Text Retrieval

Sergej Sizov

Information Retrieval

Summer term 2009

UNIVERSITÄT KOBLENZ · LANDAU

The document ranking problem

We have a collection of documents

User issues a query

A list of documents needs to be returned

Ranking method is core of an IR system:

- In what order do we present documents to the user?
- We want the "best" document to be first, second best second, etc....

Idea: Rank by probability of relevance of the document w.r.t. information need

- P(relevant|document)

Probabilistic IR: Principles

Goal:

Ranking based on sim(doc d, query q) =

P[R|d] = P [doc *d* is relevant for query *q* | *d* has term vector X1,..., Xm]

Assumptions:

Document relevance does not depend of other documents

Relevant and irrelevant documents differ in their terms.

Binary Independence Retrieval (BIR) Model:

- Probabilities for term occurrence are pairwise
- independent for different terms.
- Term weights are binary: Xi $\in$ {0,1}.

For terms that do not occur in query *q* the probabilities for such a term occurring are the same for relevant and irrelevant documents.

Probabilistic IR with Term Independence

Ranking Proportional to Relevance Odds

$$sim(d,q) = O(R \mid d) = \frac{P[R \mid d]}{P[\neg R \mid d]}$$

odds for relevance
(ratio of relevant documents)

$$= \frac{P[d \mid R] \times P[R]}{P[d \mid \neg R] \times P[\neg R]}$$

Bayes‘ theorem

$$\sim \frac{P[d \mid R]}{P[d \mid \neg R]} = \prod_i \frac{P[X_i \mid R]}{P[X_i \mid \neg R]}$$

independence or
linked dependence

$$sim(d,q)' = \log \prod_{i \in q} \frac{P[Xi \mid R]}{P[Xi \mid \neg R]}$$

Xi = 1 if d includes
i-th term, 0 otherwise

$$= \sum_{i \in q} \log P[Xi \mid R] - \log P[Xi \mid \neg R]$$

Probabilistic Retrieval

$$sim(d,q)' = \log \prod_{i \in q} \frac{P[Xi \mid R]}{P[Xi \mid \neg R]} = \sum_{i \in q} \log P[Xi \mid R] - \log P[Xi \mid \neg R]$$

$$= \sum_{i \in q} log\,(pi^{Xi} (1 - pi)^{1-Xi}) - log\,(qi^{Xi} (1 - qi)^{1-Xi})$$

with estimators pi=P[Xi=1|R] and qi=P[Xi=1|¬R] (binary features)

$$= \sum_{i \in q} log\,(\frac{pi^{Xi}(1 - pi)}{(1 - pi)^{Xi}}) - log\,(\frac{qi^{Xi}(1 - qi)}{(1 - qi)^{Xi}})$$

$$= \sum_{i \in q} Xi \log \frac{pi}{1 - pi} + \sum_{i \in q} Xi \log \frac{1 - qi}{qi} + \sum_{i \in q} \log \frac{1 - pi}{1 - qi}$$

$$\sim \sum_{i \in q} Xi \log \frac{pi}{1 - pi} + \sum_{i \in q} Xi \log \frac{1 - qi}{qi} = sim(d,q)''$$

Probabilistic Retrieval

Robertson / Sparck Jones Formula

Estimate pi und qi based on training sample
(query q on small sample of corpus) or based on
intellectual assessment of first round's result (***relevance feedback***):

Let N be #docs in sample,
R be # relevant docs in sample
ni #docs in sample that contain term i,
ri # relevant docs in sample that contain term i.

$\Rightarrow$ Estimate: $pi = \frac{ri}{R}$ $qi = \frac{ni - ri}{N - R}$

or: $pi = \frac{ri + 0.5}{R + 1}$ $qi = \frac{ni - ri + 0.5}{N - R + 1}$ **(Lidstone smoothing with λ=0.5)**

$\Rightarrow sim(d,q)'' = \sum_i Xi \log \frac{ri + 0.5}{R - ri + 0.5} + \sum_i Xi \log \frac{N - ni - R + ri + 0.5}{ni - ri + 0.5}$

$\Rightarrow$ Weight of term i in doc d: $\log \frac{(ri + 0.5)\ (N - ni - R + ri + 0.5)}{(R - ri + 0.5)(ni - ri + 0.5)}$

Probabilistic Retrieval: tf*idf Formula

Assumptions (without training sample or relevance feedback):
- pi is the same for all i.
- Most documents are irrelevant.
- Each individual term i is infrequent.

This implies:

- $\sum_i Xi \log \frac{pi}{1-pi} = c \sum_i Xi$ with constant c
- $qi = P[Xi = 1 | \neg R] \approx \frac{df_i}{N}$
- $\frac{1-qi}{qi} = \frac{N-df_i}{df_i} \approx \frac{N}{df_i}$

$$\Rightarrow \quad sim(d,q)'' = \sum_i Xi \log \frac{pi}{1-pi} + \sum_i Xi \log \frac{1-qi}{qi}$$
$$\approx c \sum_i Xi + \sum_i Xi\, idf_i$$

Scalar product over the product of tf and dampend idf values for query terms

Probabilistic Retrieval: Example

Documents with relevance feedback:

q: t1 t2 t3 t4 t5 t6

	t1	t2	t3	t4	t5	t6	R
d1	1	0	1	1	0	0	1
d2	1	1	0	1	1	0	1
d3	0	0	0	1	1	0	0
d4	0	0	1	0	0	0	0
ni	2	1	2	3	2	0	
ri	2	1	1	2	1	0	
pi	5/6	1/2	1/2	5/6	1/2	1/6	
qi	1/6	1/6	1/2	1/2	1/2	1/6	

R=2, N=4

Score of new document d5 (with Lidstone smoothing with λ=0.5):

d5∩q: <1 1 0 0 0 1> $\rightarrow$ sim(d5, q) = log 5 + log 1 + log 0.2 + log 5 + log 5 + log 5

$$sim(d,q)'' = \sum_i Xi \log \frac{pi}{1-pi} + \sum_i Xi \log \frac{1-qi}{qi}$$

BIR: Summery

- The most important classic prob. IR model
- Use only term presence/absence, thus also referred to as Binary Independence Model
- Most natural for relevance/pseudo feedback
- When without relevance judgments, the model parameters must be estimated in an ad hoc way
- Performance isn't as good as tuned VS model
- Extensions
  - Term correlations
  - Okapi
  - Smoothing

Extensions of Probabilistic IR

Consider term correlations in documents (with binary Xi)

→ Problem of estimating m-dimensional prob.distribution

P[X1=... ∧ X2= ... ∧ ... ∧ Xm=...] =: f_X(X1, ..., Xm)

One possible approach: **Tree Dependence Model**:

a) Consider only 2-dimensional probabilities (for term pairs)

$f_{ij}(Xi, Xj)=P[Xi=..\wedge Xj=..]= \sum_{X_1}.. \sum_{X_{i-1}} \sum_{X_{i+1}} .. \sum_{X_{j-1}} \sum_{X_{j+1}} .. \sum_{X_m} P[X_1 = ... \wedge .. \wedge X_m = ...]$

b) For each term pair

estimate the error between independence and the actual correlation

c) Construct a tree with terms as nodes and the

m-1 highest error (or correlation) values as weighted edges

Considering 2-dimensional Term Correlation

Variant 1:

Error of approximating f by g (**Kullback-Leibler divergence**) with g assuming pairwise term independence:

$$\varepsilon(f,g) := \sum_{\vec{X}\in\{0,1\}^m} f(\vec{X}) \log \frac{f(\vec{X})}{g(\vec{X})} = \sum_{\vec{X}\in\{0,1\}^m} f(\vec{X}) \log \frac{f(\vec{X})}{\prod_{i=1}^{m} g_i(X_i)}$$

Variant 2:

Correlation coefficient for term pairs:

$$\rho(Xi, Xj) := \frac{Cov(Xi, Xj)}{\sqrt{Var(Xi)}\sqrt{Var(Xj)}}$$

Variant 3:

level-α values or p-values of **Chi-square independence test**

Constructing the Term Dependence Tree

Given:

