10. WORD EMBEDDINGS

Word Context

The neighborhood of a word expresses a lot about its meaning

- "You shall know a word by the company it keeps" (J. R. Firth 1957)

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Word: w = banking

Context: C(w) = {government, debt, problems, turning, into, crises, as, has, happened, in}

Word-Context occurrence: (w, c), c ∈ C(w)

With latent semantic indexing we have exploited the idea that words that occur together in documents have a likelihood to have also some shared meaning. This idea is actually quite old, and has been already expressed, for example, by Firth in 1957. We will now exploit this idea in a slightly different way, by focusing on a much smaller context of a word than its document. We will just consider the local neighborhood of the work within a phrase. Typically we will consider all the neighborhoods that we can find in all documents of a large document collection.

For each word w we can identify its neighboring words, which will call its context and denote by C(W). The context can be determined by choosing a certain number of preceding and succeeding words, e.g. a typical number used is 5.

Similarity-Based Representation

One of the most successful ideas in natural language processing

- Two words are considered as similar, when they have similar contexts
- Context captures both syntactic and semantic similarity
  - Syntactic: e.g. king – kings
  - Semantic: e.g. king – queen
- Implicitly used with the term-document matrix M
  - Less localized context
  - No distinction between context and word

This idea is one of the main ideas used (successfully) in natural language processing. The neighborhood captures not only semantic relationships among words (as would also co-occurrence in the same document).It can at the same time also capture syntactic relationships. This is a main difference to methods based on document co-occurrence like LSI.

A second important difference is that methods based on this idea distinguish between a word occurring as the center of a context and as a member of context. For example, it is very unlikely the word “dog” would have in its context a second occurrence of the word “dog”. Thus dog as a “context word” needs to be treated differently from dog as the word of interest. This distinction is not made in purely co-occurrence based methods.

Idea: Word Embeddings

Try to model how likely a word and a context occur together

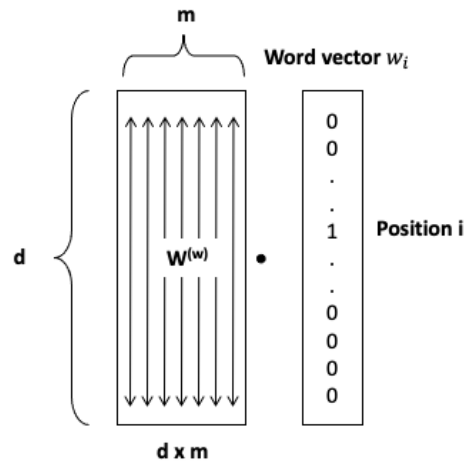
Approach:

- Map words into a low-dimensional space (e.g., d=200)
- Map context words into the same low-dimensional space
  - A different mapping as before for the same words
- Interpret the vector distance (product) as a measure for how likely the word and its context occur together
- Due to projection in low-dimensional space, similar semantically words and contexts should be close

The basic idea for word embeddings is simple. Both words and context words are mapped into the same low-dimensional space. Their representations in that space should be similar, if the word and the context word occur often together. Thus we capture the information on word context in a low-dimensional representation. Through the dimensionality reduction the expectation is that words and contexts that play similar roles would also move together, and thus we could capture syntactic and semantic similarity.

Dimensionality reduction is helpful, since vocabularies can become very large, in particular if not only words but also short phrases are included (e.g. tens of millions of entries). Thus data would be very sparse in these spaces and it would be difficult to model similarity. The low-dimensional spaces have on the other hand typically a few hundred dimensions.

Overview of the Model



Columns represent words of the vocabulary in concept space of dimension d

Mapping a word w_i

$$\boldsymbol{w}_i = W^{(w)} w_i$$

column i in $W^{(w)}$

Mapping a context word c_i

$$\boldsymbol{c}_i = W^{(c)} c_i$$

column i in $W^{(c)}$

Parameters of model θ

All coefficients of $W^{(w)}$ and $W^{(c)}$

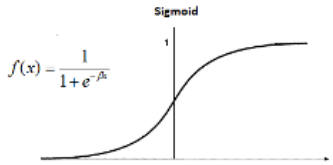
We introduce the basic notations we will use in the following. Note that the size of the vocabulary m is typically very large (e.g. millions), whereas the size of the low-dimensional space d is typically in the hundreds. The model will be fully specified by the two matrices $W^{(w)}$ and $W^{(c)}$ used to map words and context words. For convenience we will denote in the following the coefficients of these two matrices by θ.

Learning the Model from the Data

Consider a pair (w, c)
Does it origin from the data?

Model this as a probability (sigmoid)

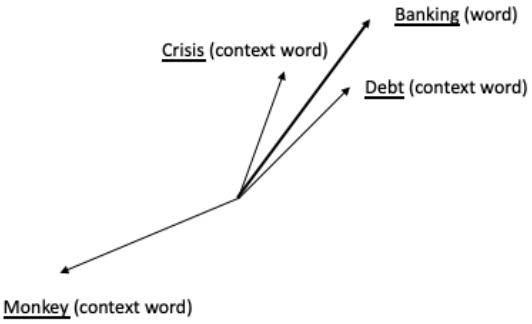
$$P(D = 1|w, c, \theta) = \frac{1}{1 + e^{-\boldsymbol{c}\cdot\boldsymbol{w}}} = \sigma(\boldsymbol{c} \cdot \boldsymbol{w})$$



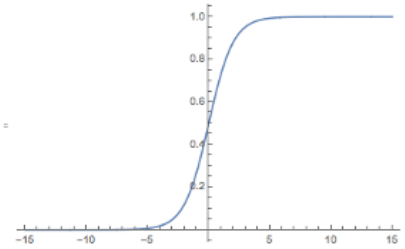
Intuition: convert similarity value in vector space into a probability

Differently to LSI, where the (parameters of the) model have been derived by a mathematical operation (SVD), the idea in word embeddings is to learn the parameters directly from the data. To start with, we consider a word-context pair (w,c) and ask the question whether it comes from the data. We would like that (ideally) the model gives us a probability of 1 if this is the case. In order to derive a probability from the model (the embedding into a low-dimensional space), we proceed as follows: we first take the scalar product of the embedded word and the context word vectors (this is where the parameters are hidden). This product will be large positive when the vectors are similar and large negative when they are opposite. In order to turn those values into a quantity that can be interpreted as probability (i.e. a value in [0,1]) we apply the sigmoid function It produces values close to 0 for large negative arguments, and values close to 1 for large positive arguments. The use of the sigmoid function in this model is for exactly the same reasons as it is for its use in binary logistic regression, converting a similarity value into a probability. Apart from mapping the values to [0,1] it also is insensitive to large outliers.

Illustration of Word Embedding



Banking • Crisis = + 100 σ(Banking • Crisis)≈1
Banking • Debt= + 80 σ(Banking • Debt)≈1
Banking • Monkey = -150 σ(Banking • Monkey)≈0



Problem: finding θ

Here we illustrate the embedding of words and context words in the same low-dimensional space. The idea is that the context words that typically co-occur together with a word, have similar embedding vectors, whereas others (like monkey, which is rarely occurring together with bank) have very different ones. The sigmoid function then converts the similarity values obtained from the scalar products of the embedding vectors into the interval [0,1].

Finding Parameters

Formulate an optimization problem:

- Assume we have positive examples D for (w, c), as well as negative examples $\widetilde{D}$, not occurring in the document collection
- Find θ such that the overall probability is maximized

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\epsilon D} P(D = 1|w, c, \theta) \prod_{(w,c)\epsilon \widetilde{D}} P(D = 0|w, c, \theta) =$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\epsilon D} \log \sigma(\boldsymbol{c} \cdot \boldsymbol{w}) + \sum_{(w,c)\epsilon \widetilde{D}} \log \sigma(-\boldsymbol{c} \cdot \boldsymbol{w})$$

Now we assume that we do not only know positive examples of word-context occurrences, but also negative ones, i.e. pairs that do not occur together. We can similarly model this case as a probability $P(D = 0|w, c, \theta)$. If we have now a (big) collection of positive and negative samples D and $\widetilde{D}$, the overall probability of such a collection to occur can be expressed as the product of all individual probabilities. Given this overall probability, the problem we would like to solve is to find parameters θ that maximize this probability. This is then the best model we can obtain under the assumptions that have been made.

