Web Information Extraction and Retrieval

Marko Bajec and Slavko Žitnik
Laboratory for Data Technologies, UL FRI

Module 1: Web crawling

Preferential crawlers

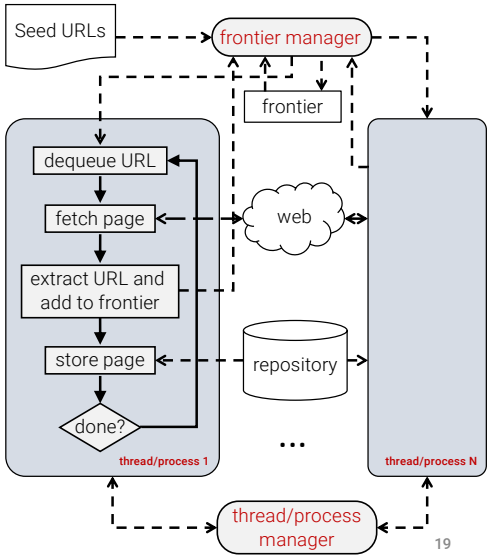
- Preferential crawling
- Important
  - Preference setting (web topology, page content)
  - Selection of seed pages
- Implementation (sorted dynamic list + hash table)
- Time complexity
- Best-first crawling

Fetching

- Implementation (web client, HTTP requests)
- Limiting response time and page size
- HTTP headers (response code, redirect, last modified...)
- Error-checking and exception handling
- Keeping statistics
- Program language support

Concurrency

- Problematical resources
  - network,
  - CPU,
  - disk
- Improvements of sequential crawler
  - multithreading
  - multiprocessing



Page repository

- File system
  - each page in a separate file
  - many pages in one file (plus hash table)
- Databases
  - embedded, e.g. Berkeley DB (key value store)

Fokusirani pajki gradijo indeks in se iz njega ucijo

Classifier based focused crawler

- Classification
  - Each crawled page is assigned relevance score: $R(p) = \sum_{c \in c^*} \Pr(c|p)$
  - where c^* denotes categories of interest
- Soft focused strategy
  - Links of page p are given preference $R(p)$ when written to the Frontier.
- Hard focused strategy
  - Find leaf category c_{leaf} that most likely include p
  - If ancestor of c_{leaf} focus category → add links(p) to Frontier (using $R(p)$).

Text classification with naïve Bayes

$$\Pr(c|d) = \frac{\Pr(d|c) * \Pr(c)}{\Pr(d)}$$

where $c \in C$ is a category from a set of categories C and d represents the crawled page. If d represented using multinominal document model then:

$\Pr(c|d) \approx \Pr(c) \prod_{i=1}^{|V|} \Pr(w_i|c)^{f_{w_i}}$, where V is the vocabulary of distinct words w_i in document d that we would like to classify and f_{w_i} is the frequency of words w_i in d. We are looking for $c_j \in C$ where

$$c_j = \underset{c_k \in C}{argmax} \Pr(c_k) \prod_{i=1}^{|V|} \Pr(w_i|\, c_k)^{f_{w_i}}$$

Context-based focused crawler*

- Procedure:
  - Use TFIDF to build vocabulary V of important words w for the topic of interest.
  - Build Naïve Bayes classifiers for each layer: $\Pr(l_x|p) = \Pr(w_1|lx) * \Pr(w_2|l_x) * \cdots * \Pr(w_n|l_x) * \Pr(lx)$, where $w_{1\ldots n} \in (p \cap V)$
  - If probability of layer with $\max \Pr(l|p) < threshold$ → classify page p to $others$ otherwise put to queue of the corresponding layer.
  - Link to be followed next is selected from a nonempty queue of the lowest layer number.

*Diligenti, M., F. Coetzee, S. Lawrence, C. Giles, and M. Gori. *Focused crawling using context graphs*. In Proceedings of **International Conference on Very Large Data Bases** (VLDB-2000), 2000.

Uporablja kontenstni graf, ki nam pove, kako pridemo na ciljno stran iz prejšnjih strani.
primer iscemo data retrieval : FRI -> Topics -> Data Retrieval

Tunneling…

- Limitations of Best-first approach
- Tunneling
  - Which links from irrelevant pages to follow?
  - How long to follow a tunnel?
  - How to estimate the potential of a certain tunnel?

Tunneling…

- Concepts*:
  - Nuggets and duds: relevant and irrelevant pages
  - Path: a sequence from one nugget to another one
  - Threshold: minimal similarity with topics to consider p as a nugget
  - Cutoff: max length of a path
  - Distance: length from the last nugget

$$distance = \begin{cases} 0, & if\ p\ nugget \\ 1 + distance(parent(p)), & otherwise \end{cases}$$

*Bergmark, D., Lagoze, C., & Sbityakov, A. (2002), *Focused Crawls, Tunneling, and Digital Libraries*, In proc. of the **European Conference on Digital Libraries** (ECDL) 2002, Rome, September 2002, 91-106.

Tunneling

- Experimental findings (on 500.000 documents)
  - Most common distances are 7 and 8
  - Better parents have better children, but the opposite is not true.
  - For high-scoring parents, highly scoring grandparents important.

Distance incorporating path history

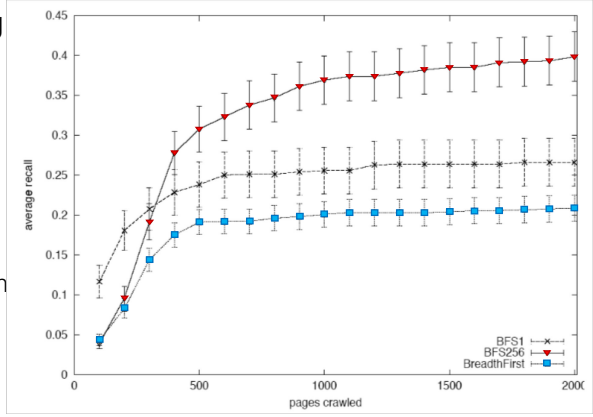
$$distance = \begin{cases} 0, & if\ p\ nugget \\ \min(1, (1-c)e^{\frac{2d_p}{C}}), & otherwise \end{cases}$$

where d_p: parent distance, c: current node correlation, C: cutoff

(Topical) tematski pajki delajo v zivo, nimajo predznanja, kalkulirajo relevantnost on the fly, lahko se uci iz besed in povezav (hyperlinks)-podobnost besed(word metrics), podobnost povezav(link metrics)

Best-N-first crawler

- Visiting of less promising pages to avoid "local optimums"
- Best-N-first crawler
  - follow best N leads
  - fetch all, merge-sort, repeat
  - Exploitation vs exploration



Pajki se med seboj razlikujejo, po tem, kako se racuna podobnost strani in kako utežujejo posamezne koscke spletne strani

Adaptive topical crawlers

- Static vs adaptive strategy (incremental learning)
- Example of adaptive strategy:

```
select features fe_i∈F
while not enough relevant pages crawled do
  crawl new page p
  calculate page priority P(p)=f(λ(fe_1), λ(fe_2),..., λ(fe_m))
  on every n crawled pages
    recalculate interest factors λ(fe_i)
    reorder frontier
end
```

Interest factor

Inkrementalno ucenje. Menja strategijo glede na okolje. Lahko menja uteži cenilk.

Intelligent crawler*

- Features
  - Content based → $I_c(c)$
  - URL token based → $I_u(c)$
  - Link based → $I_i(c)$
  - Sibling based → $I_s(c)$

$$PriorityValue = w_c \log(I_c(c)) + w_u \log(I_u(c)) + w_i \log(I_i(c)) + w_s \log(I_s(c))$$

*Aggarwal, C., F. Al-Garawi, and P. Yu. *Intelligent crawling on the World Wide Web with arbitrary predicates.* In Proceedings of 10th Internaitonal **Conference on World Wide Web** (WWW-2001), 2001.

Vrednostne funkcije se lahko naucimo na podlagi preizkušanja.

Spodbujevalno ucenje je pocasno.

Reinforcement learning (RL)

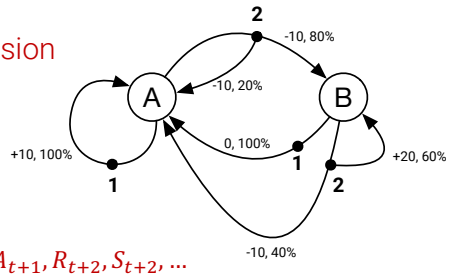
- Reinforcement learning (RL) is technique to find optimal solution by trial and error.
- RL imitates reward-motivated behavior.
- Simulation of real life learning
  - try different strategies
  - learn from the feedback
  - act according to the best strategy

Cilj fokusiranega pajka je da zbira relevantne strani in se izogiba irelevantnim. izbira najboljse povezave na dolgi rok (uteženi monte carlo).

Based on R. Sutton and A. g. Barto, **Reinforcement Learning: An Introduction**, second edition, The MIT Press, 2018.