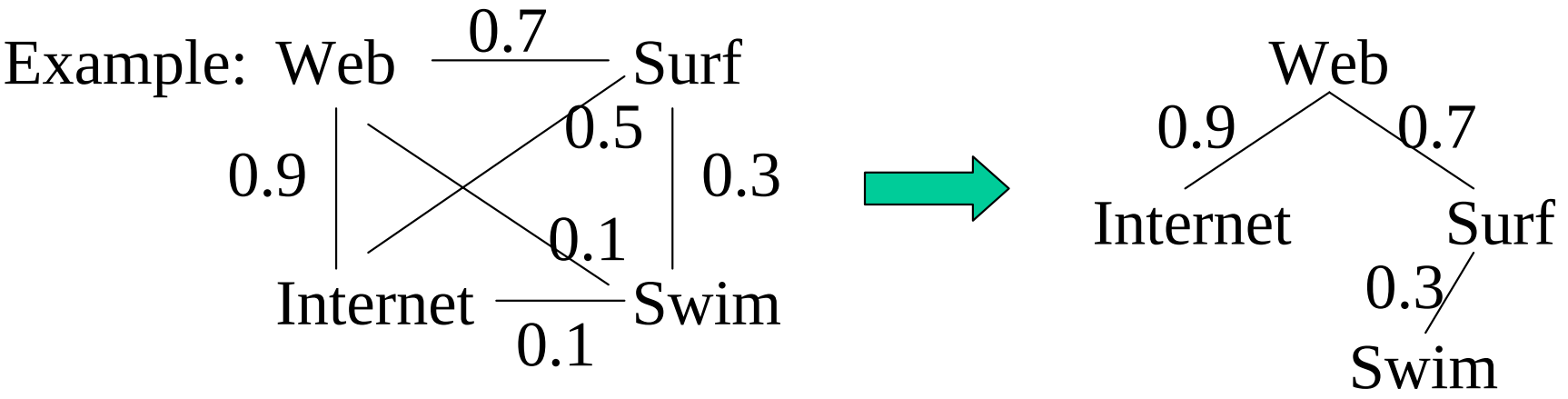
complete graph (V, E) with m nodes Xi $\in$V and
m^2 undirected edges $\in$ E with weights ε (or ρ)

Wanted:

spanning tree (V, E‘) with maximal sum of weights

Algorithm:

Sort the m^2 edges of E in descending order of weight
E‘ := $\varnothing$
Repeat until |E‘| = m-1
E‘ := E‘ $\cup$ {(i,j) $\in$E | (i,j) has max. weight in E}
provided that E‘ remains acyclic;
E := E – {(i,j) $\in$E | (i,j) has max. weight in E}



Estimation of Multidimensional Probabilities

with Term Dependence Tree

Given is a term dependence tree (V = {X1, ..., Xm}, E‘).
Let X1 be the root, nodes are preorder-numbered, and assume that
Xi and Xj are independent for (i,j) $\notin$ E‘. Then:

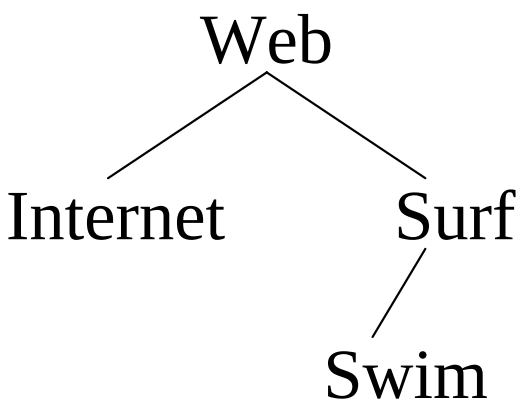
$$P[X1 = .. \wedge .. \wedge Xm = ..] = P[X1 = ..]\, P[X2 = .. \wedge Xm = .. \mid X1 = ..]$$

$$= \Pi_{i=1..m} P[Xi = .. \mid X1 = .. \wedge X(i-1) = ..]$$

$$= P[X1] \cdot \prod_{(i,j) \in E'} P[Xj \mid Xi]$$

$$= P[X1] \cdot \prod_{(i,j) \in E'} \frac{P[Xi, Xj]}{P[Xi]}$$

Example:



$$P[Web, Internet, Surf, Swim] = P[Web] \frac{P[Web, Internet]}{P[Web]} \frac{P[Web, Surf]}{P[Web]} \frac{P[Surf, Swim]}{P[Surf]}$$

Latent Semantic Analysis

- Lexical matching at term level inaccurate (claimed)
- Polysemy – words with number of ‘meanings’ – term matching returns irrelevant documents – impacts precision
- Synonomy – number of words with same ‘meaning’ – term matching misses relevant documents – impacts recall
- Fewer dimensions → dimension reduction
- Keep k strongest dimensions: remove noise

LSA assumes that there exists a LATENT structure in word usage – obscured by variability in word choice

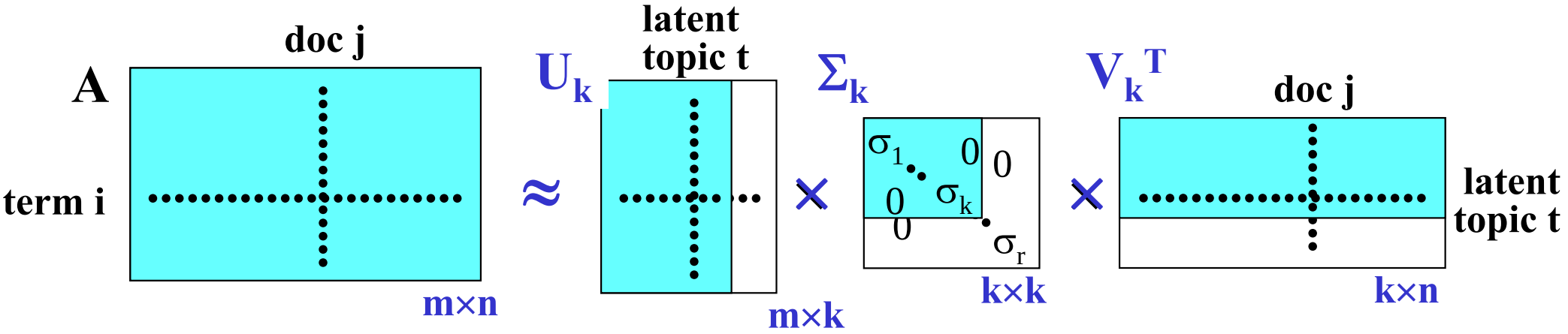
Word usage defined by term and document co-occurrence – matrix structure

Latent Semantic Indexing (LSI)

[Deerwester et al. 1990]

A is the m×n term-document matrix. Then:

- U and U_k are the m×r and m×k term-topic similarity matrices,
- V and V_k are the n×r and n×k document-topic similarity matrices,
- $A \times A^T$ and $A_k \times A_k^T$ are the term-term similarity matrices,
- $A^T \times A$ and $A_k^T \times A_k$ are the document-document similarity matrices



mapping of m×1 vectors into latent-topic space: $d_j :\to U_k^T \times d_j =: d_j'$

$q :\to U_k^T \times q =: q'$

scalar-product similarity in latent-topic space: $d_j'^T \times q' = ((\Delta_k V_k^T)_{*j})^T \times q'$

LSI: Indexing and Query Processing

- The matrix $\Delta_k V_k^T$ corresponds to a **„topic index“** and is stored in a suitable data structure.
  Instead of $\Delta_k V_k^T$ the simpler **index V_k^T** could be used.

- Additionally the **term-topic mapping U_k** must be stored.

- A **query q** (an m×1 column vector) in the term vector space is transformed into query **q‘= $U_k^T \times q$** (a k×1 column vector) and evaluated in the topic vector space (i.e. V_k)
  (e.g. by scalar-product similarity $V_k^T \times q‘$ or cosine similarity)

- A **new document d** (an m×1 column vector) is transformed into **d‘ = $U_k^T \times d$** (a k ×1 column vector) and appended to the „index“ V_k^T as an additional column **(„folding-in“)**

LSI: Example

m=6 terms
- t1: bak(e,ing)
- t2: recipe(s)
- t3: bread
- t4: cake
- t5: pastr(y,ies)
- t6: pie

n=5 documents
- d1: How to bake bread without bread recipes
- d2: The classic art of Viennese Pastry
- d3: Numerical recipes: the art of scientific computing
- d4: Breads, pastries, pies and cakes: quantity baking recipes
- d5: Pastry: a book of best French recipes

$$A = \begin{pmatrix} 0.4446 & 0.0000 & 0.0000 & 0.3422 & 0.0000 \\ 0.1083 & 0.0000 & 1.0000 & 0.0833 & 0.4002 \\ 0.8892 & 0.0000 & 0.0000 & 0.3422 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.6010 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.1908 & 0.9164 \\ 0.0000 & 0.0000 & 0.0000 & 0.6010 & 0.0000 \end{pmatrix}$$

LSI: Example

$$A = \begin{pmatrix} -0.1337 & 0.4385 & -0.0916 & -0.0858 \\ -0.4039 & 0.0798 & 0.9089 & 0.06680 \\ -0.1909 & 0.7045 & -0.1092 & -0.5105 \\ -0.1336 & 0.3000 & -0.1298 & 0.5997 \\ -0.8642 & -0.3535 & -0.3464 & -0.0906 \\ -0.1336 & 0.3000 & -0.1298 & 0.5997 \end{pmatrix} \quad \mathrm{U}$$

$$\times \begin{pmatrix} 1.4543 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.1764 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.9980 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.7115 \end{pmatrix} \quad \Delta$$

$$\times \begin{pmatrix} -0.1878 & -0.5942 & -0.2777 & -0.3233 & -0.65571 \\ 0.7061 & -0.3005 & 0.0678 & 0.5873 & -0.2482 \\ -0.0396 & -0.3471 & 0.9107 & -0.2155 & 0.0464 \\ -0.6817 & -0.1274 & 0.0940 & 0.7100 & -0.0792 \end{pmatrix} \quad \mathrm{V}^{\mathrm{T}}$$

LSI: Example

$$A_2 = \begin{pmatrix} 0.4008 & -0.0395 & 0.0890 & 0.3659 & -0.0006 \\ 0.1766 & 0.3209 & 0.1695 & 0.2451 & 0.3619 \\ 0.6373 & -0.0841 & 0.1333 & 0.5766 & -0.0237 \\ 0.2857 & 0.0094 & 0.0779 & 0.2701 & 0.0398 \\ -0.0576 & 0.8718 & 0.3209 & 0.1621 & 0.9273 \\ 0.2857 & 0.0094 & 0.0779 & 0.2701 & 0.0398 \end{pmatrix} = U_2 \times \Delta_2 \times V_2^T$$

LSI: Example

query q: baking bread
$q = (1\ 0\ 1\ 0\ 0\ 0)^T$

transformation into topic space with k=2
$q' = U_k^T \times q = (-0.3246\ 1.1430)^T$

scalar product similarity in topic space with k=2:
$sim(q, d1) = Vk_{*1}^T \times q' \approx 0.87$ $sim(q, d2) = Vk_{*2}^T \times q \approx -0.15$
$sim(q, d3) = Vk_{*3}^T \times q' \approx 0.17$ etc.