Loss Function

Define a loss function (to be minimized)

$$J(\theta) = \frac{1}{s}\sum_{t=1}^{s} J_t(\theta)$$

$$J_t(\theta) = -\log\big(\sigma(\boldsymbol{c} \cdot \boldsymbol{w}_t)\big) - \sum_{k=1}^{K} \log(\, \sigma(-\boldsymbol{c}_k \cdot \boldsymbol{w}_t))$$

$\boldsymbol{c}_k$ are negative examples of context words taken from a set $P_n(w_t)$

Following the previous derivation we can define a loss function J that optimizes the probabilities if minimized. (We transform the maximization problem into a minimization problem by defining a loss function $J(\theta)$) The loss function is defined over all terms in the vocabulary. For each term w it consists of a first component corresponding to a positive example, a (w,c) pair that occurs in the document collection, and a couple of negative examples for the same word. The question is how to obtain the negative examples.

Obtaining Negative Samples

Negative samples are taken from

$$P_n(w) = V \setminus C(w)$$

Empirical approach

- If p_w is the probability of word w in collection, choose the word with probability $p_w^{3/4}$
- Less frequent words are sampled more often
- Practically: approximate the probability by sampling a few non-context words

The method for learning the model relies on having negative samples. How to obtain them? We can choose any word from the vocabulary V that is not contained itself in the context. In order to model the occurrences we do this by considering the frequency at which the words occur in the document collection. Doing so, experience shows that high frequency words (e.g. stopwords) would however be favored to much, reducing the quality of the model. Thus the probability of word occurrence used for sampling is moderated by an exponent (in practice ¾). Also for obtaining the statistics on word probabilities for practical purposes not the whole data collection is used, but just a small sample of words not occurring in the context.

Stochastic Gradient Descent

For every $(w_t, c) \in D$, update θ

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta^{old})$$

For J_t updates only affect rows in $W^{(w)}$ and $W^{(c)}$ that correspond to $\boldsymbol{w_t}$, $\boldsymbol{c}$ and $\boldsymbol{c_k}$, e.g.,

$$\boldsymbol{w_t^{new}} = \boldsymbol{w_t^{old}} - \alpha \frac{\partial}{\partial \boldsymbol{w_t}} J_t(\boldsymbol{w_t^{old}}, \boldsymbol{c^{old}}, \boldsymbol{c_1^{old}}, \dots, \boldsymbol{c_K^{old}})$$

Finally, given the loss function, finding the optimal parameters θ can then be done using standard methods from machine learning. Then θ can be determined using gradient descent, i.e. standard search for minima using derivatives. More precisely, we will apply stochastic gradient descent. We update the parameters for each sample, separatly. Standard gradient descent would update the values of data only after having seen all samples, which would take extremely long given the very large number of samples. When stochastic gradient descent is used, only rows that contain the word or some context word in the loss function need to be updated. This implies that the data has to be organized that these rows can be efficiently accessed (e.g. using hashing).

Computing the Derivative

Some notation

$$x = \mathbf{c} \cdot \boldsymbol{w}, \mathrm{p} = \sigma(x), z_k = -\mathbf{c}_k \cdot \boldsymbol{w}, q_k = \sigma(z_k)$$

Then $$J_t(\boldsymbol{w}, \mathbf{c}, \mathbf{c}_1, \ldots, \mathbf{c}_k) = -\log(p) - \sum_{k=1}^{K} \log q_k$$

$$\frac{\partial J_t}{\partial \boldsymbol{w}} = \frac{\partial J_t}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial \boldsymbol{w}} + \sum_{k=1}^{K} \frac{\partial J_t}{\partial q_k}\frac{\partial q_k}{\partial z_k}\frac{\partial z_k}{\partial \boldsymbol{w}}$$

$$\frac{\partial J_t}{\partial p} = -\frac{1}{p}, \frac{\partial p}{\partial x} = \mathrm{p}(1-\mathrm{p}), \frac{\partial x}{\partial \boldsymbol{w}} = \mathbf{c}$$

Finally $$\frac{\partial J_t}{\partial \boldsymbol{w}} = -(1-\mathrm{p})\mathbf{c} + \sum_{k=1}^{K}(1-q_k)\,\mathbf{c}_k$$

Computing the derivative reveals that the updates to the parameters of the word embedding model are very simple computations. They involve the computation of the probability values for the words and context words and multiplying them with the existing embedding vectors. This (incomplete) computation shows how the weights of the embedding vector w of the word are updated.

Exercise: complete the missing parts of the computation.

Result

Matrices $W^{(w)}$ and $W^{(c)}$ that capture information on word similarity

- Words appearing in similar contexts generate similar contexts and vice versa
- Hence, mapped to similar representations in lower dimensional space
- Use $W = W^{(w)} + W^{(c)}$ as the low-dimensional representation

As a result we obtain two models mapping words into a low dimensional space for different reasons. In practice, the final model is constructed as the sum of the two matrices. Though, there exists no theoretical insight, why exactly this models work well, practice shows that they are organizing words in interesting ways, capturing many semantic and syntactic relationships.

A column of matrix $W^{(c)}$ represents

1. How relevant word c is for each concept
2. How often a context word c co-occurs with all words
3. A representation of word c in concept space Correct

A word embedding for given corpus …

1. depends only on the dimension d
2. depends on the dimension d and number of iterations in gradient descent
3. depends on the dimension d, number of iterations and chosen negative samples
4. there are further factors on which it depends Correct

Alternative Approaches

CBOW (Continuous Bag of Words Model)

- Predict word from context

GLOVE: exploits the following observation

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Technically similar

The model we have discussed is one variant of a number of similar models that have been proposed recently.

One interesting model is GLOVE, which is based on the observation that ratios of probabilities can be used to capture semantic relationships among terms. For example, the term solid is much more likely to co-occur with ice, than with water, and similarly steam co-occurs much more likely with gas than with ice. On the other hand, water co-occurs frequently with both. In the Glove paper it is stated: “Compared to the raw probabilities, the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion) and it is also better able to discriminate between the two relevant words” This may be interpreted as follows: the ratio of probabilities captures the semantics of the meaning of solid and gas, without referring to specific case in which this meaning occurs, namely the context of water.

Technically most of these models follow similar approaches, first modeling some observation on how co-occurrence can capture semantic relationship, and then using gradient descent methods to learn the parameters.

Properties of Word Embeddings

1. Similar terms are clustered
2. Syntactic and semantic relationships encoded as linear mappings
3. Dimensions can capture meaning

Word Embeddings have received recently a lot of attention as they exhibit fascinating capacity to capture different types of relationships among words. We will illustrate some of those in the following.

Glove: Nearest words to Frog

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

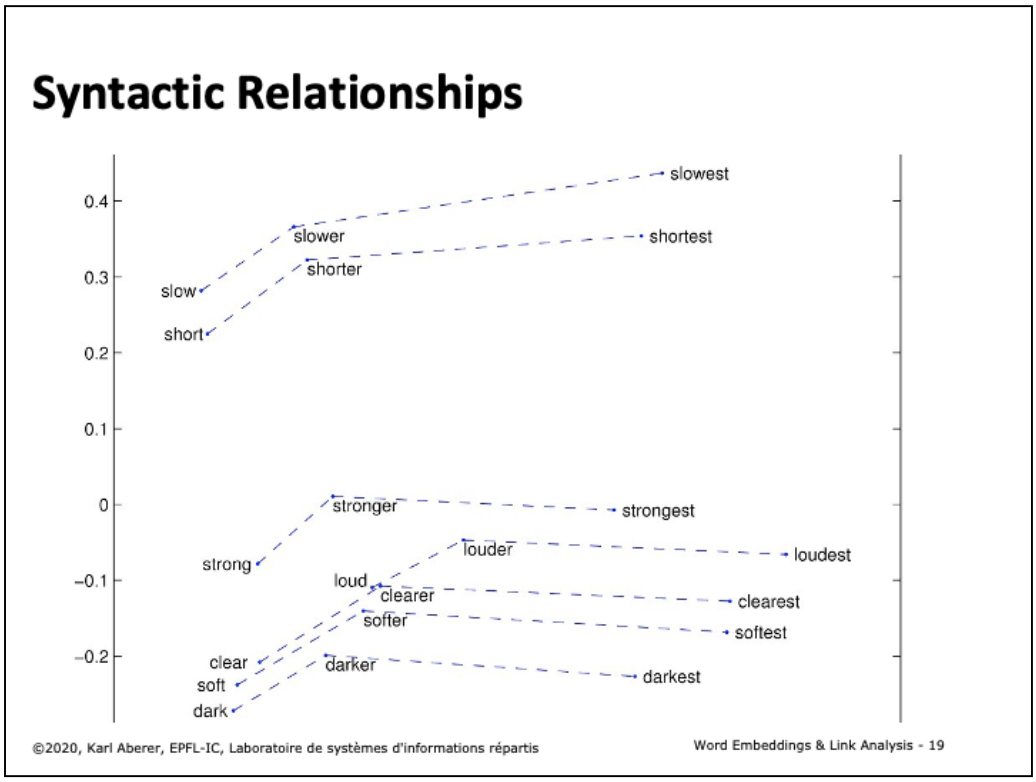
litoria

leptodactylidae

rana

eleutherodactylus

A first property is that similar words are grouped together. Here is a nice example, produced with Glove, that illustrates the point. (The underlying data is from Wikipedia).