RL – modeling environment

- Environment: A finite Markov Decision Process (MDP)
  - Discrete time t = 1, 2, 3
  - A finite set of states
  - A finite set of actions
  - A finite set of rewards
  - Life is trajectory $\dots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \dots$ with Markov dynamics:



$$p(r, s' \mid s, a) = Prob[R_{t+1} = r, S_{t+1} = s' \mid St = s, At = a]$$

RL – policies

- Policy π – defines for each state s what action a to take
- Deterministic policy: $a = \pi(s)$
- An agent following a policy: $A_t = \pi(S_t)$

- The agent's goal is to choose actions A_t in a way to maximize the discounted sum of future rewards:

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

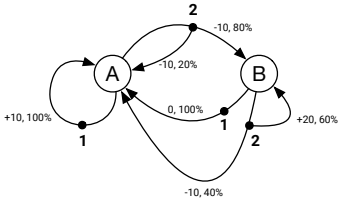
RL – Action-value functions

- An action-value function tells how good is to be in a state, take an action and then follow a policy:

$$q_\pi(s, a) \approx \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

Action-value function

State	Action	Value
A	1	130,39
A	2	133,77
B	1	166,23
B	2	146,23



RL – State value function

- A state value function tells, how good it is for the agent to be in a certain state.
- The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.
- For MDPs, we can define v_π as:

$$v_\pi(s) \approx \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]$$

RL – Optimal policies…

- An optimal policy π_* is better than other policies. It maximizes the state and action-value functions:

$$q_{\pi*}(s,a) = \max_{\pi} q_{\pi}(s,a), \quad v_{\pi*}(s) = \max_{\pi} v_{\pi}(s)$$

- Bellman optimally equations:

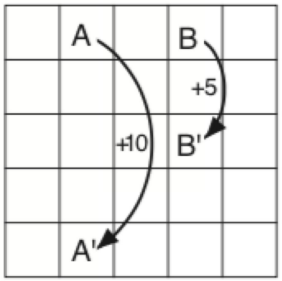
$$v_{\pi*}(s) = \max_{a} q_{\pi*}(s,a) = \max_{a} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')],$$

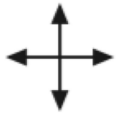
$$q_{\pi*}(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_{\pi*}(s',a')\right]$$

RL – Optimal policies

- Given the optimal value functions, it is easy to act optimally:
  - If we have $v_{\pi*}$ than we just need to check which states that are reachable from the current state gives the best value.
  - If we have $q_{\pi*}$ than we just select the best action.
- Optimal policy is greedy with respect to the optimal value functions.

Gridworld example…



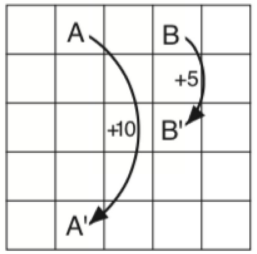


Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

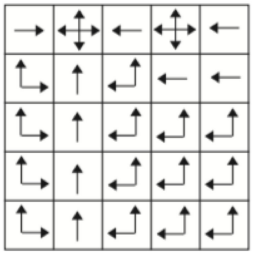
Random action policy

Gridworld example



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

Optimal state-value function



Optimal policy

Q-learning

- The simplest RL algorithm:
  - Initialize an array $Q(s,a)$
  - Choose actions in any way (e.g. to take all states several times)
  - On each time step, change one element of the array:

$$\Delta Q(S_t, A_t) = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

  - If desired, reduce step-size parameter α over time.
- Theorem*: for appropriate choice of α, Q converges to $q_{\pi*}$ and its greedy policy to an optimal policy π_* (Watkins & Dayan, 1992).

*Watkins, C. J. C. H., Dayan, P. (1992). *Q-learning*. **Machine Learning**, 8:279–292.

MDPs with Prioritizing updates

- Using prioritized URLs, focused crawler would select best link to go – no matter of its current position!
- Crawling agent has to follow this principle.
- Prioritized updates

to update state s with change δ in $U(s)$:
  update $U(s)$
  priority of $s \leftarrow 0$
  for each predecessor s' of s:
    priority $s' \leftarrow$ max of current and $\max_a \delta \hat{T}(s', as')$

Ker je premalo prostora, da bi hranili celotno tabelo za q-ucenje.

Linear Function Approximation

- In large state-action spaces, a tabular representation of action-value functions impossible.
- Action values are represented with linear function approximation:

$$\hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s, a) = \sum_{i=1}^{d} w_i x_i(s, a)$$

where $\boldsymbol{x}(s,a)$ is a feature vector and $\boldsymbol{w}$ weight vector.

Spremenimo uteži na tak nain, da bo razlika med dvema korakoma minimalna. Smeri doloamo po gradientu.

Gradient descent update on SARSA

- At each time step, weight vector $\boldsymbol{w}$ is updated.
- Gradient descent method is used (in direction that minimizes TD error):

Partial derivatives of the function

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha [r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w}_t) - \hat{q}(s_t, a_t, \boldsymbol{w}_t)] \nabla \hat{q}(s_t, a_t, \boldsymbol{w}_t)$$

TD or Bellman error: difference between the action value in two successive states, t and t+1. "

Evaluation setting…

- Crawling task starts with a seed page and terminates when n pages are visited (n set to 10.000).
- At each time step, page is crawled and analyzed (cosine and word2vec similarity measures).
- Rewards:
  - 30 if TFIDS similarity > 0
  - 30 if TFIDF=0 and word2vec > 0,5
  - 20 if TFIDF=0 and word2vec > 0,4
  - 1, otherwise

Evaluation setting

- Environment: database dump of Simple English Wikipedia.
- Target topics:
  - Representative: Fiction, Olympics, Cancer
  - Sparse: Cameras, Geology, Poetry
- Seed pages: main pages of each category
- Parameters:
  - ε = 0.1, discount rate γ = 0.9, step size α = 0.001.

Evaluation of crawlers

- How to compare crawling algorithms?
  - comparing applications
  - comparing algorithms
- Requirements:
  - sufficient number of runs over different topics
  - sound methodologies to consider temporal nature of results
- Challenges
  - unavailability of relevant sets for particular topics/queries.

Relevance estimation

- Estimation of page relevance (and quality)
  - number of keywords in the page
  - frequency of keywords
  - cosine similarity
  - classifier-based
  - N crawlers from same seeds until p pages retrieved
  - PageRank, HITS

Web Information Extraction and Retrieval

Marko Bajec and Slavko Žitnik
Laboratory for Data Technologies, UL FRI

Course content

- Module 1: Web crawling
- Module 2: Structured data extraction
- Module 3: Information Retrieval and Web Search

- Optional: Information Integration

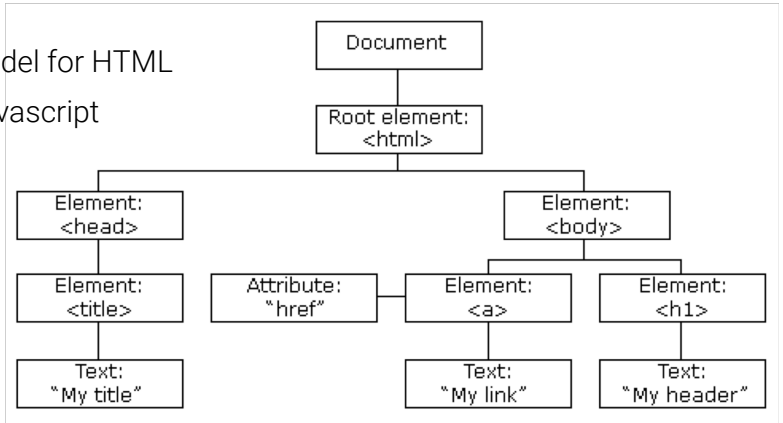
primarjalnik cen: -v živo- za vsako trgovino ovojnica, ko uporabnik napiše poizvedbo, gremo ez trgovino pridobimo podatke
-odloženi pristop - cene shranjene v bai, redno hodi pajek okoli

Module 2: Structured data extraction

ovojnica(wrapper) - algoritem/postopek/pravila za ekstrakcijo podatkov
scraping - proces ekstrakcije podatkov

Document object model (DOM)

- Object model for HTML
- API for Javascript



Web page pre-processing

- Different text fields identification
- Anchor text identification
- HTML tags removal
- Main content blocks identification
  - Approaches:
    - Visual cues-based partitioning
    - Tree matching

String matching - da ugotovimo kje je kakšen vsebinski blok
tree matching - kateri vsebinski bloki pašejo skupaj

String matching and tree matching

- Web data extraction
  - Finding the encoding template from a collection of encoded instances of the same type
- Detection of repeated patterns from HTML strings
  - String matching
  - Tree matching

String edit distance

- Vladimir Levenshtein, 1965
- Minumum number of point mutations
  - character change
  - character insertion
  - character deletion
- Distance $d(s_1, s_2)$ of strings s_1 and s_2

String edit distance

- Distance $d(s_1, s_2)$ of strings s_1 and s_2

$$d(\epsilon, \epsilon) = 0$$
$$d(s, \epsilon) = d(\epsilon, s) = |s|$$

$$d(s_{1-} + c_1, s_{2-} + c_2) = min \begin{cases} d(s_{1-}, s_{2-}) + p(c_1, c_2) \\ d(s_{1-} + c_1, s_{2-}) + 1 \\ d(s_{1-}, s_{2-} + c_2) + 1 \end{cases}$$

String edit distance

- Time complexity – $O(|s_1||s_2|)$
- Space complexity – $O(|s_1||s_2|)$

- Normalized distance

$$ND(s_1, s_2) = \frac{d(s_1,s_2)}{(|s_1|+|s_2|)/2}$$

Transpositions - zamenjave npr website in wabtise ima 2 zamenjavi

Jaro similarity

- Matthew A. Jaro, 1989
- Similarity sim_j of strings s_1 and s_2

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

- m – matching characters if not farther than $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$
- t – half the number of transpositions not farther than $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$

Jaro-Winkler similarity

- William E. Winkler, 1990
- Variant of Jaro, favorable to strings with matching prefix l
- Similarity sim_w of strings s_1 and s_2

$$sim_w = sim_j + l \cdot p \cdot (1 - sim_j)$$

- l – length of common prefix up to 4 characters
- p – scaling factor defined as 0.1

Kako podobna sta si drevesa med seboj - kateri deli drevesa so enaki oz. podobni med seboj.

Tree edit distance matching algorithms

- A variation of classic string edit distance problem
- Let A and B be two ordered rooted trees. The problem is to find a sequence of operations to transform A into B (or the vice versa).