Folding-in of a new document d6:
algorithmic recipes for the computation of pie
$d6 = (0\ \ 0.7071\ \ 0\ \ 0\ \ 0\ \ 0.7071)^T$

transformation into topic space with k=2
$d6' = U_k^T \times d6 \approx (-0.3801\ 0.2686)$

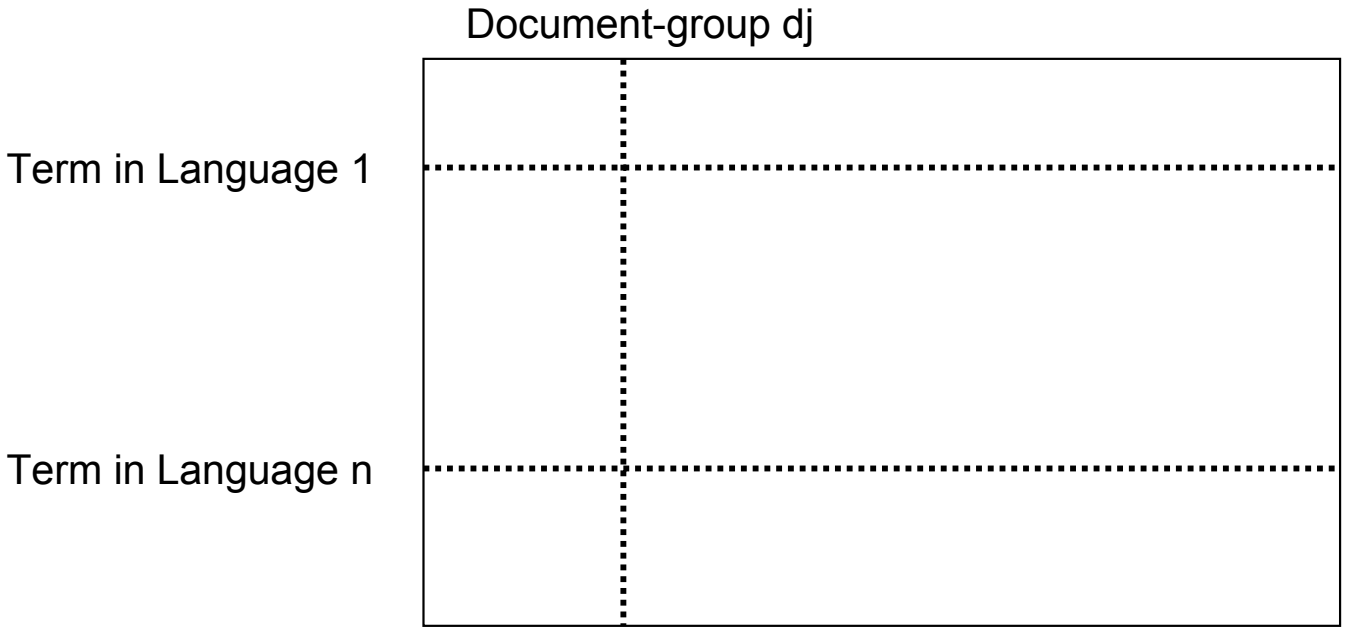
d6‘ is appended to Vk^T as a new column

Cross-Language LSI

- Build common feature-articles space
  - Don't differ between languages
  - One Document consists of several languages
- Decompose document-term-matrix by reduced SVD to build concept-document-space
- Map new documents to concept-document-space
- Mate retrieval – looking for corresponding document in a second language

Cross-Language LSI

- Construct LSI model (U_k, Δ_k, V_k^T) from training documents that are available in multiple languages:
  - Consider all language variants of the same document as a single document and
  - Extract all terms or words for all languages



- SVD with regression brings terms in different language together
- Folding in documents in one language

Datasets

Wikipedia

- Online encyclopedia
- Using a subset of 160 000 articles which are present in English, German and French connected by language links

Multext Corpus

- The Multext Corpus is an EU-Textcorpus
- Question and Answers of the European Commission in English, German, French, Spanish and Italian
- About 2780 Pairs of Question and Answer

JRC-Arquis

- EU-Textcorpus
- Documents from beginning of the EU until today

Test on Multext

Multext - Multext

60 % for training ca. 1670 documents

40 % for testing ca. 1113 documents

2 runs with different training and test sets on

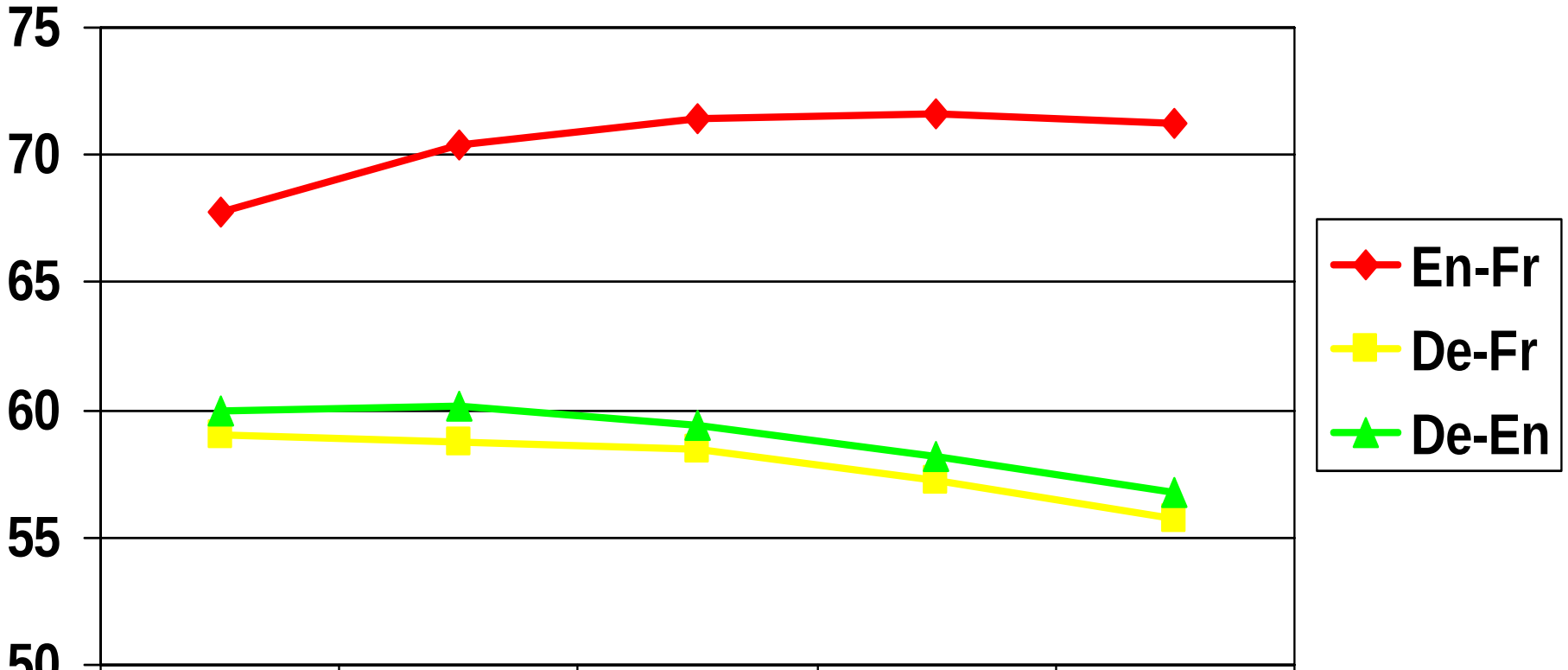
Question and Answer (as one document)

Wikipedia – Multext

Training on Wikipedia (ca. 50 000 documents)