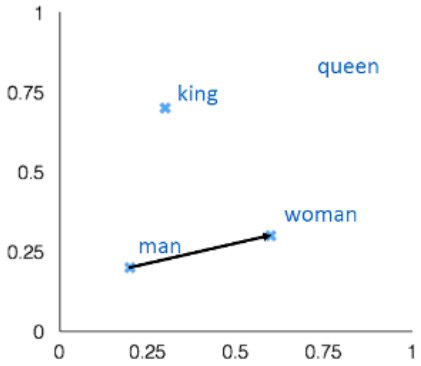


Word embeddings also capture syntactic relationships, like singular-plural or comparative and superlative, as shown here. This type of visualizations is obtained by projecting from the d-dimensional word embedding space (appropriately) into a 2-dimensional space.

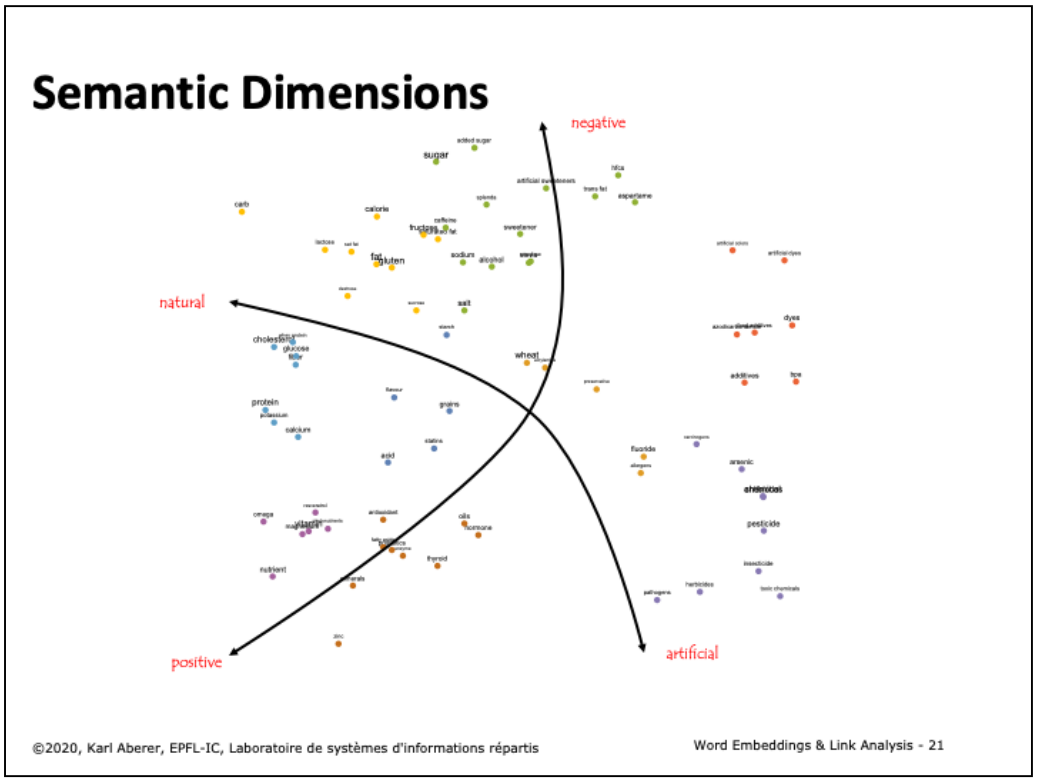
Word Analogies: semantic relationships

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Even more interestingly, word embeddings enable “computing” with relationships (this is called the word analogy task). Analogies translate into linear mappings!



Furthermore, word embeddings can capture semantic meaning in dimensions. In this example, data on nutrition has been analyzed and terms indicating qualities of foods are extracted. The word embedding clearly organizes it according to properties that are human understandable, e.g. natural vs. artificial ingredients.

Use of Word Embedding Models

Document Search

- Use word embedding vectors as document representation

Thesaurus construction and taxonomy induction

- Search engine for semantically analogous / related terms

Document classification

- Use of word embedding vectors of document terms as features

Impact

- Latent semantic indexing

Indexing by latent semantic analysis
S Deerwester, ST Dumais, GW Furnas... - Journal of the ..., 1990 - search.proquest.com
Cited by 12027 Related articles All 87 versions Web of Science: 3525 Cite Save

+2000

- Latent Dirichlet allocation

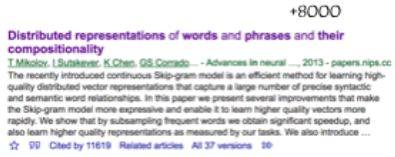
Latent dirichlet allocation
DM Blei, AY Ng, MI Jordan - Journal of machine Learning research, 2003 - jmlr.org
Cited by 17791 Related articles All 123 versions Web of Science: 5278 Cite Save

+8000

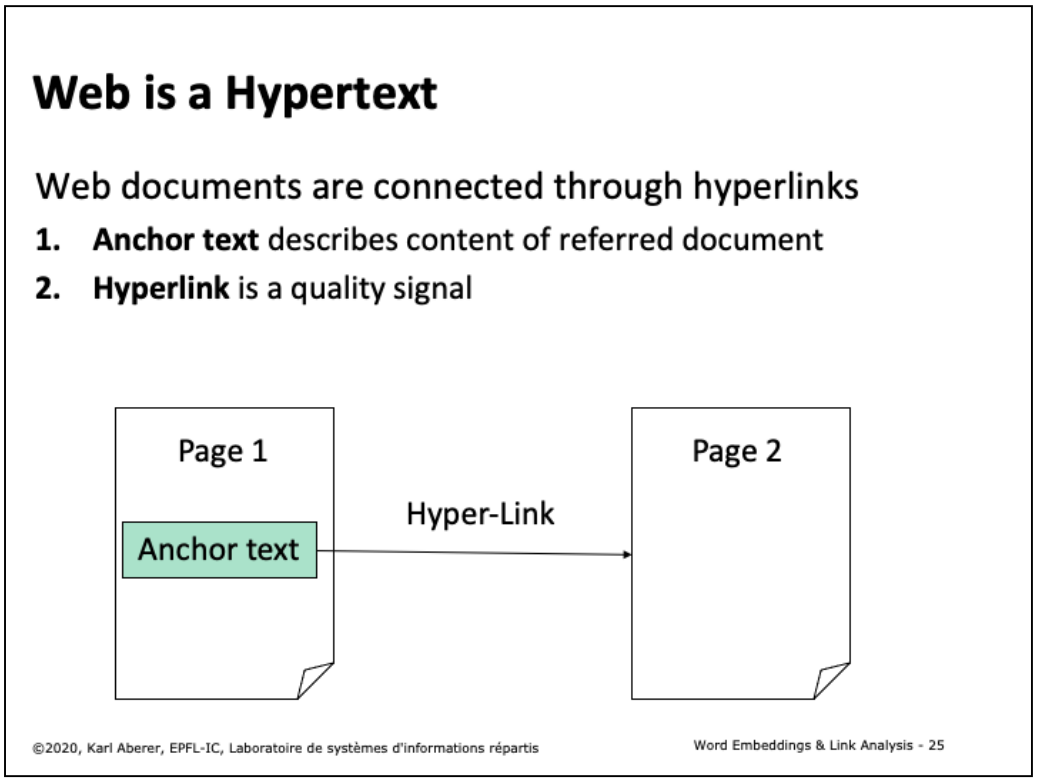
- Word embeddings

Distributed representations of words and phrases and their compositionality
T Mikolov, I Sutskever, K Chen, GS Corrado... - Advances in neural ..., 2013 - papers.nips.cc
Cited by 3370 Related articles All 23 versions Cite Save

+8000



11. LINK-BASED RANKING



Beyond the textual contents, Web documents consist also of hyperlinks. A hyperlink can be exploited for information retrieval in two ways:

1. The link is surrounded by some textual information that presumably refers to the content of the document the link is pointing to. Thus this text can complement the content of the referred document.
2. The link can also be considered as an endorsement of the referenced document by the author of the referring document. Thus the link can be used as a signal for quality and importance of the referred document.