- Minimize the (weighted) cost of operations:
  - Node deletion
  - Node insertion
  - Node replacement

Tree edit distance matching algorithms

- Let $A[i_x]$ indicate the x-th node of the tree A in a pre-order visit.
- A mapping M between A and B is a set of ordered pairs (i, j), $i \in A, j \in B$, satisfying $\forall (i_1, j_1), (i_2, j_2) \in M$:
  - $i_1 = i_2$ if and only if $j_1 = j_2$
  - $A[i_1]$ is on the left of $A[i_2]$ if and only if $B[j_1]$ is on the left of $B[j_2]$
  - $A[i_1]$ is an ancestor of $A[i_2]$ if and only if $B[j_1]$ is an ancestor of $B[j_2]$

in order(l, root, r), preorder(root, l , r), postorder(l,r,root)

The simple tree matching algorithm

- A computationally efficient solution for tree edit distance matching.
- Time complexity: $O(\text{nodes}(A) \cdot \text{nodes}(B))$
- Normalized version of the algorithm:

$$NSTM(A, B) = \frac{\text{SimpleTreeMatching}(A, B)}{(\text{nodes}(A) + \text{nodes}(B))/2}$$

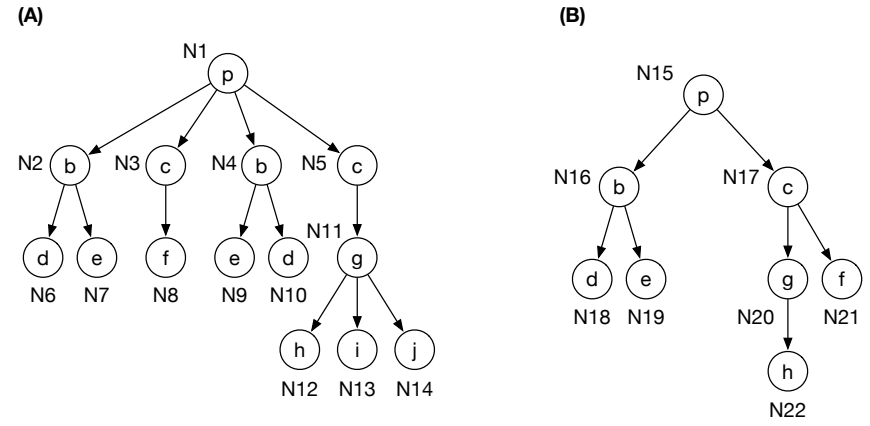
The simple tree matching algorithm

Algorithm: SimpleTreeMatching(T', T'')

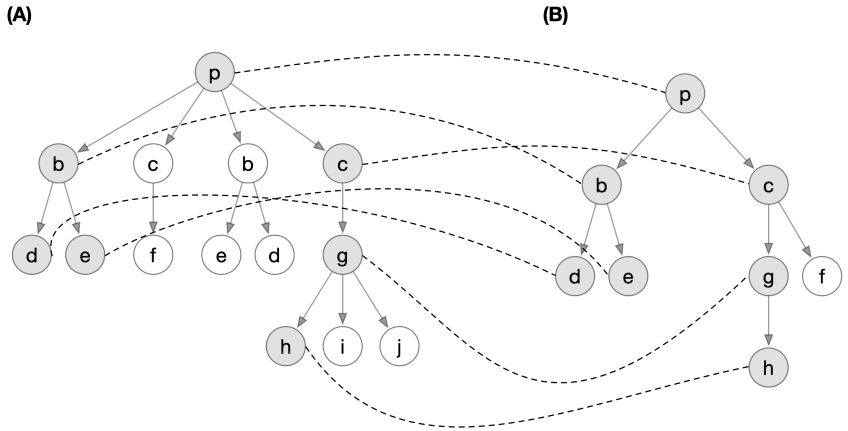
```
1  if T' has the same label of T'' then
2    m ← d(T')
3    n ← d(T'')
4    for i = 0 to m do
5      M[i][0] ← 0
6    for j = 0 to n do
7      M[0][j] ← 0
8    forall i such that 1 ≤ i ≤ m do
9      forall j such that 1 ≤ i ≤ n do
10       W_ij = SimpleTreeMatching(T'(i − 1), T''(j − 1))
11       M[i][j] ← Max(M[i][j − 1], M[i − 1][j], M[i − 1][j − 1] + W_ij)
12   return M[m][n] + 1
13 else
14   return 0
```

$d(n)$ – degree of node n
$T(i)$ – i-th subtree of node T

Tree matching algorithm - example



Tree matching algorithm - example



The weighted tree matching algorithm

Algorithm: WeightedTreeMatching(T', T'')
The code for the algorithm is similar to the *SimpleTreeMatching* algorithm, where you change the line 12 with the following code:

```
1 if m > 0 AND n > 0 then
2    return M[m][n] * 1/Max(t(T'),t(T''))
3 else
4    return M[m][n] + 1/Max(t(T'),t(T''))
```

- $\mathbf{t(n)}$ – number of total siblings of node $\mathbf{n}$ including itself.

Multiple alignment

- Finding repeated patterns from HTML strings

  Usklajuje se vec objektov na strani med seboj.

- Approaches
  - Optimal multiple alignment (Carrillo and Lipman, 1988)
  - Center star method (Gusfield, 1997)
  - Partial tree alignment method (Zhai and Liu, 2005)
  - ...

Center star method

- Classic technique
- Used for multiple string alignment, can be adopted to trees.

- Let $d(s_c, s_i)$ be the distance of two strings and S the set of strings for alignment. Then a string s_c - center string is selected as:

$$s_c = \arg\min_{s_c} \sum_{s_i \in S} d(s_c, s_i)$$

- The algorithm iteratively computes the alignment of other strings with s_c.

Center star method

Algorithm: CenterStar(S)

1 choose the center star s_c
2 $M = List(s_c)$
3 **for** each s in $S - \{s_c\}$ **do**
4 $c* =$ aligned version of $s_c \in M$
5 $s', c*' =$ optimally aligned strings of $s$and $c*$
6 update strings in M with spaces where added to $c*$
7 M.add(s')
8 M.add($c*'$)
9 **return** multiple string alignment M

Center star method

- Time complexity: $O(k^2n^2)$

- Shortcomings
  - slow for pages with many data records / many tags
  - if s_c does not have a particular data item, other records might not be properly aligned

Partial tree alignment

- Used for multiple tree alignment
  - Can be adopted to strings
  - Solves the *Center star method* shortcomings

- Idea
  - progressively growing a *seed tree* T_s
  - for each T_i $(i \neq s)$ find matching nodes otherwise insert (*)

- Partial – if insertion cannot be done, it is left unmatched

Partial tree alignment – two trees

- Some nodes in T_i can be matched to nodes in T_s
- Non-matched nodes from T_i can contain optional items, thus v_i is inserted into T_s
  - Insertion only if a location in T_s can be *uniquely determined*

- Let $v_j \dots v_m$ be unmatched consecutive sibling nodes from T_i:
  - Parent of $v_j \dots v_m$ has a match in T_s, where nodes should be inserted
  - Inserted only if position can be uniquely determined, otherwise not inserted

Partial tree alignment – two trees

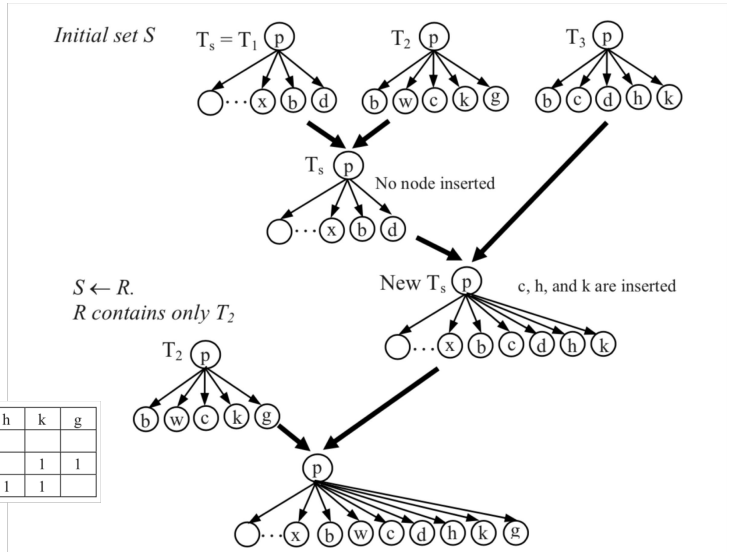
- Location for inserting $v_j \dots v_m$ from T_i to T_s uniquely decided if:
  - If $v_j \dots v_m$ have two neighboring siblings in T_i, one on the right and one on the left, that are matched with two consecutive siblings in T_s .
  - If $v_j \dots v_m$ has only one left neighboring sibling x in T_i and x matches the right most node x in T_s, then $v_j \dots v_m$ can be inserted after node x in T_s.
  - If $v_j \dots v_m$ has only one right neighboring sibling x in T_i and it matches the left most node x in T_s, then $v_j \dots v_m$ can be inserted before node x in T_s.