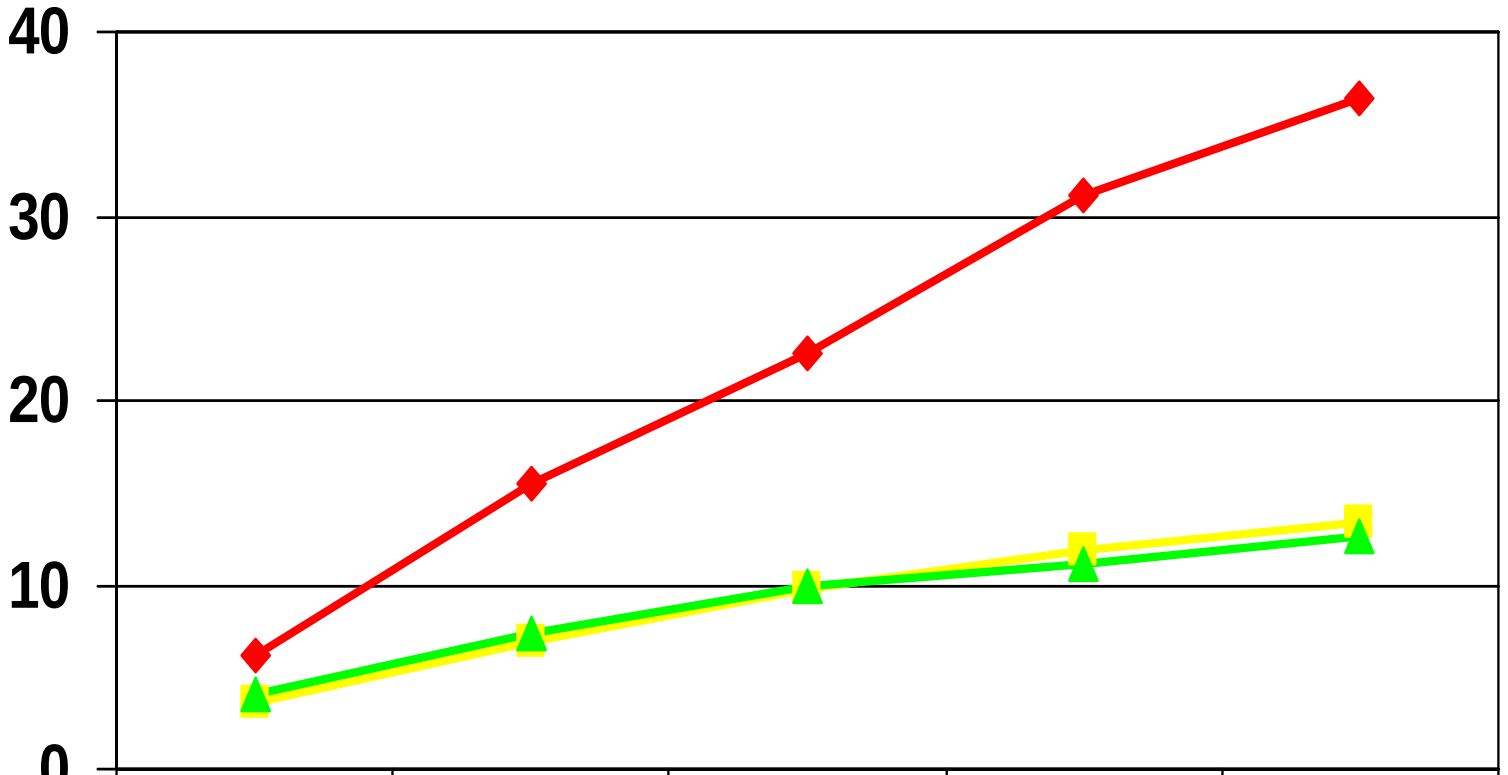
Mate Retrieval on Multext

MeanMate-Multex Top1



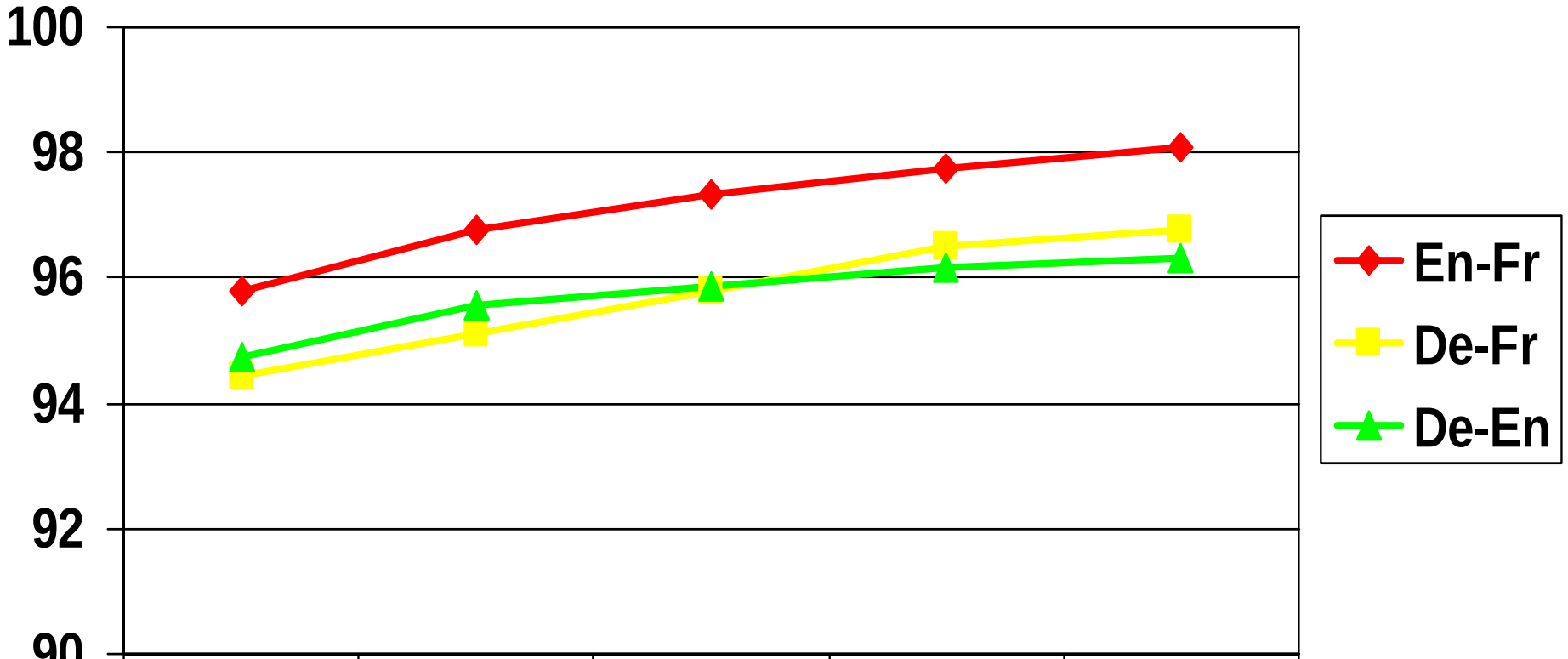
	100	200	300	400	500
En-Fr	67,7448	70,3504	71,42855	71,60825	71,20395
De-Fr	58,9847	58,7601	58,4906	57,27765	55,7502
De-En	59,92815	60,10785	59,434	58,1312	56,73855

MeanMate Multex-Wiki Top1



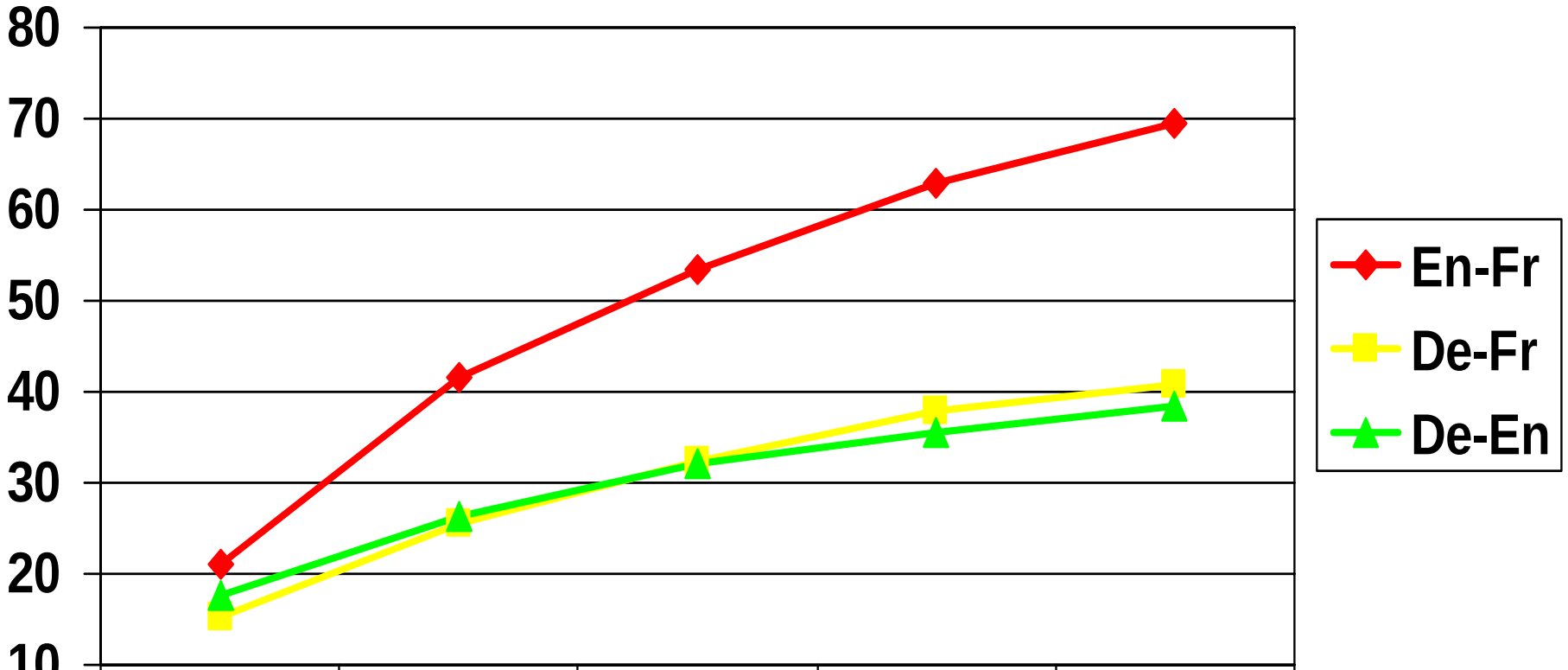
	100	200	300	400	500
En-Fr	6,14445	15,55875	22,5656	31,09955	36,3457
De-Fr	3,57525	6,88105	9,71975	11,82175	13,42075
De-En	4,00645	7,36615	9,9353	11,193	12,7021