Indexing Anchor Text

Anchor text is loosely defined as the text surrounding a hyperlink

Example: "You can find cheap cars here."
Anchor text: "You can find cheap cars here"

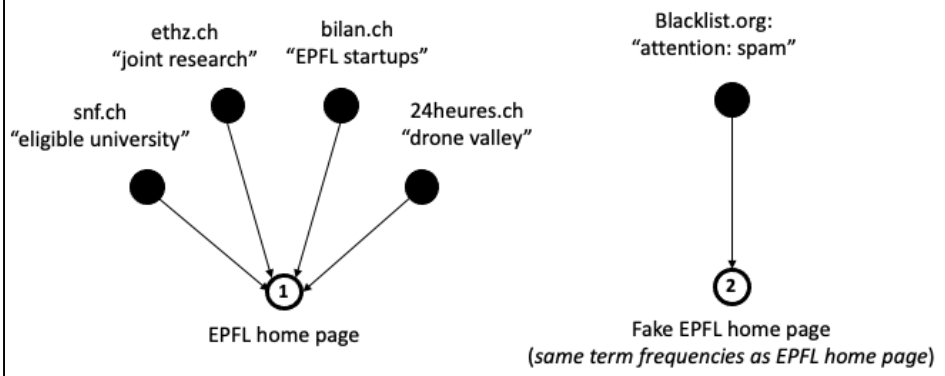
Anchor text may contain a lot of additional content on the referred page

- It might be a better description of the page than the page content itself

Anchor text can be defined in different ways. In general, it is considered as the text that is surrounding the link, and not only the text contained as part of the link tag (in the example, this text would simply be "here".) This text can contain valuable information on the referred page and thus be very helpful in retrieval.

Example

When indexing a document *D*, include (with some weight) anchor text from links pointing to *D*



This example illustrates the use of anchor text in retrieval. Typically a home page is very graphical and contains often very little relevant content. If we consider a home page, such as the EPFL home page, we would probably found many pages that very well characterize EPFL, such as pages mentioning topics related to research and technology transfer. Typically this links would point to the EPFL home page. Thus it would also let the home page stand out from other EPFL pages (such as pages on the laws and ordonnances of EPFL).

Assume that a malicious Internet user would create a fake EPFL home page. Then chances that such a page is referred by reputed organizations, such as SNF, is very low. On the other hand pages listing spam pages might point to such a page and indicate its true character.

Therefore it makes a lot of sense to include such anchor text in the representation of the document. This is usually done by adding it with a given weight.

Scoring of Anchor Text

Score anchor text with a weight depending on the authority of the anchor page's website

- E.g., if we were to assume that content from cnn.com or yahoo.com is authoritative, then trust (more) the anchor text from them

Score anchor text from other sites (domains) higher than text from the same site

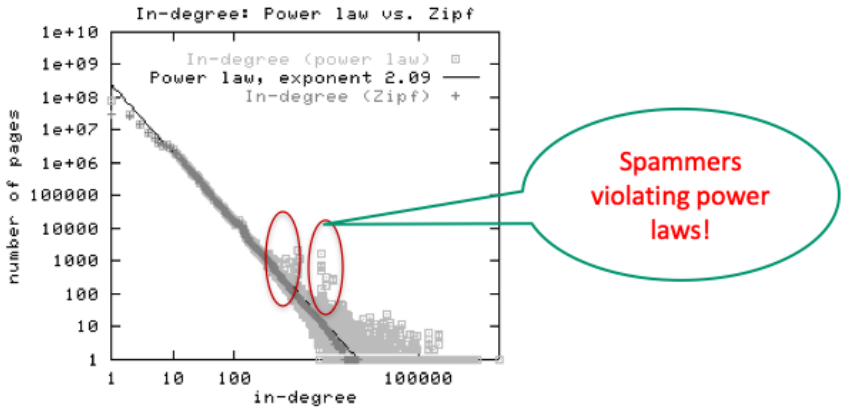
- non-nepotistic scoring

To fight spam, on can adapt the weighting function to the reputation of the page that is referring. This could be done by having a directory of highly reputed sites and to give high weights to its members. A more fundamental of producing such reputation scores we will see in the following part on link-based ranking.

In order to avoid self-promotion, another method to fight link spamming is to give lower weights to links within the same site (nepotistic scoring = promoting your own family members).

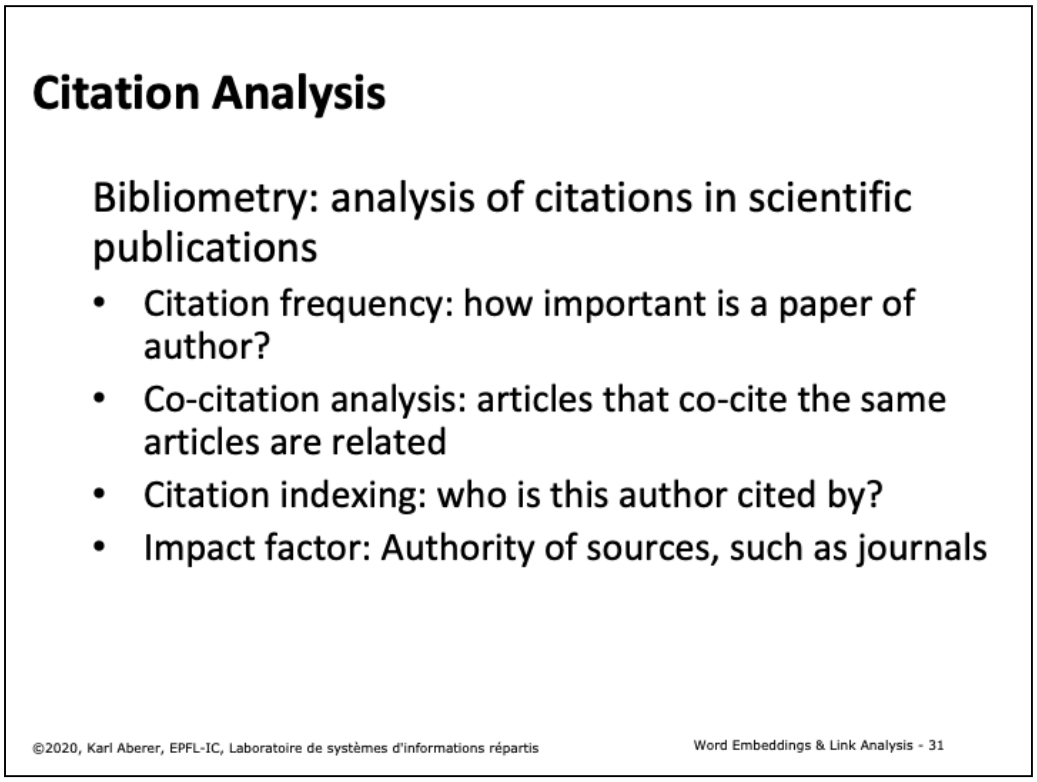
Indexing Anchor Text

Can sometimes lead to unexpected effects, e.g., easily **spammable**



One of the risks of including anchor text is that it makes pages spammable. Malicious users could create spam pages that point to web pages and try to relate it to contents that serve their interests (e.g., higher the quality of preferred pages by adding links, lower the quality of the undesired page by attaching negative anchor text). That this is happening can be seen from analyzing the in-degree distribution of Web pages. The figure shows a standard log-log representation of the in-degree vs. the frequency of pages. Normally this relationship should follow a power-law, which shows in a log-log representation as a linear dependency. In real Web data, we see that this power law is violated, and that certain levels of in-degrees are over-represented. This can be attributed to link spamming, which does create moderate numbers of additional links on Web pages.

LINK-BASED RANKING: PAGERANK

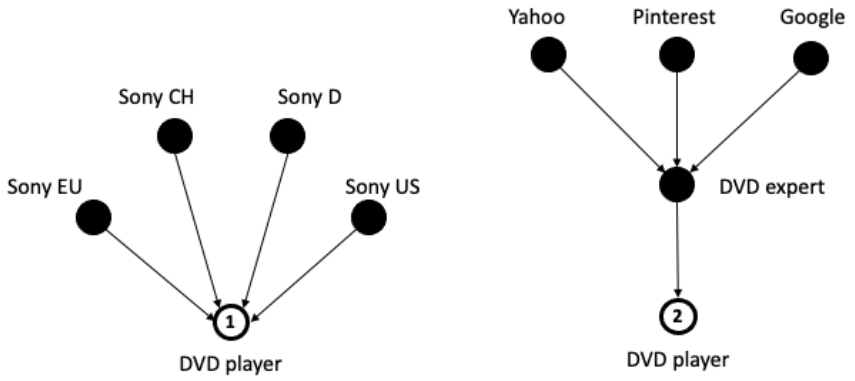


The use of links in order to evaluate the quality of information sources has a long tradition, in particular in science. The scientific discipline of bibliometry is fully devoted to the problem of evaluating the quality of research through citation analysis. Different ideas can be exploited to that end:

- The frequency of citations to a paper, indicating how popular / visible it is
- Co-citation analysis in order to identify research working in related disciplines
- Citation indexing in order to explore the profile of researchers referring to an author (as indicator of both the discipline and the quality)
- Analysis of the authority of sources of scientific publications, e.g. journals, publishers, conferences. This measure can then in turn be used to weight the relevance of publications.

