **cocitation matrix**
The SALSA algorithm
The SALSA algorithm

- Introduced by Lempel & Moran in 2000
  - « A Stochastic Approach to Link Structure Analysis »
  - Combines ideas from PageRank and HITS
The SALSA algorithm

- From the neighborhood graph, compute two sets of nodes
  - The hub nodes
  - The authority nodes
  - View this as a bipartite graph

![Diagram showing a bipartite graph with hubs and authorities]
The SALSA algorithm

- From this bipartite graph, compute a Markov chain with
  - $P^h = (D_{in})^{-1}W^T$: the probability of jumping from an authority node to a hub node
  - $P^a = (D_{out})^{-1}W$: the probability of jumping from a hub node to an authority node

Thus:

$$x^h(k+1) = (P^h)^T x^a(k)$$
$$x^a(k+1) = (P^a)^T x^h(k)$$

where $x^h, x^a$ are the probability distributions for hub and authority nodes
The SALSA algorithm

- The transition probabilities matrix of the Markov chain

  - Restricted to the authority nodes is $P_h P_a$

  - Restricted to the hub nodes is $P_a P_h$
The SALSA algorithm

- The **steady-state probability distribution** of the two restricted Markov chains are the hub scores and authority scores
  - When removing dangling nodes
  - When computing the steady-state for each connected component
Collaborative recommendation
Collaborative recommendation
Collaborative recommendation

- Suppose we have tables connected by relationships
  - Example: A movie database
Collaborative recommendation

- Computing similarities between people and movies allows to suggest movies to watch or not to watch

= Collaborative recommendation

- Based on historical data (purchases)
Collaborative recommendation

- Most popular algorithms based on a bipartite graph:
  - Individuals-based nearest neighbours algorithms
- We thus have individuals and items (the movies)
  - Each individual $i$ is represented by a vector $v_i$ in the item space
  - Containing, as elements, 1 if individual $i$ purchased the item, and 0 otherwise
  - We call this vector the individuals’ profile vector
Collaborative recommendation

- We also compute the individuals–items frequency matrix $A$
  - Containing as element $a_{ij}$
  - Which represents the number of times individual $i$ purchased item $j$
Collaborative recommendation

- We first compute a similarity between two individuals $i$ and $j$ on the basis of their profile vectors.
- There is a large choice of such similarities
  - Suppose $a$, $b$, $c$, and $d$ are defined by

<table>
<thead>
<tr>
<th>Individual $i$</th>
<th>Individual $j$ 1</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual i</td>
<td>$a$</td>
<td>$a+b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$c+d$</td>
</tr>
<tr>
<td></td>
<td>$d$</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td>$a+c$</td>
<td>$p = a+b+c+d$</td>
</tr>
<tr>
<td></td>
<td>$b+d$</td>
<td></td>
</tr>
</tbody>
</table>
Collaborative recommendation

- Counting the number of 1’s and 0’s in common
  - Here are some popular choices:

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{a + d}{p} )</td>
<td>Equal weights for 1-1 matches and 0-0 matches.</td>
</tr>
<tr>
<td>( \frac{2(a + d)}{2(a + d) + b + c} )</td>
<td>Double weight for 1-1 matches and 0-0 matches.</td>
</tr>
<tr>
<td>( \frac{a + d}{a + d + 2(b + c)} )</td>
<td>Double weight for unmatched pairs.</td>
</tr>
<tr>
<td>( \frac{a}{p} )</td>
<td>No 0-0 matches in numerator.</td>
</tr>
<tr>
<td>( \frac{a}{a + b + c} )</td>
<td>No 0-0 matches in numerator or denominator. (The 0-0 matches are treated as irrelevant).</td>
</tr>
<tr>
<td>( \frac{2a}{2a + b + c} )</td>
<td>No 0-0 matches in numerator or denominator. Double weight for 1-1 matches.</td>
</tr>
<tr>
<td>( \frac{a}{a + 2(b + c)} )</td>
<td>No 0-0 matches in numerator or denominator. Double weight for unmatched pairs.</td>
</tr>
<tr>
<td>( \frac{a}{b + c} )</td>
<td>Ratio of matches to mismatches with 0-0 matches excluded.</td>
</tr>
</tbody>
</table>
Collaborative recommendation

- A good choice is for instance
  \[ \text{sim}(i, j) = \frac{2(a + d)}{2(a + d) + b + c} \]

- Or the cosine distance
  \[ \text{sim}(i, j) = \frac{(v_i^T v_j)}{\|v_i\| \|v_j\|} \]

  where \( v_i \) is a binary vector containing the items individual \( i \) purchased (or not)
Collaborative recommendation

- From this similarity measure between individuals, we compute the $k$ nearest neighbours (KNN) of individual $i$.
  - These are the individuals having similar tastes.

- From these nearest neighbours, we compute some « predicted values » for the items.
  - The first items proposed to individual $i$ are the ones with the highest predicted value.
Collaborative recommendation

The predicted value of item $j$ for individual $i$ is computed as the number of times item $j$ has been purchased by $i$'s neighbours

- Weighted by the similarities between the individuals

$$pred(i, j) = \frac{\sum_{p=1}^{k} sim(i, p) \cdot a_{pj}}{\sum_{p=1}^{k} sim(i, p)}$$

where $a_{pj}$ is an element of the individuals-items frequency matrix $A$ and we keep only the $k$ nearest neighbours
Performance evaluation

- For the movie database:
- We delete links to some watched movies in the training set
  - These deleted watched movies form a test set
- The algorithm has to predict these watched movies (test set)
  - Indeed, the watched movies should be well ranked!
Performance evaluation

- Then, use the standard information retrieval performance indicators:
  - Recall
  - Precision
- Always compare to a ranking algorithm based on maximum frequency
Collaborative recommendation: some extensions

- More general similarity measures are developed
  - Computing similarities between elements of remote tables

- One can also develop item-based algorithms
  - Which are based on items nearest neighbours
  - Items-based nearest neighbours