MeanMate-Multex Top10



	100	200	300	400	400
En-Fr	95,7772	96,7655	97,3495	97,75385	98,0683
De-Fr	94,42945	95,1033	95,77715	96,496	96,7655
De-En	94,74395	95,5526	95,86705	96,1815	96,31625

MeanMate Multex-Wiki Top10



	100	200	300	400	500
En-Fr	20,98455	41,6277	53,48545	62,8279	69,5652
De-Fr	15,18145	25,61985	32,35715	37,85485	40,8372
De-En	17,69675	26,24865	32,03375	35,5192	38,50165

LSI: Summary

+ Elegant, mathematically well-founded model
+ „Automatic learning“ of term correlations (incl. morphological variants, multilingual corpus)
+ Implicit thesaurus (by correlations between synonyms)
+ Implicit discrimination of different meanings of polysems (by different term correlations)
+ Improved precision and recall on „closed“ corpora (e.g. TREC benchmark, financial news, patent databases,etc.) with empirically best k in the order of 100-200
- In general difficult choice of appropriate k
- Computational and storage overhead for very large (sparse) matrices
- No convincing results for Web search engines (yet)

Probabilistic Latent Semantic Analysis

Identification of Latent Classes (Aspects) using an Expectation Maximization (EM) Algorithm

Shown to solve

- Polysemy
  - Java could mean “coffee” and also the “PL Java”
  - Cricket is a “game” and also an “insect”
- Synonymy
  - “computer”, “pc”, “desktop” all could mean the same

Has a better statistical foundation than LSA

What is a Statistical Language Model?

generative model for word sequence
(generates probability distribution of word sequences,
or bag-of-words, or set-of-words, or structured doc, or ...)

Example: P[„Today is Tuesday“] = 0.001
P[„Today Wednesday is“] = 0.000000001
P[„The Eigenvalue is positive“] = 0.000001

LM itself highly context- / application-dependent

Examples:

- **speech recognition**: given that we heard „Julia“ and „feels“, how likely will we next hear „happy“ or „habit“?
- **text classification**: given that we saw „soccer“ 3 times and „game“ 2 times, how likely is the news about sports?
- **information retrieval**: given that the user is interested in math, how likely would the user use „distribution“ in a query?

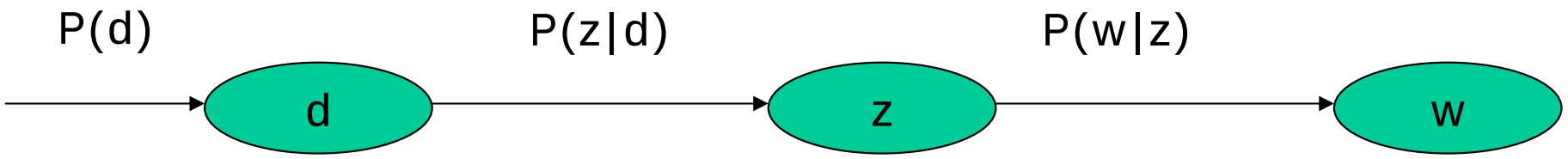
Aspect Model

Latent Variable model for general co-occurrence data

- Associate each observation (w,d) with a class variable z ∈ Z{z_1,…,z_K}

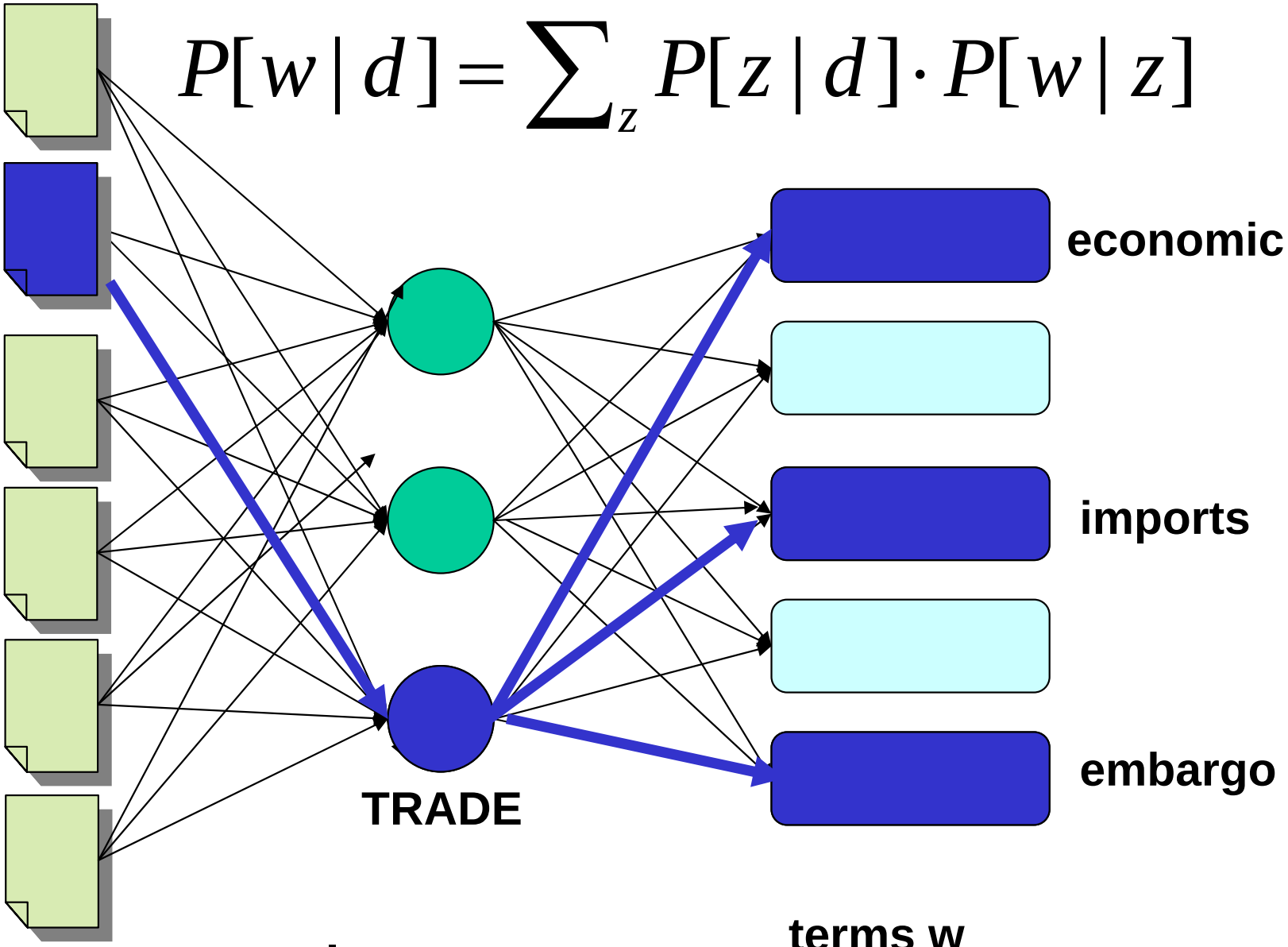
Generative Model

- Select a doc with probability P(d)
- Pick a latent class z with probability P(z|d)
- Generate a word w with probability P(w|z)