All these ideas could also be exploited for any other document collections that have references, in particular for Web document collections with hyperlinks.

Citations on the Web



Full text retrieval result with equal ranking; which page is more relevant ?

- relevance related to number of referrals (incoming links)
- relevance related to number of referrals with high relevance

When retrieving documents from the Web, the link structure bears important information on the relevance of documents. Generally speaking, a document that is referred more often by other documents through Web links, is likely to be of higher interest and therefore relevance. So a possibility to rank documents is considering the number of incoming links. Doing this allows to distinguish documents that otherwise would be ranked equally or similarly when relying on text retrieval alone.

However, when doing this, also the importance of documents having a link to the document to be ranked may be different. Therefore not only counting then number of incoming links, but also weighing the links by the relevance of documents that contain these links appears to be appropriate. The same reasoning of course again applies then for evaluating the relevance of documents pointing to the document and so forth.

The Web isn't Scholarly Citation

Millions of participants, each with self interests

- Spamming is widespread

Once search engines began to use links for ranking (roughly 1998), link spam grew

- You can join a link farm – a group of websites that heavily link to one another

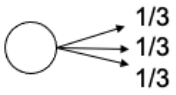
Simple link counting might not be appropriate!

In the web, analysis of links servers not only (or necessarily) the purpose of finding authorative sources, such as in science. It also and in particular can help to fight the aforementioned problem of link spamming. When using such methods, such as in any arms race, of course counter-measures have been devised by the adversaries. These resulted in the creation of link farms, that try to boost the relevance of Web pages by creating many (artificial) links to them. So just counting incoming links is probably not such a good idea in order to evaluate the quality of a Web page.

Link-based Ranking: Idea

Imagine a user doing a **random walk** on Web pages:

- Start at a random page
- At each step, leave the current page along one of the links on that page, with same probability

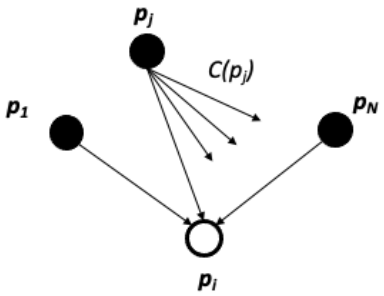


"In the long run" each page has a long-term visit rate - use this as the page's score

The fundamental idea of link-based scoring while attempting to fight spam is based on the concept of a random walker on the Web. Assume a random walker would visit a Web page. Then it would follow each f the outgoing links with equal probability. If walking for a very long time in the Web, it would have a certain fraction of visits it passes by every page.

One of the consequences of this model would be that pages that have few in-links, would be relatively infrequently visited. Since link farms usually have not many links pointing to them, in this way their influence in terms of link spamming would be moderated.

Random Walker Model



$$P(p_i) = \sum_{p_j | p_j \to p_i} \frac{P(p_j)}{C(p_j)}$$

N is the number of Web pages

$C(p)$ is the number of outgoing links of page p

$P(p_i)$ probability to visit page p_i, where page p_i is pointed to by pages p_1 to p_N = **relevance**

Result

- If a random walker visits a page more often it is more relevant
- takes into account the number of referrals AND the relevance of referrals

Here we describe the random walker model more formally. The long term probability of visiting a page, depends on the long-term probability of having visited a page that is referring to the page. Thus the formulation of the random walker model becomes recursive.

Transition Matrix for Random Walker

The definition of $P(p_i)$ can be reformulated as matrix equation

$$R_{ij} = \begin{cases} \frac{1}{C(p_j)}, if\ p_j \rightarrow p_i \\ 0, otherwise \end{cases}$$

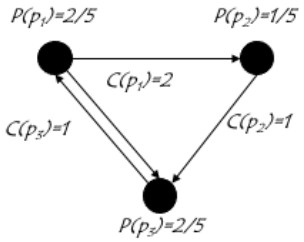
$$\vec{p} = (P(p_1), ..., P(p_n))$$

$$\vec{p} = R.\vec{p}, \quad \|\vec{p}\|_1 = \sum_{i=1}^{n} p_i = 1$$

The vector of page relevance values is the Eigenvector of the matrix R for the largest Eigenvalue

In order to determine the solution to the recursive formulation of the probabilities of a random walker to visit a page, we introduce a transition probability matrix R, which captures the probability of transitioning from one page to another. We also require that the long-term probabilities of visiting a page add up to 1. With this representation the long-term visiting probabilities become the Eigenvector of matrix R. More precisely, they are the Eigenvector with the largest Eigenvalue.

Example



Links from p_1

$$L = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Links to p_1

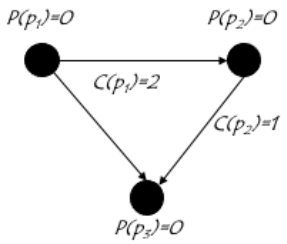
Link Matrix

$$R = \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} \frac{2}{5} \\ \frac{1}{5} \\ \frac{2}{5} \end{pmatrix}$$

Ranking p_1, $p_3 > p_2$

This example illustrates the computation of the probabilities for visiting a specific Web page. The values $C(p_i)$ correspond to the transition probabilities. They can be derived from the link matrix. The link matrix is defined as $L_{ij}=1$ if there is a link from p_j to p_i. The link matrix is normalized by the outdegree, by dividing the values in the columns by the sum of the values found in the column, resulting in Matrix R. The probability of a random walker being in a node is then obtained from the Eigenvector of this matrix.

Modified Example



Links from p_1

$$L = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Links to p_1

Link Matrix

$$R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

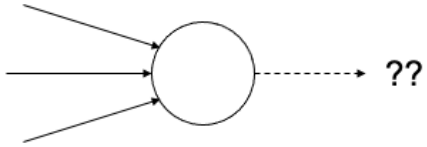
No Ranking

This example illustrates the problem of dead ends. We see that there exists a node p_3 that is a "sink of rank". Any random walk ends up in this sink, and therefore the other nodes do not receive any ranking weight. Consequently also the rank of sink does not. Therefore the only solution to the equation p=Rp is the zero vector.

Pure Random Walker Does Not Work

The web is full of dead-ends

- Random walk can get stuck in dead-ends
- Makes no sense to talk about long-term visit rates



Teleporting

- At a dead end, jump to a random web page
- At any non-dead end, jump to a random web page with some probability (e.g. 15%)
- Result: Now cannot get stuck locally, there is a long-term rate at which any page is visited

A practical problem with the random walker is the fact that there exist Web pages that have no outgoing links. Thus the random walker would get stuck. To fix this, the concept of teleporting is introduced, where the random walker jumps to any randomly selected Web page. In case of arriving at a dead end, it jumps in any case, otherwise with a given probability instead of following an outgoing link.

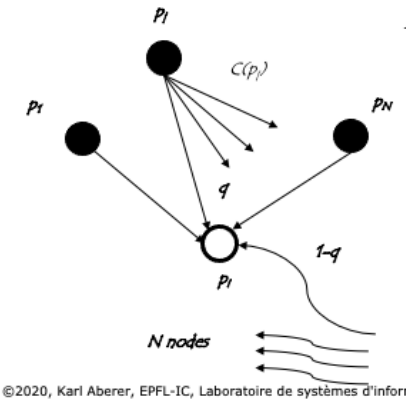
Another problem are pages that have no incoming links: they would never be reached by the random walker, and the weight that they could provide to other pages would not be considered. Also this problem is solved by teleporting.