Partial tree alignment – multiple trees

Algorithm: PartialTreeAlignment(S)

```
1  Sort trees in S in descending order of the number of data items
2  T_s ← select and delete the first tree from S
3  R ← ∅
4  while S ≠ ∅ do
5     T ← select and delete next tree from S
6     STM(T_s, T_i)
7     AlignTrees(T_s, T_i)
8     if T_i is not completely aligned with T_s then
9        if InsertIntoSeed(T_s, T_i) then
10          S ← S ∪ R
11          R ← ∅
12       if still unaligned items in T_i that are not inserted into T_s then
13          R ← R ∪ {T_i}
14 if R ≠ ∅ then
15    PartialTreeAlignment(R)
16 return data table of alignment results for each T_i
```


Partial tree alignment – multiple trees (demo)



	…	x	b	w	c	d	h	k	g
T_1	…	1	1			1			
T_2			1	1	1			1	1
T_3			1		1	1	1	1	

Building DOM/tag trees

- Necessary for many data extraction algorithms
- Approaches:
  - Using tags only
  - Using tags and visual cues

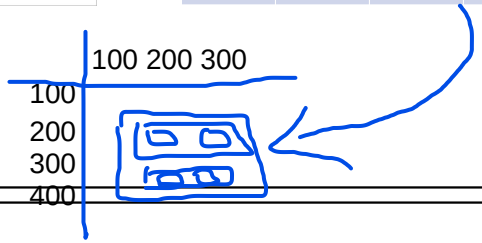
1. HTML cleaning(<li>, <hr>,
, nepravilno zaprti tagi) - rešimo z beautifulsoup
2. Tree building (1. najdemo robove elementov, vsaka vsebovanost predstavlja dedovanje, za vsak slucaj lahko renderiramo)

Building DOM/tag trees

- Tags and visual cues example

```
1   <table border="2">
2       <tr>
3           <td> data1 <td>
4           </td> data2 </td>
5       <tr>
6           <td> data3 </td>
7           <td> data4
8       <tr>
9   </table>
```

left	right	top	bottom
100	300	200	400
100	300	200	300
100	200	200	300
200	300	200	300
100	300	300	400
100	200	300	400
100	300	300	400

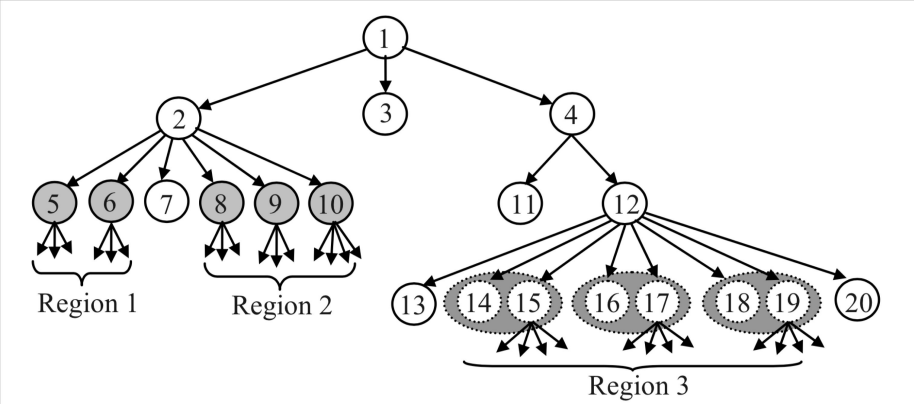


Data records use the same HTML markup encoding.
Da dobimo sezname išemo podobna drevesa znotraj DOM.

Mining data regions

- Mining data regions containing a list of data records
- A *generalized node* (node combination) of length r ($r > 1$):
  - the nodes have the same parent
  - the nodes are adjacent
- A *data region* – a collection of two or more generalized nodes:
  - the generalized nodes have the same parent
  - the generalized nodes have the same length
  - the generalized nodes are adjacent
  - the similarity between generalized nodes is greater than a fixed threshold

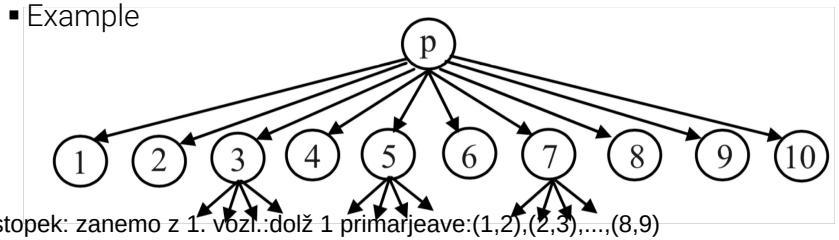
Mining data regions



k<10 - empirino, koliko naporednih tagov ima generalised node.

Generalized nodes comparison

- Goals:
  - Where does a first generalized node start?
  - How many consecutive tags does a generalized node have?
- Example



postopek: zanemo z 1. vozl.:dolž 1 primarjeave:(1,2),(2,3),...,(8,9)
dolž. 2 primerjave: (12, 34),..(78, 9 10)
dolž 3 psrimerjave: (123, 456), (456, 789)
nato za 2. vozl:dolz 2:(23, 45), (45, 67), (67,89)
dolz 3: (234, 567), (567, 89 10) ITD.
vsake izmed njih preverimo, ali obstajajo kakšni podobni med seboj, kar skrajša dolžino ki jo je treba preveriti.

Automatic web extraction issues:
-Extraction based on encoding template(finite state machines, pattern matching, extracting simmilar pages)
-Detecting new templates
-Disjunction or optional values(kako razlikovati med 100€ in out of stock)
-list of set type or attribute type
-Labeling and integration
-Extraction of unwanted data

Web wrappers

web wrappers - procedures leveraging tree matching to perform data extraction they can be semi or fully automatic.
wrapper lifecycle: 1.Generation 2. Extraction 3. Maitenence

```
<persons>
  <person>
    <name>John</name>
    <height unit=”cm”>191</height>
    <sport>Running</sport>
    <sport>Cycling</sport>
  </person>
  <person>
    <name>Mandy</name>
    <height>140</height>
   <sport>Swimming</sport>
  </person>
</persons>
```

XPath

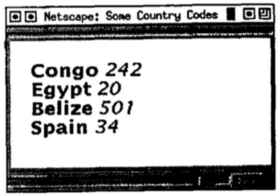
Examples:

- `/persons/person/name`
- `/persons/person/name/text()`
- `//person/height[@unit="cm"]/text()`
- `//sport`
- `//person[name="Mandy"]/sport/text()`

ML-based approach (WIEN, 2000)

(a)

(b)



(c) $P_{CC} =$

```
<HTML><TITLE>Some Country Codes</TITLE><BODY>
<B>Congo</B> <I>242</I><BR>
<B>Egypt</B> <I>20</I><BR>
<B>Belize</B> <I>501</I><BR>
<B>Spain</B> <I>34</I><BR>
</BODY></HTML>
```

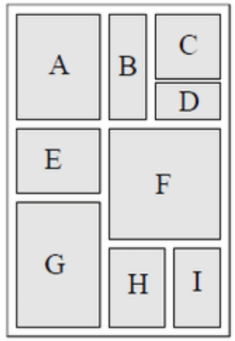
(d) $L_{CC} = \{ \langle \text{'Congo', '242'} \rangle, \langle \text{'Egypt', '20'} \rangle, \langle \text{'Belize', '501'} \rangle, \langle \text{'Spain', '34'} \rangle \}$

(e)

procedure ccwrap_{LR}(page P)
  while there are more occurrences in P of '<B>'
    for each $\langle \ell_k, r_k \rangle \in \{\langle \text{'<B>', '</B>'} \rangle, \langle \text{'<I>', '</I>'} \rangle\}$
      scan in P to next occurrence of ℓ_k; save position as start of k^{th} attribute
      scan in P to next occurrence of r_k; save position as end of k^{th} attribute
  return extracted pairs $\{\ldots, \langle \text{country, code} \rangle, \ldots\}$

Rendered web page cut into a visual grid(elements allocated according to coordinates)
Cuts then applied recursively and stored into a X-Y tree until non-empty leaves.
We start with a vertical cut.

X-Y cut algorithm



Example by *Simone Marinai*: Reflowing and Annotating Scientific Papers on eBook Readers, 2013

Information retrieval models

- IR model
  - how to represent documents and queries?
  - how to compute relevance of a document d to a query q?
- Main IR models:
  - Boolean model,
  - vector space model,
  - language model,
  - probabilistic model.

Bag of words -> na besede gledajo kot neodvisne enote

IR model formalization

- D – collection of documents,
- $V = \{t_1, t_2, \dots, t_{|V|}\}$ – set of distinctive terms t_i in D – called vocabulary of D with size $|V|$,
- Each term t_i of a document $\boldsymbol{d_j} \in D$ has a weight $w_{ij} > 0$.
- Each document $\boldsymbol{d_j}$ is represented with a term vector: $\boldsymbol{d_j} = (w_{1j}, w_{2j}, \dots, w_{|V|j})$, where w_{ij} corresponds to $t_i \in \mathrm{V}$ signifying the importance of t_i in $\boldsymbol{d_j}$.

Je zelo hiter, primeren za uporabo, kjer iščemo tocno te besede v poizvedbi.

Pri tem modelu ne moremo dobro rangirati, ker lahko govorimo samo v vsebovanosti.

Boolean IR model

- Simplest IR model, queries and documents represented as sets of terms (with vectors):

  $q = \{t_i, tp, tj\}$, where $t_i, t_p, t_j \in V$

	t_1	t_2	...	t_i	...	t_p	...	t_j	...	$t_{\|V\|}$
q	0	0	...	1	...	1	...	1	...	0

- Retrieval:
  - Exact matching based on Boolean algebra (AND, OR, NOT)

  `query:` `computer NOT science`

Najvecjo težo po TF-IDF imajo besede, ki sse velikokrat pojavijo v dokumentu, a malokrat v drugih dokumentih.

Vector space model...

- Best known and widely used in IR
- Documents represented as weight vectors
- Variations of TF/TF-IDF used for weight calculation
- TF – term frequency:
  - $w_{ij} = f_{ij}$ where f_{ij} number of times t_i appears in $\boldsymbol{d_j}$.
  - TF can be normalized, e.g. Euclidean normalization:

$$tf_{ij} = \frac{f_{ij}}{\sqrt{f_{1j}^2 + f_{1j}^2 + \cdots + f_{|V|j}^2}}$$

TF-IDF scheme

- TF: non discriminative words (e.g. "the", "this",...) can have high weights.
- TF-IDF scheme relativizes terms appearing in many documents.
- Computed e.g. as $idf_i = log \frac{N}{df_i}$
  - N total number of documents,
  - d_{fi} the number of documents with t_i.
- According to TF-IDF, $w_{ij} = tf_{ij} * idf_{ij}$

TF-IDF for queries

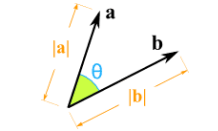
- Number of variations for calculating IDF exists.
- How about queries?
- Sulton and Buckley (1988)* suggests:

$$w_{ij} = \left(0.5 + \frac{0.5 * f_{ij}}{\max\{f_{1j}, f_{2j}, \dots, f_{|V|j}\}}\right) * log \frac{N}{df_i}$$

*Salton, G. and C. Buckley. *Term-weighting approaches in automatic text retrieval.* **Information Processing & Management**, 1988, 24(5): p. 513-523.