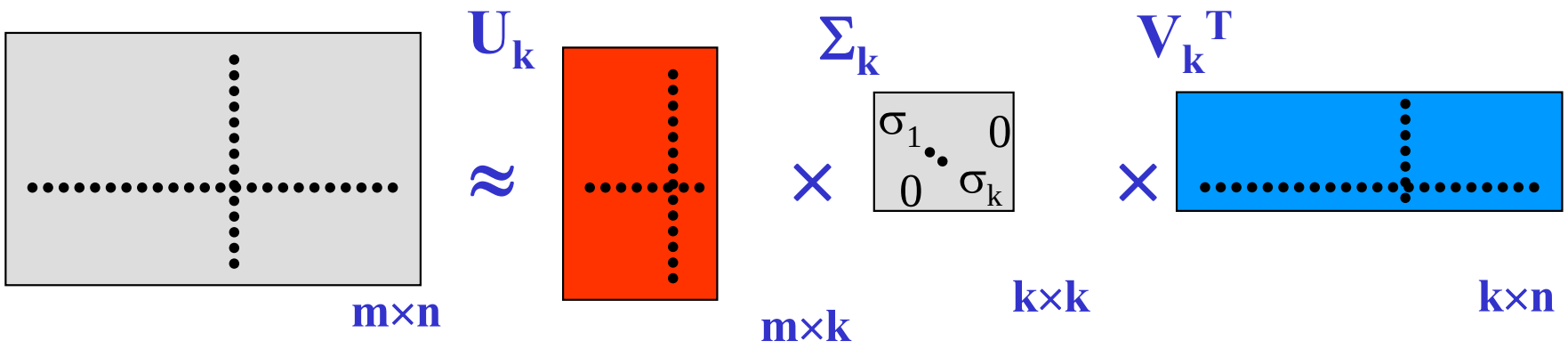
$$P[w \mid d] = \sum_z P[z \mid d] \cdot P[w \mid z]$$

d and w conditionally independent given z



Relationship of pLSI to LSI

$$P[d,w] = \sum_z P[d|z] \cdot P[z] \cdot P[w|z]$$



doc probs per concept

concept probs

term probs per concept

Key difference to LSI:
- **non-negative matrix decomposition**
- **with L1 normalization**

Key difference to LMs:
- **no generative model for docs**
- **tied to given corpus**

Aspect Model

Aspect Model

- Document is a mixture of underlying (latent) K aspects
- Each aspect is represented by a distribution of words P(w|z)

Model fitting with Tempered EM

The joint distribution *P(d,w)* is the sum over all aspects *z*

$$P(d,w) = \sum_z P(z)P(d,w \mid z)$$

Assuming conditional independence of *d* and *w* given *z*.

$$P(d,w) = \sum_z P(z)P(d \mid z)P(w \mid z)$$

Expectation-Maximization Method (EM)

Key idea:

when $L(\theta, X_1, ..., X_n)$ (where the X_i and θ are possibly multivariate) is analytically intractable then

- introduce ***latent (hidden, invisible, missing) random variable(s) Z*** such that
  - the ***joint distribution $J(X_1, ..., X_n, Z, \theta)$ of the „complete" data*** is tractable (often with Z actually being $Z_1, ..., Z_n$)
- derive the incomplete-data likelihood $L(\theta, X_1, ..., X_n)$ by ***integrating (marginalization) J:***

$$\hat{\theta} = arg\,max_\theta \sum_z J[\theta, X_1, ..., X_n, Z \mid Z = z]P[Z = z]$$

EM Procedure

Initialization: choose start estimate for $\theta^{(0)}$

Iterate (t=0, 1, …) until convergence:

E step (expectation):

estimate posterior probability of Z: $P[Z \mid X_1, \ldots, X_n, \theta^{(t)}]$
assuming θ were known and equal to previous estimate $\theta^{(t)}$,
and compute $E_{Z \mid X1, \ldots, Xn, \theta(t)}[\log J(X_1, \ldots, X_n, Z \mid \theta)]$
by integrating over values for Z

M step (maximization, MLE step):

Estimate $\theta^{(t+1)}$ by maximizing
$E_{Z \mid X1, \ldots, Xn, \theta(t)}[\log J(X_1, \ldots, X_n, Z \mid \theta)]$

convergence is guaranteed
(because the E step computes a lower bound of the true L function,
and the M step yields monotonically non-decreasing likelihood),
but may result in local maximum of log-likelihood function

EM at Indexing Time

(pLSI Model Fitting)

observed data: n(d,w) – absolute frequency of word w in doc d

model params: P[z|d], P[w|z] for concepts z, words w, docs d

maximize **log-likelihood** $\sum_d \sum_w n(d,w) \cdot \log P[dw]$

E step: posterior probability of latent variables

$$P[z|d,w] = \frac{P[z|d]P[w|z]}{\sum_y P[y|d]P[w|y]}$$

prob. that occurrence of word w in doc d can be explained by concept z

M step: MLE with completed data

$$P[w|z] \sim \sum_d n(d,w)P[z|d,w] \qquad P[z|d] \sim \sum_w n(d,w)P[z|d,w]$$

freq. of w associated with z

freq. of d associated with z

actual procedure „perturbs“ EM for „smoothing“ (avoidance of overfitting) → tempered annealing

EM Details (PLSI Model Fitting)

$$P[z \mid d,w] = \frac{P[z \mid d]P[w \mid z]}{\sum_y P[y \mid d]P[w \mid y]} \qquad \textbf{(E)}$$

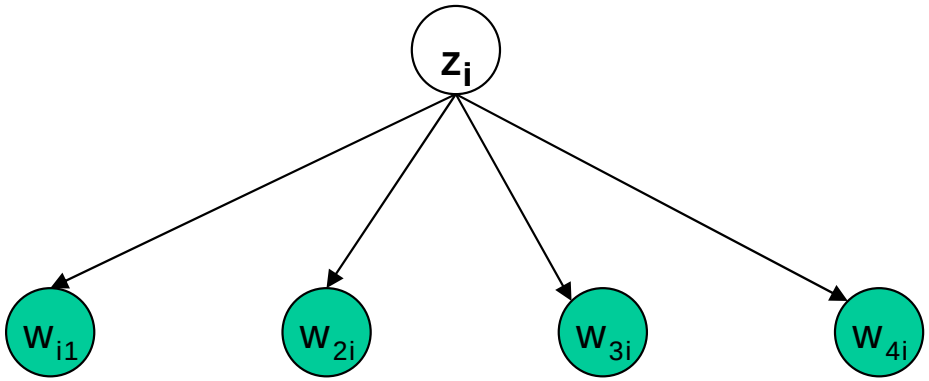
$$P[w \mid z] = \frac{\sum_d n(d,w)P[z \mid d,w]}{\sum_{d,u} n(d,u)P[z \mid d,u]} \qquad \textbf{(M1)}$$

$$P[z \mid d] = \frac{\sum_w n(d,w)P[z \mid d,w]}{\sum_{w,y} n(d,w)P[y \mid d,w]} \qquad \textbf{(M2)}$$

or equivalently compute P[z], P[d|z], P[w|z] in M step (see S. Chakrabarti, pp. 110/111)

Generalization: Language Models

Mixture of Unigrams



Mixture of Unigrams Model (this is just Naïve Bayes)

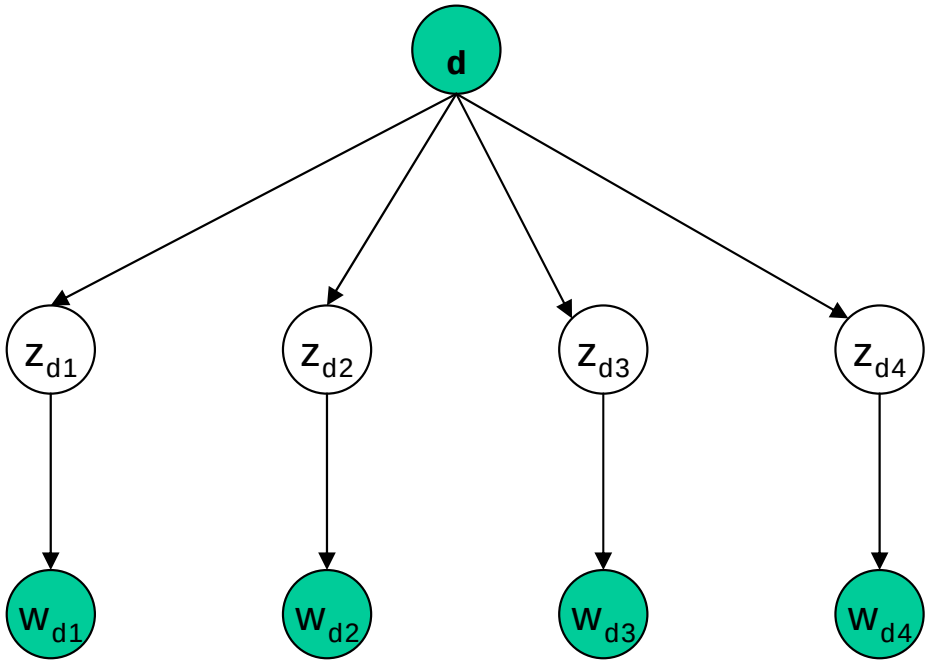
For each of M documents,

Choose a topic z.