Source of Rank: Teleporting

Assumption

- random walker jumps with probability 1-q to an arbitrary node
- thus it can leave dead ends and nodes without incoming links are reached

PageRank method



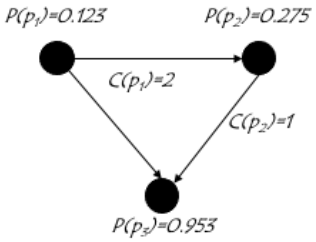
$$P(p_i) = c(\frac{(1-q)}{N} + q \sum_{p_j | p_j \to p_i} \frac{P(p_j)}{C(p_j)}), c \leq 1$$

$$\vec{p} = c((qR + (1-q)E)).\vec{p}, \quad E = \left[\frac{1}{N}\right]_{NxN}$$

$$\vec{p} = c(qR.\vec{p} + \frac{(1-q)}{N}\vec{e}), \quad \vec{e} = (1,...,1)$$

To avoid the previously described problem, we add a "source of rank". The idea is that a random walker in each step can, rather than following a link, jump to any page with probability 1-q. Therefore the random walker can leave pages that have no outgoing links and also pages without incoming links can be reached by the random walk and give weight to other pages. In the mathematical formulation of the random walk this is resulting in an additional a term for the source of rank. Since at each step the random walker makes a jump with probability 1-q and any of the N pages is reached with the same probability the additional term is (1-q)/N. Reformulating this again as matrix equation requires adding a NxN Matrix E with all entries being 1/N. This is equivalent to saying that with probability 1/N transitions among any pairs of nodes (including transition from a node to itself) are performed. Since the vector p has norm 1, i.e., the sum of the components is exactly 1, E.p=e, where e is the unit vector, the matrix equation can be reformulated in the second form shown below. The method described is called PageRank and is used by Google. By modifying the values of the matrix E also a priori knowledge about the relative importance of pages can be added to the algorithm.

Modified Example



$$R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, E = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}, q = 0.9$$

$$qR + (1-q)E = \begin{pmatrix} \frac{1}{30} & \frac{1}{30} & \frac{1}{30} \\ \frac{29}{60} & \frac{1}{30} & \frac{1}{30} \\ \frac{29}{60} & \frac{14}{15} & \frac{1}{30} \end{pmatrix} \quad \vec{p} = \begin{pmatrix} 0.091 \\ 0.203 \\ 0.705 \end{pmatrix}$$

Ranking $p_3 > p_2 > p_1$

With the modification of rank computation using a source of rank, we obtain for our example again a non-trivial ranking which appears to be appropriate.

The 1-norm of p is {0.0912349, 0.203606, 0.705159}.

Practical Computation of PageRank

Iterative computation

$$\vec{p}_0 \leftarrow \vec{s}$$
$$while\ \delta > \varepsilon$$
$$\vec{p}_{i+1} \leftarrow qR \bullet \vec{p}_i$$
$$\vec{p}_{i+1} \leftarrow \vec{p}_{i+1} + \frac{(1-q)}{N}\vec{e}$$
$$\delta \leftarrow \left\| \vec{p}_{i+1} - \vec{p}_i \right\|_1$$

ε termination criterion
s arbitrary start vector, e.g. s = e

For the practical computation of the PageRank ranking an iterative approach can be used. It is derived from the second form of the formulation of the visiting probabilities of the random walker that we have given. The vector e used to add a source of rank has not necessarily to assign uniform weights to all pages, but might reflect itself a ranking of Web pages.

Example: ETHZ Page Rank

Doc_ID	Rank_Value	URL
1	0.002536	http://www.ethz.ch/
146	0.002292	http://www.ethz.ch/r_amb/
10	0.000654	http://www.ethz.ch/default_de.asp
35	0.000511	http://www.rereth.ethz.ch/
376124	0.000503	http://computing.ee.ethz.ch/sepp/matlab-5.2-to/helpdesk.html
67378	0.000497	http://computing.ee.ethz.ch/sepp/
59887	0.000485	http://www.computing.ee.ethz.ch/sepp/
89307	0.000485	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2.2ref/docs/api/overview-summary.html
216716	0.000485	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2/api/overview-summary.html
147932	0.000484	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2ref/docs/api/overview-summary.html
175544	0.000484	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2ref/docs/api/overview-summary.html
186766	0.000478	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2/api/overview-summary.html
228634	0.000477	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2.1ref/docs/api/overview-summary.html
228421	0.000464	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2.2ref/docs/api/overview-summary.html
3161	0.00045	http://www.ethz.ch/r_amb/reto_ambuehler.html
215673	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/files.html
259672	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/globals.html
259671	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/functions.html
259670	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/annotated.html
259669	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/classes.html

Figure 1: Top 20 of ETH Zurich Web Documents

These are the top documents from the PageRank ranking of all Web pages at ETHZ (Data from 2001). It is interesting to see that documents related to Java documentation receive high ranking values. This is related to the fact that these documents have many internal cross-references.

Web Search

PageRank is part of the ranking method used by Google

- Compute the global PageRank for all Web pages
- Given a keyword-based query retrieve a ranked set of documents using standard text retrieval methods
- Merge the ranking with the result of PageRank to both achieve high precision (text retrieval) and high quality (PageRank)
- Google uses also many other methods to improve ranking

Technical challenges

- Crawling the Web
- Efficient computation of Page Rank for large link databases
- Combination with other ranking methods (text)

PageRank is used as one metrics to rank result documents in Google. Essentially Google uses text retrieval methods to retrieve relevant documents and then applies PageRank to create a more appropriate ranking. Google uses also many other methods to improve ranking, e.g., by giving different weights to different parts of Web documents and user relevance feedbacks. For example, title elements are given higher weight. The details of the ranking methods are trade secrets of the Web search engine providers.

Other issues that Web search engines have to deal with, are crawling the Web, which requires techniques that can explore the Web without revisiting pages too frequently. Also the enormous size of the document and link database poses implementation challenges in order to keep the ranking computations scalable. One of the outcomes of solving these challenges are the recent “cloud computing” infrastructures, which are large-scale computing clusters constructed from commodity hardware.

The relevance determined using the random walker model corresponds to

1. The number of steps a random walker needs to reach a page
2. The probability that the random walker visits the page in the long term Correct
3. The number of incoming links a random walker can use to visit the page
4. The probability that the random walker will visit once the page

Consider a random jump matrix with entries 1/3 in the first column and 0 otherwise. It means

1. A random walker can always leave node 1 even without outgoing edges Correct
2. A random walker can always reach node 1, even without incoming edges
3. A random walker can always leave node 2, even without outgoing edges
4. none of the above

LINK-BASED RANKING: HITS

Hyperlink-Induced Topic Search (HITS)

Key Idea: in response to a query, instead of an ordered list of pages, find two sets of inter-related pages:

- **Hub pages** are good lists of links on a subject
  - e.g., "World top universities"
- **Authorative** *pages* are referred recurrently on good hubs on the subject
  - e.g., "EPFL"

Best suited for "broad topic" queries rather than for page-finding queries

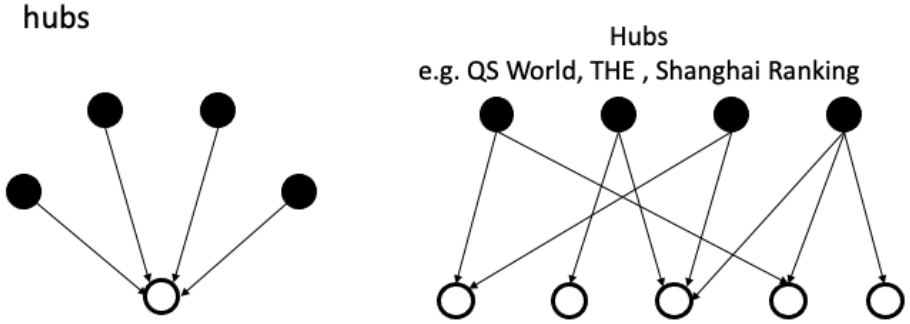
- Understand common perception of quality

Hub-Authority ranking identifies not only pages that have a high authority, as measured by the number of incoming links, but also pages that have a substantial "referential" value, having many outgoing links (to pages of high importance). Different to the PageRank algorithm this technique has been originally proposed to post-process query results (rather than to rank pages from the complete Web graph). It can be used as an add-on to existing search engines, but as well as an alternative to the PageRank method.

Hub-Authority Ranking

Approach

- **Hubs** are pages that point to many/relevant authorities
- **Authorities** are pages that are pointed to by many/relevant hubs



page with large in-degree
e.g. EPFL

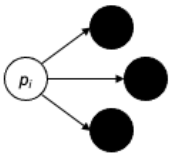
Authorities
e.g. EPFL, MIT, Stanford, ETHZ

In order to realize the idea of distinguish authorities from hubs a simple approach is taken. Hub pages are consider as those that are referred a lot by authority pages and vice versa. The example illustrates of how this would ideally separate authority pages, in the case of universities well known universities, from hub pages, such as university ranking and portal sites.