Relevance ranking in vector space model

- Documents relevance ranking:
  - how relevant is $\boldsymbol{d}_j$ to q?
  - many variants to measure relevance
- Dot product: $sim(\boldsymbol{d}_j, \boldsymbol{q}) = \langle \boldsymbol{d}_j \cdot \boldsymbol{q} \rangle$
- Cosine similarity:



$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \times |\mathbf{b}| \times \cos(\theta)$

$\mathbf{a} \cdot \mathbf{b} = a_x \times b_x + a_y \times b_y$

$$cosine(\boldsymbol{d}_j, \boldsymbol{q}) = \frac{\langle \boldsymbol{d}_j \cdot \boldsymbol{q} \rangle}{\|\boldsymbol{d}_j\| * \|\boldsymbol{q}\|} = \frac{\sum_{i=1}^{|V|} w_{ij} * w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} * \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

Boljša za kot cosine simmilarity se posebaj za kratke poizvedbe

Okapi method

- Variation of a relevance measure, Okapi method*:

$$okapi(d_j, q) = \sum_{t_i \in q, d_j} ln \frac{N - df_i + 0.5}{df_i + 0.5} * \frac{(k_1+1) f_{ij}}{k_1\left(1 - b + b\frac{dl_j}{avdl}\right) + f_{ij}} * \frac{(k_2+1) f_{iq}}{k_2 + f_{iq}}$$

  - where:
    - df_i is the number of documents that contain the term t_i
    - dl_j is the document length (in bytes) of d_j
    - $avdl$ is the average document length of the collection
    - Parameters: k_1 (between 1.0-2.0), b (usually 0.75) and k_2 (between 1-1000).

*Robertson, S., S. Walker, and M. Beaulieu. *Okapi at TREC-7: automatic ad hoc filtering,* **VLC and interactive track. NIST Special Publications**, 1999: p. 253-264.

Statistical language model

- Based on probability, funded on statistical theory.
- Idea:
  - for each document d estimate a language model
  - rank documents by the likelihood of the query q given the language model.
- Similar approaches used in NLP and speech recognition.
- First proposed by Ponte and Croft*

*Ponte, J. and W. Croft. *A language modeling approach to information retrieval*. In Proceedings of **ACM SIGIR Conf. on Research and Development in Information Retrieval** (SIGIR-1998), 1998..

Formalization

- We are interested in estimating $\Pr(q|d_j)$, where
  - $q = q_1 q_2 \dots q_m$ is a query
  - D is document collection and $d_j \in D$
- Typically based on unigrams ->besede so med seboj neodvisne - to delamo zaradi hitrosti.
- Better models consider n-grams

Multinominal distribution

- Multinominal distribution over unigrams (words):

$$Pr(q|d_j) = \prod_{i=1}^{m} \Pr(q_i|d_j) = \prod_{i=1}^{|V|} \Pr(t_i|d_j)^{f_{iq}}$$

  - where
    - $q = q_1 q_2 \dots q_m$,
    - V is vocabulary of d_j
    - $t_i \in V$ and f_{iq} tells the number of times t_i appears in q
    - $\Pr(t_i|d_j) = \frac{f_{ij}}{|d_j|}$ and $\sum_{i=1}^{|V|} \Pr(t_i|d_j) = 1$

Laplace and Lindstone smoothing

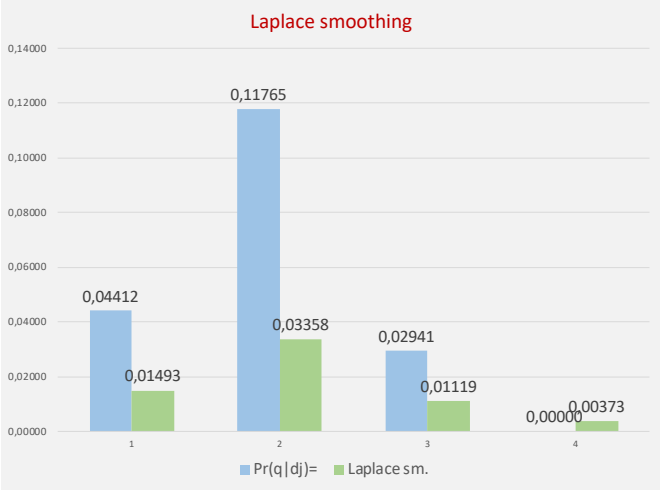
- What if $t_i \in q$ and $t_i \notin d_j \Longrightarrow \Pr(t_i|d_j) = 0$?
- Additive smoothing:

$$Pr_{add}(t_i|d_j) = \frac{\lambda + f_{ij}}{\lambda|V| + |d_j|}$$

- Laplace smoothing: $\lambda = 1$
- Lindstone smoothing: $0 < \lambda < 1$

Example

- $q = t_1 t_2 t_3 t_4$
- $|V| = 200$
- $|d_j| = 68$
- $f_{1j} = 3$
- $f_{2j} = 8$
- $f_{3j} = 2$
- $f_{4j} = 0$



Relevance feedback

- Improving IR based on user's feedback
- Rocchio method:
  - let $\boldsymbol{q}$ be original query vector and
  - D_r the set of relevant and D_{ir} the set of irrelevant documents.
  - The expanded query $\boldsymbol{q}_e$ can be calculated as:

$$\boldsymbol{q}_e = \alpha \boldsymbol{q} + \frac{\beta}{|D_r|} \sum_{d_r \in D_r} \boldsymbol{d}_r - \frac{\gamma}{|D_{ir}|} \sum_{d_{ir} \in D_{ir}} \boldsymbol{d}_{ir}$$

  - where α, β and γ parameters

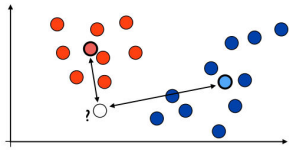
alfa - moc originalne opizvedbe
beta - moc relevantnih dokumentov
gama - moc irelevantnih dokumentov

Rocchio classification

- Supervised machine learning on feedback – no comparission with query.
- Prototype vectors c_i built for each class:

$$\boldsymbol{c}_i = \frac{\alpha}{|D_i|} \sum_{\boldsymbol{d} \in D_i} \frac{\boldsymbol{d}}{||\boldsymbol{d}||} - \frac{\beta}{|D - D_i|} \sum_{\boldsymbol{d} \in D - D_i} \frac{\boldsymbol{d}}{||\boldsymbol{d}||}$$

- where:
  - D_i set of documents of class i
  - α and β parameters



alfa - moc relevantnih dokumentov
beta - moc irelevantnih dokumentov

Vemo kaj nas zanima, ne vemo s katerimi kljucnimi besedami je najbolje opisati.

Word embeddings

- Representation of words in high-dimensional space
  - e.g. 300 – 500 dimensions
  - words with similar semantics/ usage appear closer...

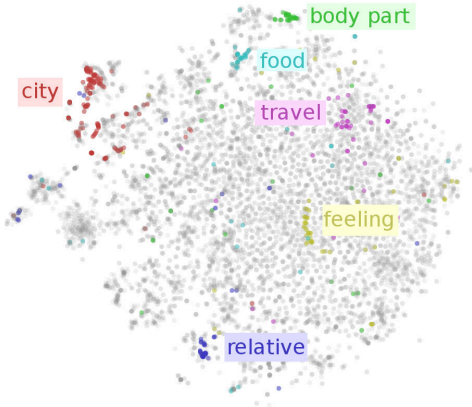


Image source: http://ruder.io/word-embeddings-1/

Vsaka beseda ima enko v slovarju tocno tam, kjer je tocno ta beseda iz slovarja.
To je uporabno za strojno ucenje za vektorje.

One hot encoding

- Encoding words with vocabulary words
  - vocabulary = {house, real-estate, rent, agency, provision, flat}
- One hot encoding

Word	v1	v2	v3	v4	v5	v6
house	1	0	0	0	0	0
real-estate	0	1	0	0	0	0
rent	0	0	1	0	0	0
agency	0	0	0	1	0	0
provision	0	0	0	0	1	0
flat	0	0	0	0	0	1

Document	v1	v2	v3	v4	v5	v6
real-estate agency	0	1	0	1	0	0

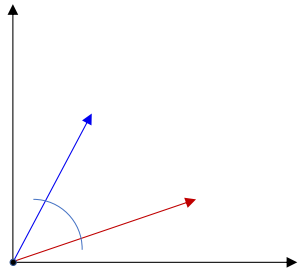
Problem je, da so vse besede ortagonalne.

Zato lahko gledamo glede na razdalje, ki so konstantne med podobnimi poizvedbami npr. kralj-> kraljica, moski->zenska

One hot encoding in vector space

- Corpus:
  - Romeo and Juliet.
  - Juliet: O happy dagger!
  - ...
- Vocabulary:
  - V = {romeo, juliet, happy, dagger,... },

Category	w_1	w_2	...	$w_{\|V\|}$
romeo	1	0	...	0
juliet	0	1	...	0
...				