Choose N words by drawing each one independently from a multinomial conditioned on z.

In the Mixture of Unigrams model, we can only have one topic per document!

The pLSI Model



Probabilistic Latent Semantic Indexing (pLSI) Model

For each word of document d in the training set,

Choose a topic z according to a multinomial conditioned on the index d.

Generate the word by drawing from a multinomial conditioned on z.

In pLSI, documents can have multiple topics.

Motivations for LDA

In pLSI, the observed variable d is an index into some training set. There is no natural way for the model to handle previously unseen documents.

The number of parameters for pLSI grows linearly with M (the number of documents in the training set).

We would like to be Bayesian about our topic mixture proportions.

Dirichlet Distributions

In the LDA model, we would like to say that the *topic mixture proportions* for each document are drawn from some distribution.

So, we want to put a distribution on multinomials. That is, k-tuples of non-negative numbers that sum to one.

The space is of all of these multinomials has a nice geometric interpretation as a (k-1)-*simplex*, which is just a generalization of a triangle to (k-1) dimensions.

Criteria for selecting our prior:

- It needs to be defined for a (k-1)-simplex.
- Algebraically speaking, we would like it to play nice with the multinomial distribution.

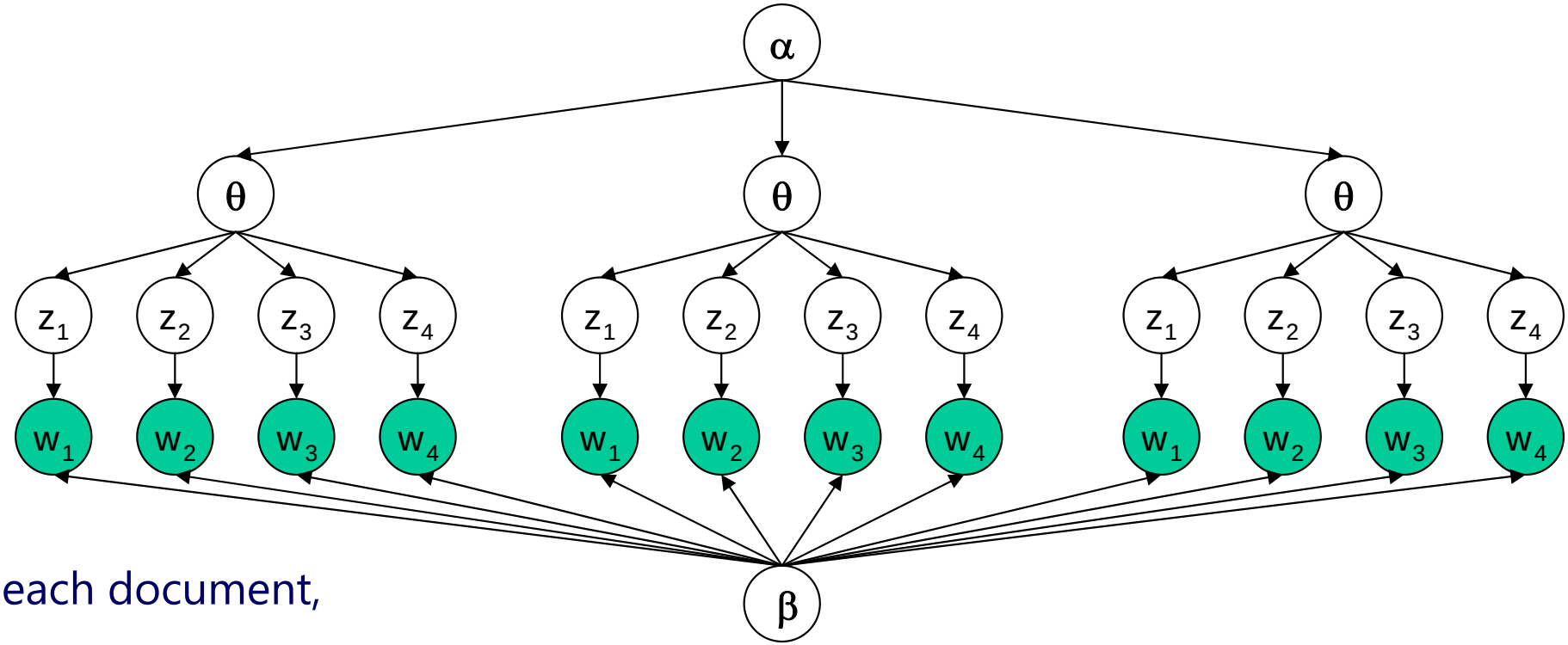
Dirichlet Distributions

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$$

Useful Facts:

- This distribution is defined over a (k-1)-simplex. That is, it takes k non-negative arguments which sum to one. Consequently it is a natural distribution to use over multinomial distributions.
- In fact, the Dirichlet distribution is the conjugate prior to the multinomial distribution. (This means that if our likelihood is multinomial with a Dirichlet prior, then the posterior is also Dirichlet!)
- The Dirichlet parameter α_i can be thought of as a prior count of the i^{th} class.

The LDA Model



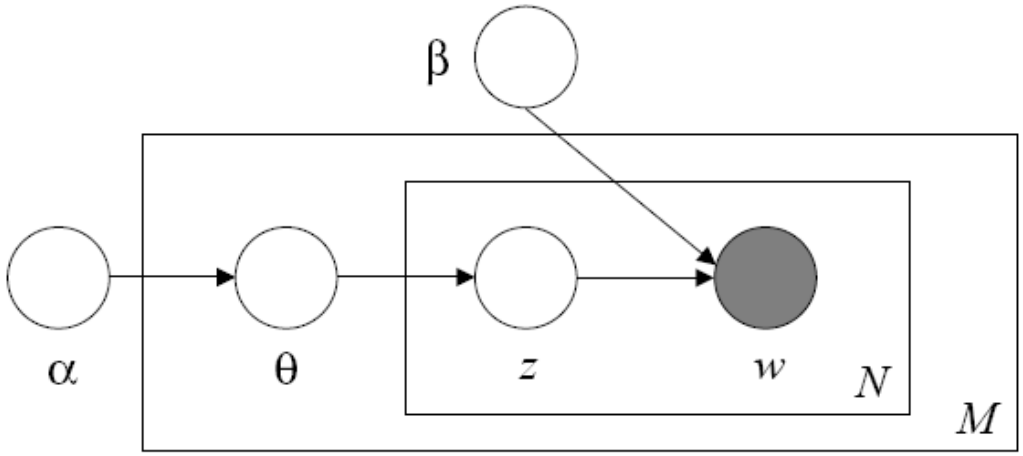
For each document,

Choose $\theta \sim \text{Dirichlet}(\alpha)$

For each of the N words wn:

- Choose a topic $z_n \sim$ Multinomial(θ)
- Choose a word w_n from $p(w_n|z_n,\beta)$, a multinomial probability conditioned on the topic z_n.

The LDA Model



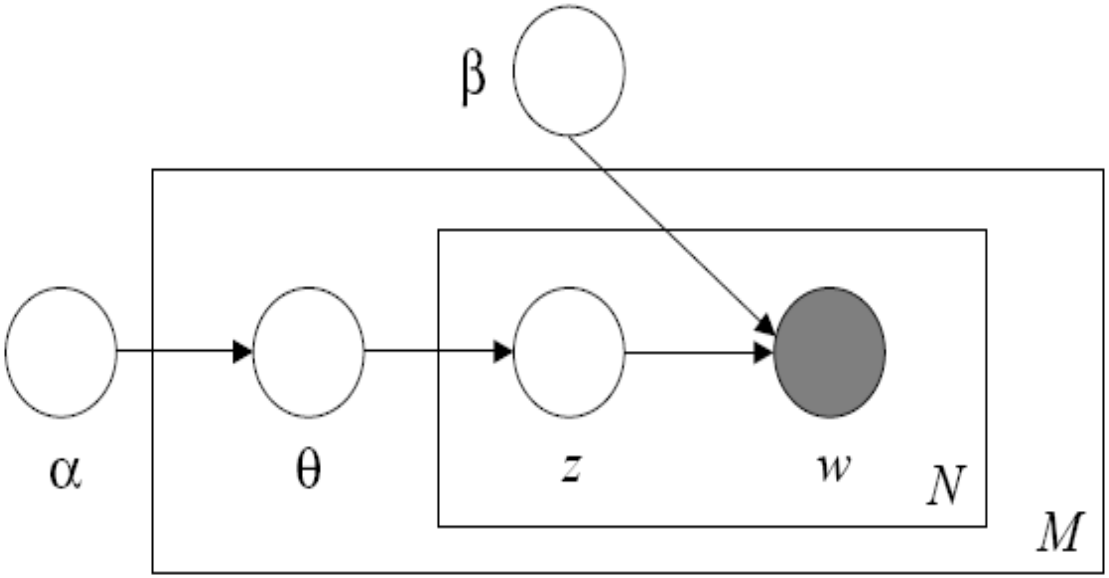
For each document,

Choose $\theta \sim \text{Dirichlet}(\alpha)$

For each of the N words w_n:

- Choose a topic $z_n \sim \text{Multinomial}(\theta)$
- Choose a word w_n from $p(w_n|z_n,\beta)$, a multinomial probability conditioned on the topic z_n.