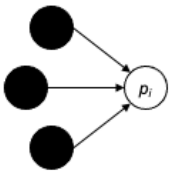
Computing Hubs and Authorities

Repeat the following updates, for all *p*

$$H(p_i) = \sum_{p_j \in N | p_i \to p_j} A(p_j)$$



$$A(p_i) = \sum_{p_j \in N | p_j \to p_i} H(p_j)$$



Normalize values (scaling)

$$\sum_{p_j \in N} A(p_j)^2 = 1 \qquad \sum_{p_j \in N} H(p_j)^2 = 1$$

More precisely, the scores are recomputed by simply adding the scores of all incoming edges. For computing authority scores this is the hub scores and for hub scores the authority scores. In order to avoid that the scores grow continuously, they are rescaled to in each step, by normalizing the score vectors to one.

HITS algorithm

$$n := |N|;\ (a_0, h_0) := \frac{1}{n^2}\big((1, \dots, 1), (1, \dots, 1)\big)$$

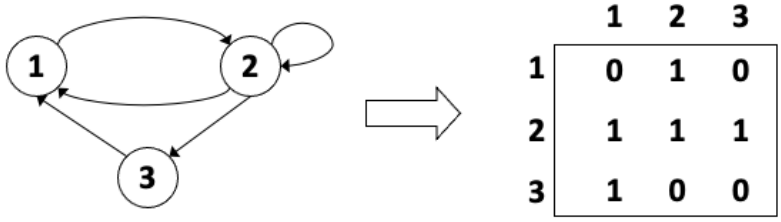
$$\begin{aligned} &\textit{while } l < k \\ &\quad l := l + 1 \\ &\quad a_l := (\textstyle\sum_{p_i \to p_1} h_{l-1,i}, \dots, \sum_{p_i \to p_n} h_{l-1,i}) \\ &\quad h_l := (\textstyle\sum_{p_1 \to p_i} a_{l,i}, \dots, \sum_{p_n \to p_i} a_{l,i}) \\ &\quad (a_l, h_l) := \left(\frac{a_l}{|a_l|}, \frac{h_l}{|h_l|}\right) \end{aligned}$$

In practice, 5 iterations sufficient to converge!

As for PageRank the Hub/Authority values can be iteratively computed. x corresponds to the authority weight and y to the hub weight. At each iteration the values are renormalized to length 1.

Convergence of HITS

n x n link matrix L^t



Up to normalization

$h = La$, $a = L^t h$, thus

a^* is an Eigenvector of LL^t

h^* is an Eigenvector of L^tL

We can interpret the iterative algorithm for computing HITS in terms of computing Eigenvectors for the link matrix. If L is the link matrix then the computation of a from h is defined as $a=Lh$ and the computation of h from a is defined as $h=L^ta$. Therefore a^*, the authority vector obtained after convergence, is the principal Eigenvector of Matrix LL^t and h^*, the hub vector obtained after convergence, is the principal Eigenvector of matrix L^tL.

Remains the question whether this algorithm always converges. The algorithm is a particular algorithm for computing eigenvectors: the *power iteration* method. It is proven to converge against the principal Eigenvalue. Important is the normalization of the vectors obtained in each step.

When computing HITS, the initial values

1. Are set all to 1
2. Are set all to 1/n
3. Are set all to 1/n^2 Correct
4. Are chosen randomly

If the first column of matrix L is (0,1,1,1) and all other entries are 0 then the authority values

1. $(0,1,1,1)$
2. $\left(0, 1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3}\right)$ Correct
3. $\left(1, 1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3}\right)$
4. $(1,0,0,0)$

Obtaining the Base Set

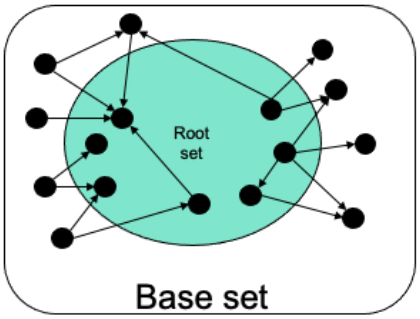
Given a query (e.g. “EPFL”), obtain all pages mentioning the query

- call this the **root set** of pages.

Add page that either

- points to a page in the root set, or
- is pointed to by a page in the root set.

Use this set as **base set**



Sample Results Obtained

(java) Authorities	
.328 http://www.gamelan.com/	*Gamelan*
.251 http://java.sun.com/	*JavaSoft Home Page*
.190 http://www.digitalfocus.com/digitalfocus/faq/howdoi.html	*The Java Developer: How Do I...*
.190 http://lightyear.ncsa.uiuc.edu/~srp/java/javabooks.html	*The Java Book Pages*
.183 http://sunsite.unc.edu/javafaq/javafaq.html	*comp.lang.java FAQ*
(censorship) Authorities	
.378 http://www.eff.org/	*EFFweb - The Electronic Frontier Foundation*
.344 http://www.eff.org/blueribbon.html	*The Blue Ribbon Campaign for Online Free Speech*
.238 http://www.cdt.org/	*The Center for Democracy and Technology*
.235 http://www.vtw.org/	*Voters Telecommunications Watch*
.218 http://www.aclu.org/	*ACLU: American Civil Liberties Union*

These are two example results that have been obtained by applying this method. (It is interesting to compare those to the ones you would obtain by using a search engine alone). In particular Gamelan will not show up in the result, whereas the Java Sun page is usually among the top ranked.

HITS Conclusions

Potential issues

- Topic Drift: off-topic pages can cause off-topic "authorities" to be returned
- Mutually Reinforcing Affiliates: clusters of affiliated pages/sites can boost each others' scores

Social Network Analysis

- PageRank and HITs are examples of Social Network (SN) Analysis algorithms
- SNs contain a lot of other interesting structure (see later)

Implementation

- How to efficiently obtain the base set?

Similarly as for PageRank, the HITS algorithm is also vulnerable structural anomalies of the link structure, be it caused by conscious manipulation or by chance. Topic drift would imply that the set of base pages refers to a topic that is different from the original intended one, expressed by a search query. For example, if we would search for top European universities, it could well happen that the topic would drift to top US universities, when ranking hubs come into play that strongly refer to those. We could have also clusters of related pages, that in the iterative computation mutually boost each others scores, and thus give too high weight to pages that would not merit it. This problem is similar to the one of link spamming with link farms.

In terms of implementation, link-based ranking algorithms require the capability to efficiently retrieve the link graph for the Web. This problem is addressed by so-called connectivity servers that we will introduce next.

11. LINK INDEXING

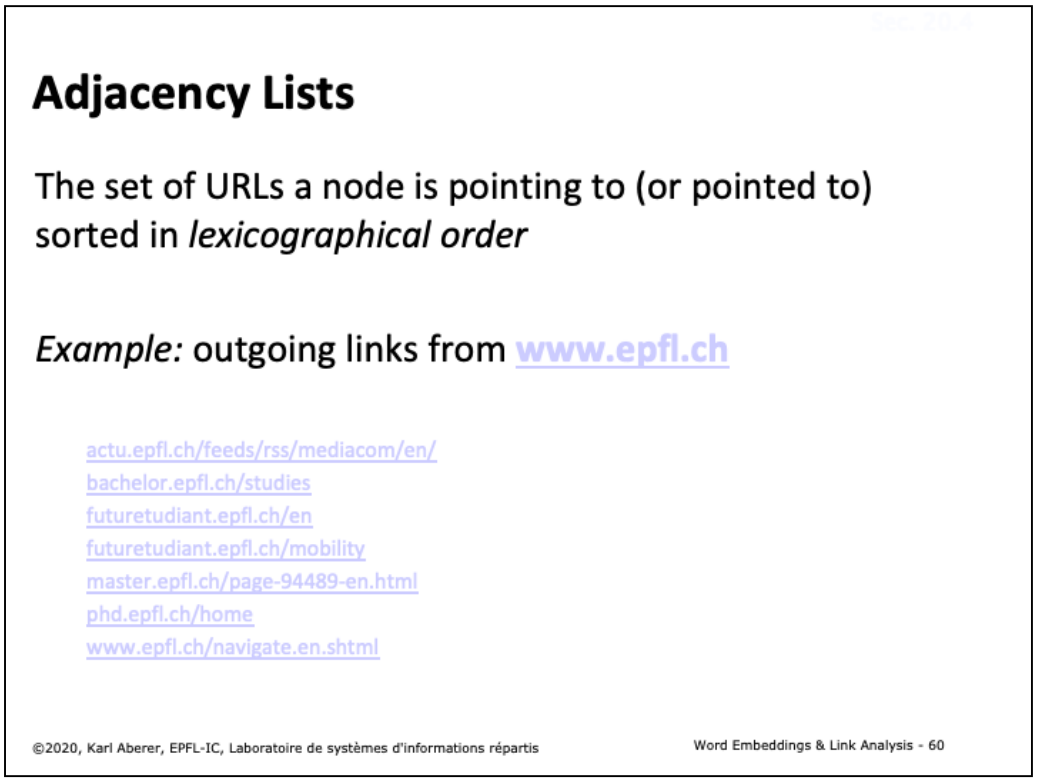
Connectivity Server

Support for fast queries on the web graph
- Which URLs point to a given URL?
- Which URLs does a given URL point to?