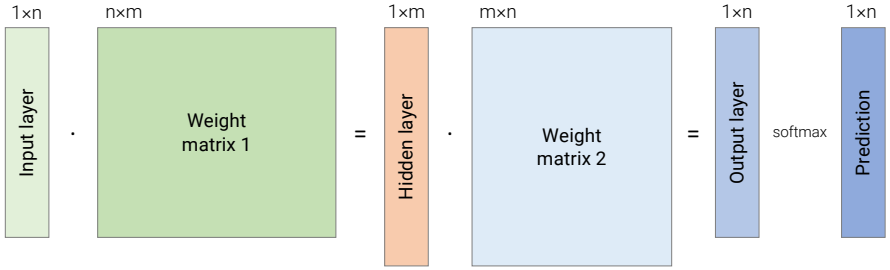


Word2Vec -> pretvori besede v vektorje za potrebe strojnega ucenja

Doc2Vec -> isto za dokumente

Learning word embeddings with NN

- Shallow NN



p(i) - pove, koliko do sedaj dokumentov je relevantnih
r(i) - pove, koliksen delez relevantnih dokumentov je bilo do sedaj

Evaluation of ranking

- Let D be a collection of documents with $|D| = N$.
- For a query q the retrieval algorithm produces ranking $R_q: \langle d_1^q, d_2^q, \dots, d_N^q \rangle$, where $d_1^q \in D$ most relevant and $d_N^q \in D$ most irrelevant document to q.
- P and R can be calculated for each rank position:
  - Recall at rank position $i:\ r(i) = \frac{S_i}{|D_q|}$,
  - Precision at rank position $i:\ p(i) = \frac{S_i}{i}$

$D_q \subseteq D$: set of actual relevant documents
S_i: number of relevant documents from position 1 to i.

Example of ranking at each position

Rank i	+/–	p(i)	r(i)
1	+	1/1 = 100%	1/8 = 13%
2	+	2/2 = 100%	2/8 = 25%
3	+	3/3 = 100%	3/8 = 38%
4	–	3/4 = 75%	3/8 = 38%
5	+	4/5 = 80%	4/8 = 50%
6	–	4/6 = 67%	4/8 = 50%
7	+	5/7 = 71%	5/8 = 63%
8	–	5/8 = 63%	5/8 = 63%
9	+	6/9 = 67%	6/8 = 75%
10	+	7/10 = 70%	7/8 = 88%
11	-	7/11 = 64%	7/8 = 88%
12	+	8/12 = 67%	8/8 = 100%

Average precision

Rank i	+/–	p(i)	r(i)
1	+	1/1 = 100%	1/8 = 13%
2	+	2/2 = 100%	2/8 = 25%
3	+	3/3 = 100%	3/8 = 38%
4	–	3/4 = 75%	3/8 = 38%
5	+	4/5 = 80%	4/8 = 50%
6	–	4/6 = 67%	4/8 = 50%
7	+	5/7 = 71%	5/8 = 63%
8	–	5/8 = 63%	5/8 = 63%
9	+	6/9 = 67%	6/8 = 75%
10	+	7/10 = 70%	7/8 = 88%
11	-	7/11 = 64%	7/8 = 88%
12	+	8/12 = 67%	8/8 = 100%

- Average precision:

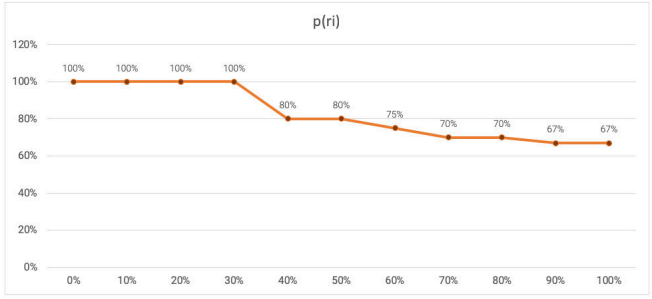
$$p_{avg} = \frac{\sum_{d_i^q \in D_q} p(i)}{|D_q|}$$

- $p_{avg} = \frac{100\% + 100\% + 100\% + 80\% + 71\% + 67\% + 70\% + 67\%}{8} = 82\%$

Precision-recall curve

- Plotted at standard recall levels: 0%, 10%, 20%, ..., 100%
- $p(r_i) = \max_{r_i \leq r \leq r_{10}} p(r)$

i	Relevant	$p(r_i)$	r_i
1	+	100%	13%
2	+	100%	25%
3	+	100%	38%
4	-	75%	38%
5	+	80%	50%
6	-	67%	50%
7	+	71%	63%
8	-	63%	63%
9	+	67%	75%
10	+	70%	88%
11	-	64%	88%
12	+	67%	100%



Povprecimo vse poizvedbe

Evaluation on multiple queries

- Overall precision on multiple queries:

$$\bar{p}(r_i) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} p_j(r_i)$$

- where:
  - $|Q|$ number of queries,
  - $p_j(r_i)$ precision q_j at the recall level r_i

Inverted, ker za vsak termin iscemo dokumente, ki ga vsebujejo ne obratno.

Inverted index

- Most efficient index for IR
- Consists of:
  - vocabulary V of distinctive terms of the document set $D = \{d_1, d_2, \dots, d_n\}$
  - for each term t_i an inverted index of postings.

$t_i \rightarrow \langle posting_1 \rangle, \langle posting_2 \rangle, \dots, \langle posting_n \rangle$

- Posting: doc id + other relevant information for retrieval

Example

- Example of inverted index for proximity search:
- $< id_j, f_{ij}, [o_1, o_2, \dots, o_{|f_{ij}|}] >$ where
  - f_{ij} - frequency of t_i in d_j
  - o_i - offset of t_i in d_j

Corpus:

id_1: Web mining is useful.
1 2 3 4

id_2: Usage mining applications.
1 2 3

id_3: Web structure mining studies the Web hyperlink structure.
1 2 3 4 5 6 7 8

Vocabulary:
{web, mining, useful, applications, usage, structure, studies, hyperlink}

Index:
- applications: $\langle id_2, 1, [3] \rangle$
- hyperlink: $\langle id_3, 1, [7] \rangle$
- mining: $\langle id_1, 1, [2] \rangle, \langle id_2, 1, [2] \rangle, \langle id_3, 1, [3] \rangle$
- structure: $\langle id_3, 2, [2, 8] \rangle$
- studies: $\langle id_3, 1, [4] \rangle$
- usage: $\langle id_2, 1, [1] \rangle$
- useful: $\langle id_1, 1, [4] \rangle$
- web: $\langle id_1, 1, [1] \rangle, \langle id_3, 2, [1, 6] \rangle$

Search on inverted index

- Algorithm:

```
for each query term q_i ∈ Q
   find q_i in vocabulary V
   retrieve list list_i of docs d_j where d_j ∈ D and q_i ∈ dj
end
merge doc lists list_i
for each doc d_j ∈ merge_list
   compute rank score
end
print ranked list of docs
```


Example

- Which document gets the highest rank in the proximity search for Q = web mining?

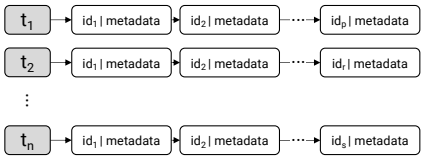
Corpus:

id_1: Web mining is useful.
id_2: Usage mining applications.
id_3: Web structure mining studies the Web hyperlink structure.

Index:
- applications: $\langle id_2, 1, [3] \rangle$
- hyperlink: $\langle id_3, 1, [7] \rangle$
- mining: $\langle id_1, 1, [2] \rangle, \langle id_2, 1, [2] \rangle, \langle id_3, 1, [3] \rangle$
- structure: $\langle id_3, 2, [2, 8] \rangle$
- studies: $\langle id_3, 1, [4] \rangle$
- usage: $\langle id_2, 1, [1] \rangle$
- useful: $\langle id_1, 1, [4] \rangle$
- web: $\langle id_1, 1, [1] \rangle, \langle id_3, 2, [1, 6] \rangle$

Query: web mining

Index construction



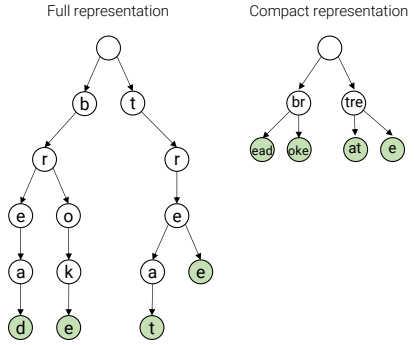
- Algorithm:

```
for each doc d_j ∈ D
   for each term t ∈ d_j
         find position e of t in index
         if t not found then create new element e in index
         add (ID, metadata) of d_j to index at position e
   end
end
```


korene besed damo notri da pohitrimo invertni indeks.
Mora imeti stop character.

Trie data structure

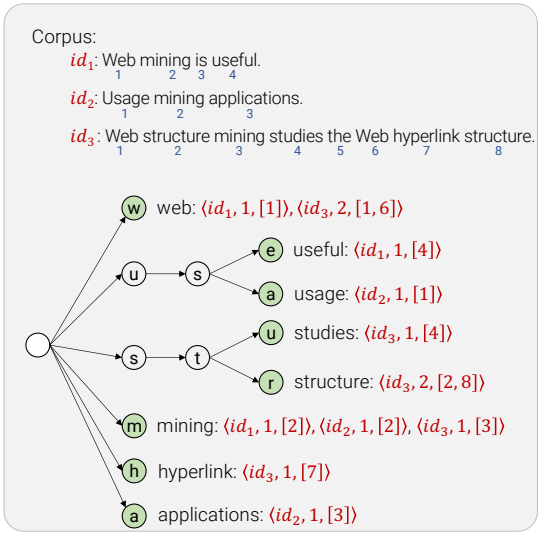
- Trie – retrieval
- Prefix tree with ordered content
- Nodes: string characters
- Example:
  - words: bread, broke, treat, tree
- Visualization:
  https://www.cs.usfca.edu/~galles/visualization/Trie.html



Example

- Build trie data structure for corpus: id_1, id_2, id_3
- Vocabulary:
  {applications, hyperlink, mining, structure, studies, usage, useful, web}

Za brisanje gremo do konca in brisemo nazaj do krizisca.



Krajsanje trie strukture gor primer

Ce hocemo notri dat "use", dodamo se eno vejitev na f in pustimo zeleno.

Time complexity

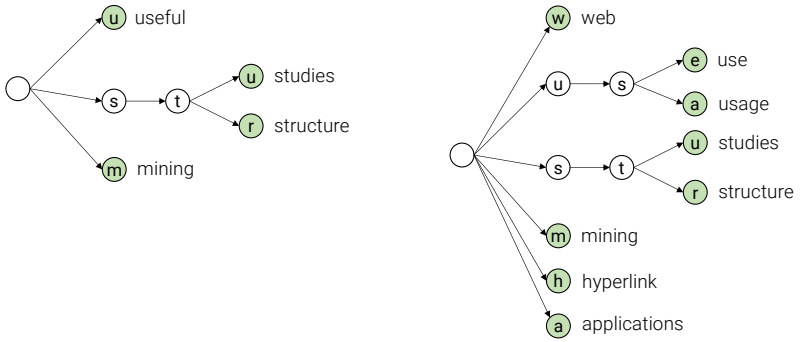
- Trie construction
  - Linear: $O(T)$, where T number of terms including duplicates
- Trie search
  - Linear: $O(n)$, where n length of a retrieved term

Trie construction for Web

- In Web environment, trie becomes very large.
- Changes in the construction algorithm:
  - write to a partial index I_i
  - when memory full, write partial index to disk
  - continue with new partial index
  - when all documents processed, merge indices:
    - pairwise (i.e. I_{1-2}, I_{3-4}, ..., I_{n-n+1} then I_{12-34}, ...) or
    - step by step (i.e. I_{1-2}, I_{12-3}, I_{123-4}, ...)

Drevesa so zelo razvejana brez velike globine(besede niso tako dolge).
To je se posebaj primerno za web, saj na casovno kompleksnost vpliva globina drevesa.

Trie merging



Obcasno obnavljamo glavni indeks, vedno posodabljamo D+ in D-. Posoben pristom pri ISRAM drevesih.

Additional indices

- Web dynamic, pages constantly added, deleted
- Additional indices:
  - index of added pages
  - index of deleted pages
- Algorithm:
  - search in main index → D_0
  - search in index of added pages → D_+
  - search in index of deleted pages → D_-
  - result: $(D_0 \cup D_+) - D_-$

Index compression

- Index compression needed to fit entire index or its large part to memory...
- Content of index represented with integers
- Basic idea:
  - Integers represented with 4 bytes (32 bits)
  - Many integers smaller than 4 bytes – use integer coding.
  - Doc ids can be very large numbers – use gaps (id offsets) instead.
    - Example, IDs: 4, 10, 300, and 305. Represented with offset, 4, 6, 290 and 5. Frequent terms have smaller gaps!

Types of integer coding

Coding type	Coding	Rules
Variable-bit scheme	Unary coding	for int x, prefix 1 with $x-1$ zeroes
	Elias Gamma coding	for int x, prefix 1 with $\lfloor log_2 x \rfloor$ zeroes and add x in binary without the most significant bit
	Elias Delta coding	for int x, represent $1+\lfloor log_2 x \rfloor$ in Gamma code and add x in binary without the most significant bit.
	Golomb coding	for int x, represent $q = \lfloor \frac{x}{b} \rfloor$ in unary and add binary representation of $r = x - qb$ using coding tree.
Variable-byte scheme	Variable-byte coding	Use seven bits in each byte to code int x, with the least significant bit set to 0 in the last byte, or to 1 if further bytes follow.

Exercise

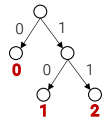
- Represent number 5 in variable-bit coding.

Unary: 00001

Elias gamma: $\text{concat}(001, 01) = 00101$
$\lfloor log_2 5 \rfloor = 2, \text{binary}(5) = 101$

Elias Delta: $\text{concat}(011, 01) = 01101$
$1 + \lfloor log_2 5 \rfloor = 3, \text{gamma}(3) = 011, \text{binary}(5) = 101$

Golomb (b=3): $\text{concat}(01, 11) = 0111$
$\text{unary}(\lfloor \frac{5}{3} \rfloor) = 01, \text{r} = 5 - \lfloor \frac{5}{3} \rfloor 3 = 2, \text{i} = \lfloor log_2 3 \rfloor = 1,$
$d = 2^{i+1} - b = 4 - 3 = 1, coding\_tree(r) = coding_{tree}(2) = 11$



Exercise

- Represent number 135 in variable-byte coding.

135 with 4-bytes binary: $00000000\ 00000000\ 00000000\ 10000111$

135 with variable-byte coding: $00000011\ 00001110$

Decoding...

- Elias Gamma
  - count zeroes - K
  - consider first one after K zeroes as the first digit of the integer, i.e. 2^K
  - read the remaining K bits of the integer.
- Elias Delta
  - count zeroes - L
  - consider first one after L zeroes as the first digit of the integer, i.e. 2^L
  - read the remaining L bits of the integer M.
  - use $M-1$ bits and add one on the first place.

Decoding

- Golomb
  - decode unary-coded quotient q.
  - compute $i = \lfloor log_2 b \rfloor$ and $d = 2^{i+1} - b$.
  - retrieve the next i bits and assign it to r.
  - if $r \geq d$ then
    - retrieve one more bit and append it to r at the end;
    - $r = r - d$.
  - return $x = qb + r$.
- Variable byte
  - read all bytes until a byte with the zero last bit is seen.
  - remove the least significant bit from each byte read so far and concatenate the remaining bits.

Exercise

- count zeroes - L
- consider first one after L zeroes as the first digit of the integer, i.e. 2^L
- read the remaining L bits of the integer M.
- use $M - 1$ bits and add one on the first place.

- Decode 00100001 written in Elias Delta coding.

Number of zeroes: 2

M = first one and remaining 2 bits: $100 = 4$

Result = one plus $M - 1$ leftmost bits: $1001 = 9$

Empirical tests

- Unary – nonefficient for large numbers, space consuming
- Parametrized Golomb more space efficient and faster for IR than non-parameterized Elias coding.
- Variable-byte integers often faster than variable-bit integers (fewer CPU operations required for decoding).
- A suitable compression allow retrieval to be up to twice as fast than without compression, while the space requirement averages 20% – 25% of the cost of storing uncompressed integers.

kodiranje je hitrejse, ker tudi s kodiranjem vred manj beremo.

Latent Semantic Indexing

- IR models typically based on keyword matching.
- Same concepts can be described with different words (synonyms)... important docs might not be retrieved.
- Latent Semantic Indexing* (LSI) identifies statistical associations among terms.
- Based on Singular Value Decomposition (SVD).

*Deerwester, S., S. Dumais, G. Furnas, T. Landauer, and R. Harshman. *Indexing by latent semantic analysis*. **Journal of the American Society for Information Science**, 1990, 41(6): p. 391-407.

Formalization

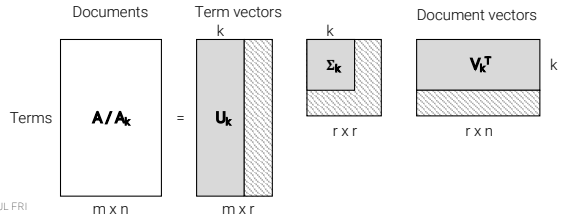
- Let D be the document collection, with m distinctive words and size $|D| = n$.
- Size of term-document matrix A is $m \times n$.
- Let A_{ij} be computed as TF, i.e. the number of times term i occurs in document j.

Singular Value Decomposition

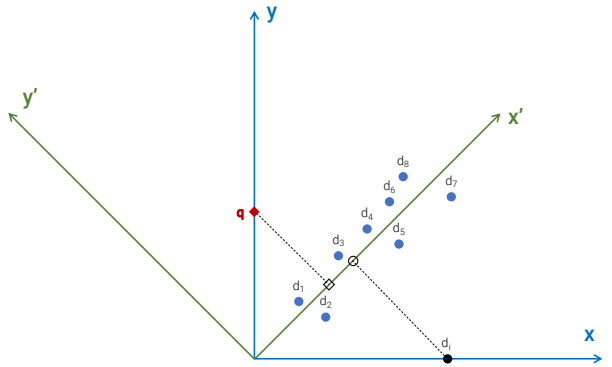
- SVD factors A into the product of 3 matrices: $A = U\Sigma V^T$, where:
  - U is a $m \times r$ matrix with columns that are unit orthogonal vectors called left singular vectors. They are eigenvectors associated with the r non-zero eigenvalues of AA^T.
  - V is a $n \times r$ matrix with columns that are unit orthogonal vectors called right singular vectors. They are eigenvectors associated with the r non-zero eigenvalues of A^TA.
  - Σ is a $r \times r$ diagonal matrix, $\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_r)$, $\sigma_i > 0$. σ_i called singular values, are the non-negative square roots of the r eigenvalues of AA^T (arranged in decreasing order).
  - r represents rank of A, $r \leq \min(m, n)$

Reducing hidden concept space

- Matrix Σ can be reduced by removing dimensions associated with smallest singular values.
- Let's say we use only k largest singular values in Σ. This leads us to the k-concept space: $A_{k=}U_k\Sigma_k V_k^T$



Intuition of LSI



Query and retrieval using LSI

- q treated as a document and transformed to k-concept space
- Transformed documents are represented with V_k.
- q_k becomes an additional column/document in V_k^T. Thus q is:

$$q = U_k \Sigma_k q_k^T$$

- and q_k is:

$$q_k = q^T U_k \Sigma_k^{-1}$$

- For retrieval, q_k is compared with other documents in V_k.

Exercise...

- For the below corpus use LSI to find best matching documents for the query q = *dies, dagger*.
- Corpus:
  - d_1: *Romeo and Juliet.*,
  - d_2: *Juliet: O happy dagger!*,
  - d_3: *Romeo died by dagger.*,
  - d_4: *'Live free or die', that's the New-Hampshire's motto.*,
  - d_5: *Did you know, New-Hampshire is in New-England.*

Exercise

- Procedure:
  - pre-process corpus (punctuation/stopwords removal, lemmatization, lowercase...)
  - define vocabulary $vocab$
  - create TF matrix: A for vocabulary $vocab$
  - decompose A using SVD: $A = U\Sigma V^T$
  - reduce dimensions of the SVD matrices to the size k: $A_k = U_k \Sigma_k V_k^T$
  - calculate query representation in k-concept space: $q_k = q^T U_k \Sigma_k^{-1}$
  - for each document d_i as represented in V^T, calculate cosine distance with q_k.