Stores mappings in memory from
- URL to outlinks, URL to inlinks

Applications
- Link analysis (PageRank, HITS)
- Web crawl control
- Web graph analysis
  - Connectivity, crawl optimization

In order to efficiently analyze the Web graph the Web links need to stored in a database, respectively index. A connectivity server is such an index. In essence, it stores for each URL all the URLs that the Web page at the URL is pointing to, and the URLs of other Web pages that point to this URL. Apart from link-based ranking algorithms as described before the applications are manifold.



A connectivity server has to store all outgoing (and incoming) links to a web page. For example, the home page of EPFL contains a large set of outgoing links, some of which are shown here. As a first step, the lists of links are sorted in lexicographical order. As a result we obtain the adjacency list, which is similar to the posting list of a document.

Representation of Adjacency Lists

Assume each URL represented by an integer

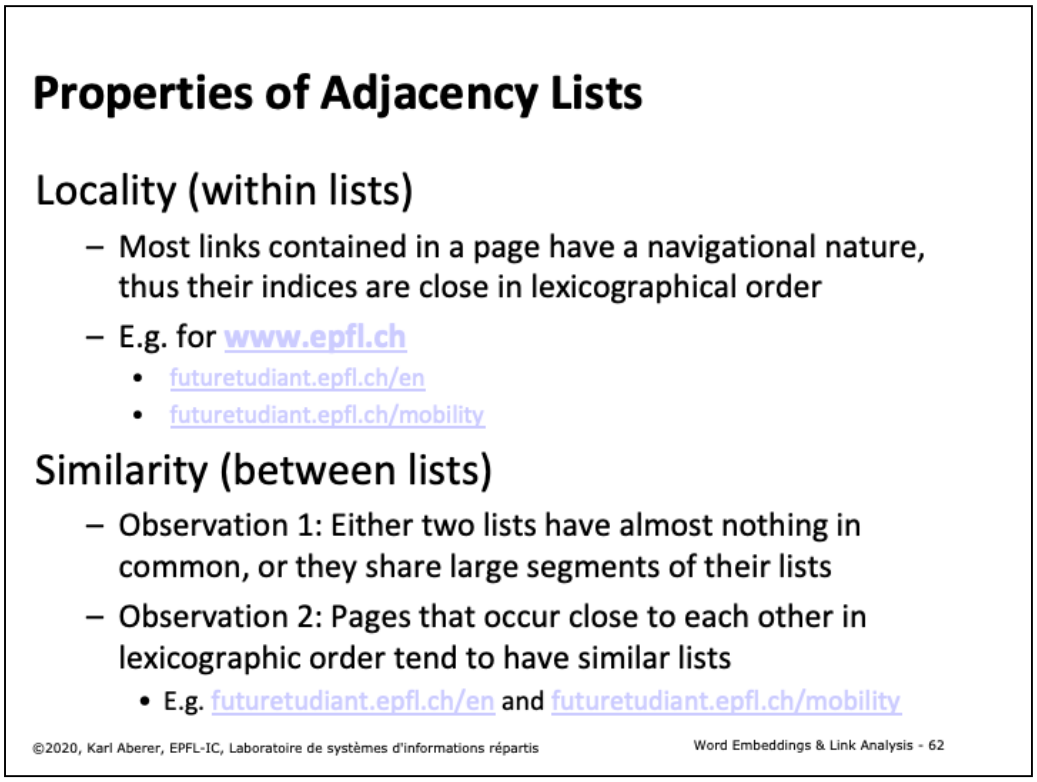
- For a 4 billion page web, we need 32 bits per node
- Naively, this demands 64 bits to represent each hyperlink (source and destination node); on average 10 links per page
- For the current Web: 320 GB
- Can we do better (for main memory storage)?

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

As a first means to optimize the representation of adjacency lists, we represent each URL by one integer, instead of storing its textual form. From this we can estimate that for the current Web, we need 320 GB to represent all links:

- The current (crawled) Web has about 4 billion pages (http://www.worldwidewebsize.com/)
- It is estimated that a page contains on average 10 links
- We need 32 bits for each URL, which demands 64 bits for the storage of a single link.

Even with large memories this still a significant index size. In the following, we will show how to get this down to approximately ~3 bits/link which makes the index much more manageable. This will be achieved by systematically compressing the adjacency lists.



For compressing adjacency lists we base ourselves on the following observations:

Locality. Most links contained in a page have a navigational nature: they lead the user to some other pages within the same host (as we can see clearly from the example of the EPFL home page). If we compare the source and target URLs of these links, we observe that they share often a long common prefix. Thus, if URLs are sorted lexicographically, the index of source and target are close to each other. Locality is a property of a list, thus addresses intra-list similarity.

Similarity. Pages that occur close to each other (in lexicographic order) tend to have many common successors. This is because many navigational links are the same within the same local cluster of pages, and even non-navigational links are often copied from one page to another within the same host. Similarity is a property concerning different lists, thus addresses inter-list similarity.

Exploiting Locality

Use Gap Encoding (as in inverted files)

- $S(x) = (s_1,\ldots,s_k)$ will be represented as $(s_1\text{-}x, s_2\text{-}s_1\text{-}1,\ldots,s_k\text{-}s_{k\text{-}1}\text{-}1)$
- Use of varying length encoding

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

Node	Outdegree	Successors
...	...	...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...	...	...

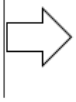
Locality can be exploited in a way analogous of how compression of posting lists for text indexing has been performed. Instead of storing the absolute values of the URL identifiers, differences are stored. In other words, we perform gap encoding. The resulting differences are then encoded using a varying length compression scheme, such as gamma coding.

Exploiting Similarity

Copy data from similar lists (exploit observation 1)

- Reference list: reference to another list
  - Searched in a neighboring window of nodes (exploit observation 2)
- Copy list: bitmap indicates nodes copied from reference list
- Extra nodes: additional nodes not in reference list

Node	Outdegree	Successors
...	...	...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...	...	...



Node	Outd.	Ref.	Copy list	Extra nodes
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...	...	...	...	...

Result: about 3 bytes / link (with some further compression)

For exploiting similarity we apply a similar idea as gap encoding, but now applied to the differences among different adjacency lists. If we consider a reference list (in the example the list of Node 15), subsequent adjacency lists can story a bitlist that indicates which nodes of the reference lists are retained in the subsequent adjacency list, and which are not. 0 indicates that the node in the reference lists is not retained, and 1 indicates that it is retained. Since the subsequent adjacency list can also contain other nodes, that are not stored in the reference list, those extra nodes are stored explicitly.

Candidates for potential reference lists are searched among neighboring lists using a window of predefined size. The choice of the window size is critical, as larger windows increase chances of finding good candidates, but also increase the cost of compression

Together with some further compression applied to the copy lists and the extra nodes, this index compression achieves about 3 Bytes/link cost in the representation of the Web graph.

When compressing the adjacency list of an URL, a reference list

1. Is chosen from neighboring URLs that can be reached in a small number of hops
2. May contain URLs not occurring in the current page Correct
3. Lists all URLs not contained in the current page
4. All of the above

Which is true?

1. Exploiting locality with gap encoding may increase the size of an adjacency list
2. Exploiting similarity with reference lists may increase the size of an adjacency list
3. Both of the above is true Correct
4. None of the above is true

References

Course material based on

- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008 (http://www-nlp.stanford.edu/IR-book/)
- Course Information Retrieval by TU Munich (http://www.cis.lmu.de/~hs/teach/14s/ir/)
- http://videolectures.net/deeplearning2015_manning_language_vectors/

Relevant articles

- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- Goldberg, Yoav, and Omer Levy. "word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method." *arXiv preprint arXiv:1402.3722* (2014).

References

Relevant articles

- Sergey Brin , Lawrence Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems, v.30 n.1-7, p.107-117, April 1, 1998.
- Jon M. Kleinberg: Authoritative Sources in a Hyperlinked Environment. JACM 46(5): 604-632 (1999)
- Boldi, Paolo, and Sebastiano Vigna. "The webgraph framework I: compression techniques." Proceedings of the 13th international conference on World Wide Web. ACM, 2004.