LSI vs keywords-based methods

- LSI performs better than traditional keywords-based methods.
- Drawbacks:
  - Time complexity of SVD ($O(m^2 n)$) too high for large corpuses.
  - Concept space not interpretable.
  - Finding optimal k for dimension reduction hard problem.

Web search

- Web search mainly based on the vector space model and term matching.
- Main steps behind web searching:
  - crawling
  - parsing
  - indexing
  - searching and ranking

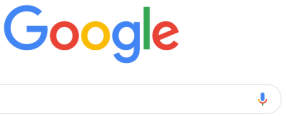
Ranking

- Ranking most important feature of search engines.
- Traditional approaches such as cosine similarity do not suffice.
- For Web, quality of pages also important factor:
  - quality of pages on the Web differs a lot…
  - hyperlinks can be exploited to measure how important is a specific page.
- Both, page content and reputation usually considered when ranking pages.

Content based ranking

- Query terms can occur in several places in a page. Different kind of info is considered:
  - Occurrence type: title, anchor text, URL, body
  - Count: for each type of occurrence
  - Position: for each type of occurrence – used for proximity evaluation in case of multiple query terms
- Weights are assigned to occurrence types and counts:
  - occurence type vector: $\overrightarrow{otv} = [tw_{title}, tw_{anchor}, tw_{url}, tw_{body}]$
  - count vector: $\overrightarrow{cv} = [cw_{title}, cw_{anchor}, cw_{url}, cw_{body}]$

Single word queries



- Procedure:
  - define occurrence type vector $\overrightarrow{otv}$
  - for each page p retrieved from the inverted index
    - calculate count vector $\overrightarrow{cv}$
    - calculate IR score: $\overrightarrow{cv} \circ \overrightarrow{otv}$
    - calculate Reputation score
    - calculate Page score = $f(IR\ score, Reputation\ score)$

Multiple word queries

- Additionally to consider:
  - term proximity
  - term ordering
- Procedure (ignoring term ordering):
  - define type proximity vector $\overrightarrow{tpv}$
  - for each page p retrieved from the inverted index
    - for each pair of query terms calculate proximity value pv
    - calculate count vector $\overrightarrow{cv}$ for each occurrence type and proximity value
    - calculate IR score: $\overrightarrow{cv} \circ \overrightarrow{tpv}$
    - calculate Reputation score
    - calculate Page score = $f(IR\ score, Reputation\ score)$

Evaluation of page reputation

- Page reputation computed based on the link structure of the page.
- Two algorithms very important: PageRank and HITS.
- Both introduced in 1998:
  - HITS presented by Jon Kleinberg,
  - PageRank presented by Sergey Brin and Larry Page. Based on the algorithm, they built the search engine Google.

Prestige

- PageRank and HITS based on prestige.
- Prestige types:
  - Degree prestige: an actor is prestigious if it receives many in-links: $P_D(i) = \frac{d_I(i)}{n-1}$, where $d_I(i)$ is the in-degree of i and n is the total number of actors in the network.
  - Proximity prestige: only actors that are directly or indirectly connected to i are considered (influence domain): $P_P(i) = \frac{|I_i|/(n-1)}{\sum_{j \in I_i} d(j,i)/|I_i|}$, where $|I_i|/(n-1)$ is the proportion of actors that can reach actor i.

Rank prestige

- Rank prestige: considers the prominence of actors that "vote" for the actor under observation.
  - Rank prestige $PR(i)$ is defined as a linear combination of links that point to i: $P_R(i) = \sum_j P_r(j)$, where page j points to page i.
  - Let $\boldsymbol{P}$ represent the vector with rank prestige values of all actors: $\boldsymbol{P} = (P_R(1), P_R(2),..., P_R(n))^T$ and A matrix where A_{ij}=1 if i points to j or 0 otherwise.
  - We then have $\boldsymbol{P} = \boldsymbol{A}^T\boldsymbol{P}$ where $\boldsymbol{P}$ is eigenvector of $\boldsymbol{A}^T$ and can be calculated using well known Power iteration method.
  - Note: Power iteration not directly useful for PageRank calculation, since A does not fulfil required preconditions.

PageRank

- The basic idea of PageRank follows the idea of rank prestige with the following difference:
  - Since a page may point to many other pages, its prestige score should be shared among all the pages that it points to. Thus
  - $P(i) = \sum_{(j,i)\in E} \frac{P(j)}{O_j}$, where Web is represented as a directed graph $G = (V, E)$ with V as a set of nodes and E as set of edges or hyperlinks and O_j is the number of out-links of j. Let n be the total number of pages, i.e. $n = |V|$.

Modeling Web with Markov chain...

- Markov chain models Web surfing as a stochastic process where the surfer randomly and uniformly choses where to go next.
- We can calculate the probability of a random surfer to be in state j after one step as follows: $p_1(j) = \sum_{i=1}^{n} A_{ij}(1)\, p_0(i)$, where $A_{ij}(1)$ is the probability of going from i to j in 1 transition, and $A_{ij}(1) = A_{ij}$. In matrix form: $\boldsymbol{p_1 = A^T p_0}$.
- The probability distribution after k steps is $\boldsymbol{p_k = A^T p_{k-1}}$.

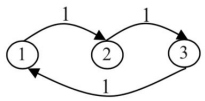
Ergodic Theorem of Markov chains

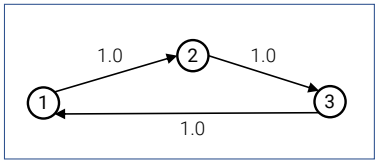
- By the Ergodic Theorem of Markov chains, a finite Markov chain defined by the stochastic transition matrix A has a unique stationary probability distribution if A is irreducible and aperiodic.
- Stationary probability distribution: after a series of transitions $\boldsymbol{p_k}$ will converge to a steady-state probability vector $\boldsymbol{\pi}$ regardless of the choice of the initial probability vector $\boldsymbol{p_0}$, i.e.: $\lim_{k\to\infty} \boldsymbol{p_k} = \boldsymbol{\pi}$
- When in steady state, $\boldsymbol{p_k = p_{k+1} = \pi}$ which means $\boldsymbol{\pi = A^T \pi}$ and $\boldsymbol{\pi}$ is the principal eigenvector of $\boldsymbol{A^T}$ with eigenvalue of 1.

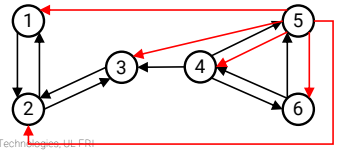
Stationary probability distribution

- Matrix A has to be:
  - Stochastic: a stochastic matrix is the transition matrix for a finite Markov chain whose entries in each row are non-negative real numbers and sum to 1.
  - Irreducible: irreducible means that the Web graph G is strongly connected.
  - Aperiodic: a state i is periodic with period $k > 1$ if k is the smallest number such that all paths leading from state i back to state i have a length that is a multiple of k. If a state is not periodic (i.e., k = 1), it is aperiodic. A Markov chain is aperiodic if all states are aperiodic.
- When modeling Web with Markov chain, neither of these requirements is satisfied.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

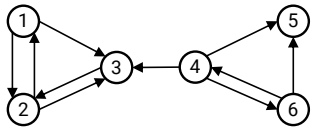






Stochastic matrix

- Example of non stochastic Web subgraph

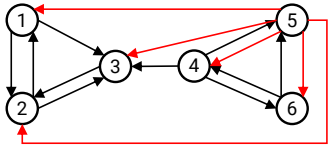


$$A = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix}$$

- Solution:
  - Remove dangling pages including links. Calculate PageRank and then include dangling pages back. Calculate their PageRank.
  - For each dangling page add out links to all pages in the Web.

Irreducible matrix

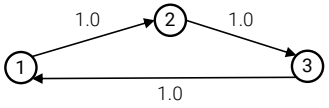
- If we make A stochastic, it can still remain irreducible. There is still no possibility to get from 3 to 4.



$$\overline{A} = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix}$$

Aperiodic matrix

- Aperiodic matrix has no periodic states. Web subgraph below is periodic. All states are periodic.



$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Making matrix irreducible and aperiodic

- Solution:
  - Add a link from each page to every page and give each link a small transition probability controlled by a parameter d.
- New rules for random surfer:
  - with probability d, randomly choose an out-link to follow.
  - with probability $1-d$, jump to a random page without a link.

$$\boldsymbol{P} = \left((1-d)\frac{\boldsymbol{E}}{n} + d\boldsymbol{A}^T\right)\boldsymbol{P}$$

  - Where $\boldsymbol{e}$ is vector of all 1 and $E = \boldsymbol{e}\boldsymbol{e}^T$ is a matrix of $n \times n$ of all 1.

PageRank formula

- If we scale P so that $\boldsymbol{e}^T\boldsymbol{P} = n$ we get:

$$\boldsymbol{P} = (1-d)\boldsymbol{e} + d\boldsymbol{A}^T\boldsymbol{P}$$

- Which for page i is:

$$P(i) = (1-d) + d\sum_{j=1}^{n} A_{ji}P\,(\mathrm{j})$$

$$P(i) = (1-d) + d\sum_{j=1}^{n} \frac{P(j)}{O_j}$$

PageRank algorithm

- Algorithm

```
PageRank-Iterate(G)
    P_0 ← e/n
    k ← 1
    repeat
        P_k ← (1−d)e + dA^T P_{k−1}
        k ← k + 1
    until ||P_k − P_{k−1}|| < ε
    return P_k
```


PageRank strengths and weaknesses

- Strengths
  - query independent
  - hard to cheat
- Weaknesses
  - authority in general and not in connection with the query
  - time not considered