Inference



The inference problem in LDA is to compute the posterior of the hidden variables given a document and corpus parameters α and β. That is, compute $p(\theta,z|w,\alpha,\beta)$.

Unfortunately, exact inference is intractable, so we turn to alternatives…

Some Results

Given a topic, LDA can return the most probable words.

For the following results, LDA was trained on 10,000 text articles posted to 20 online newsgroups with 40 iterations of EM. The number of topics was set to 50.

"politics"	"sports"	"space"	"computers"	"christianity"
Political	Team	Space	Drive	God
Party	Game	NASA	Windows	Jesus
Business	Play	Research	Card	His
Convention	Year	Center	DOS	Bible
Institute	Games	Earth	SCSI	Christian
Committee	Win	Health	Disk	Christ
States	Hockey	Medical	System	Him
Rights	Season	Gov	Memory	Christians

Q: How can we exploit the topical model (pLSI, LDA) for ranked retrieval?!

Quick solution: roll out the vector space model with cosine similarity..

Better: exploit (dis)similarity measures known from Information Theory (how?!)

Top-K query processing for text documents

Simplified Query Processor

1. Remove stopwords from query and map all words onto terms (word stems)
2. Convert query *q* into normal norm

   [(t_11 AND t_12 AND … AND t_1k$_1$)
   OR …
   (t_r1 AND t_r2 AND … AND t_rk$_r$)]
   AND NOT t_r+1 AND NOT t_r+2 …

3. For i=1 to r do

   find DocId-Lists for ti_1 bis ti_ki and

   compute union V_i of these DocIds

   set *rank(d)* of d in V_i auf sim(d, qi)

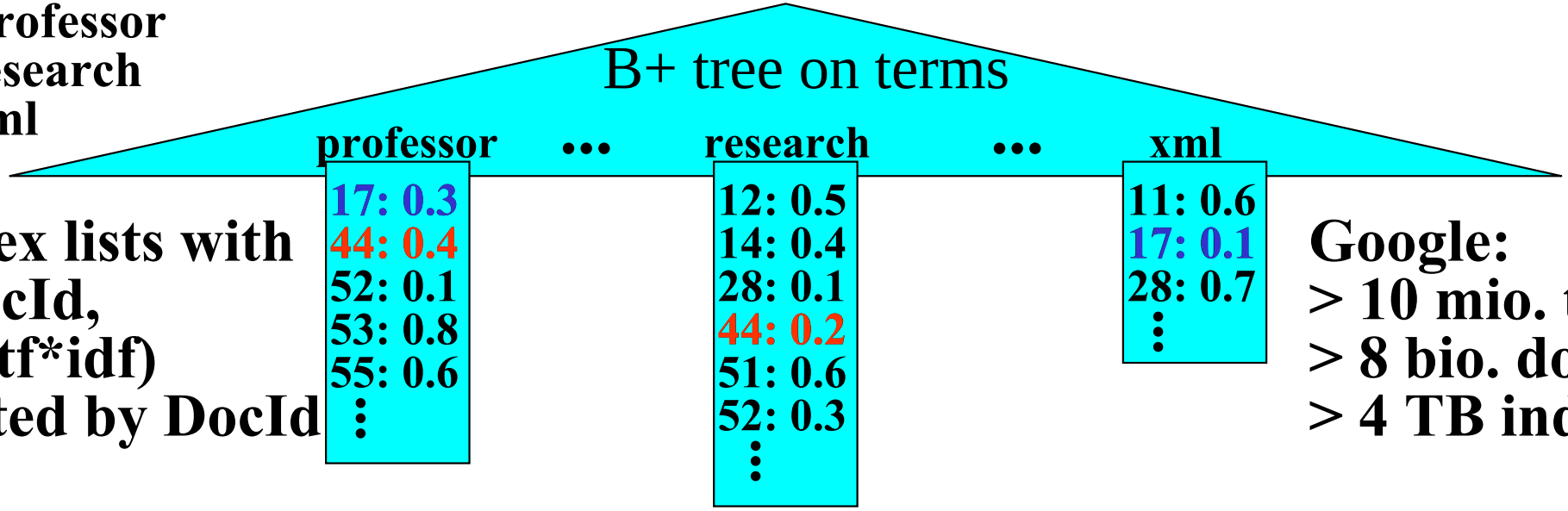
Compute union V of DocId-Sets V1, ..., Vr

4. Sort all d in *V* in descending order of *rank(d)*
5. Retrieve docs d in *V* in sorted order (possibly with heuristic termination criterions),

   eliminate docs *d* that contain one or more terms t_r+1, t_r+2, ...

Vector space model suggests *m×n term-document matrix*,
but data is sparese and queries are even very sparse
→ better use *inverted index lists* with terms as keys for B+ tree

q: professor
research
xml



index lists with (DocId, s = tf*idf) sorted by DocId

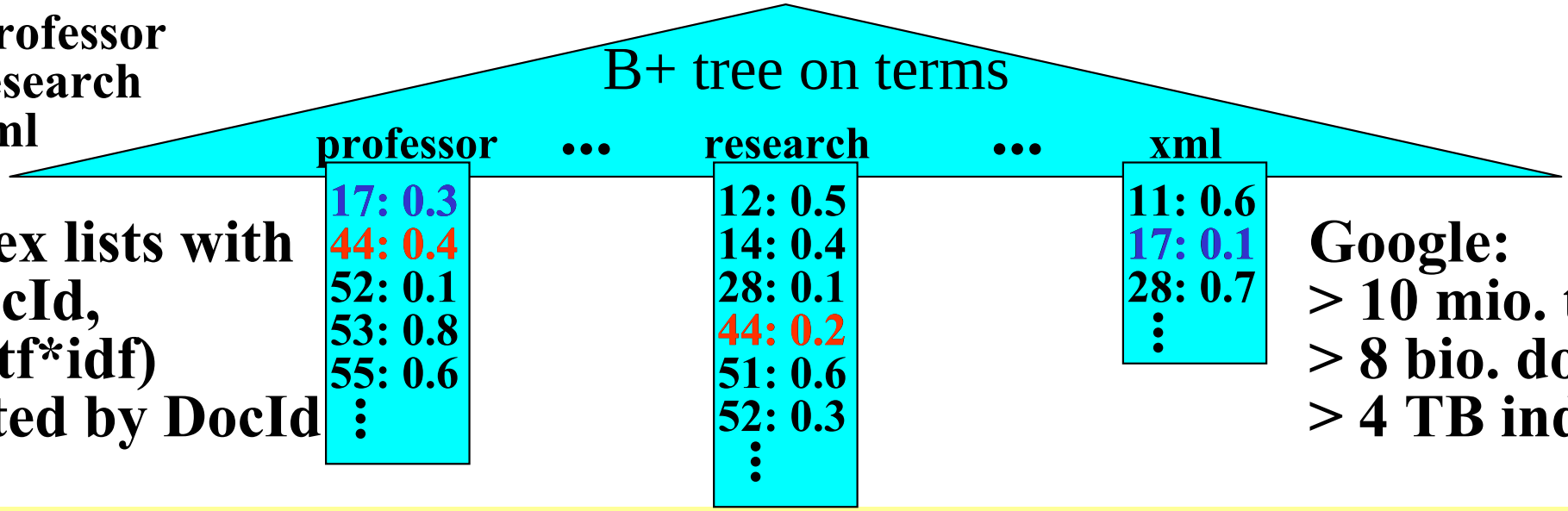
Google:
> 10 mio. terms
> 8 bio. docs
> 4 TB index

terms can be full words, word stems, word pairs,
word substrings, etc.
(whatever „dictionary terms“ we prefer for the application)

queries can be conjunctive or „andish“ (soft conjunction)

DBS-Style Top-k Query Processing

q: professor
research
xml



index lists with (DocId, s = tf*idf) sorted by DocId

Google:
> 10 mio. terms
> 8 bio. docs
> 4 TB index

Given: query $q = t_1\ t_2\ ...\ t_z$ with z (conjunctive) keywords
similarity scoring function score(q,d) for docs $d \in D$, e.g.: $\vec{q} \cdot \vec{d}$
with precomputed scores (index weights) $s_i(d)$ for which $q_i \neq 0$

Find: top k results w.r.t. score(q,d) = $aggr\{s_i(d)\}$ (e.g.: $\Sigma_{i \in q}\ s_i(d)$)

Naive join&sort QP algorithm:

top-k (
$\sigma[term=t_1]$ (index) $\left|\times\right|_{DocId}$
$\sigma[term=t_2]$ (index) $\left|\times\right|_{DocId}$
...
$\sigma[term=t_z]$ (index) $\left|\times\right|_{DocId}$
order by s desc)

Index List Processing by Merge Join

Keep L(i) in **ascending order of doc ids**

Compress L(i) by actually storing the gaps between successive doc ids

(or using some more sophisticated prefix-free code)

QP may start with those L(i) lists that are short and have high idf

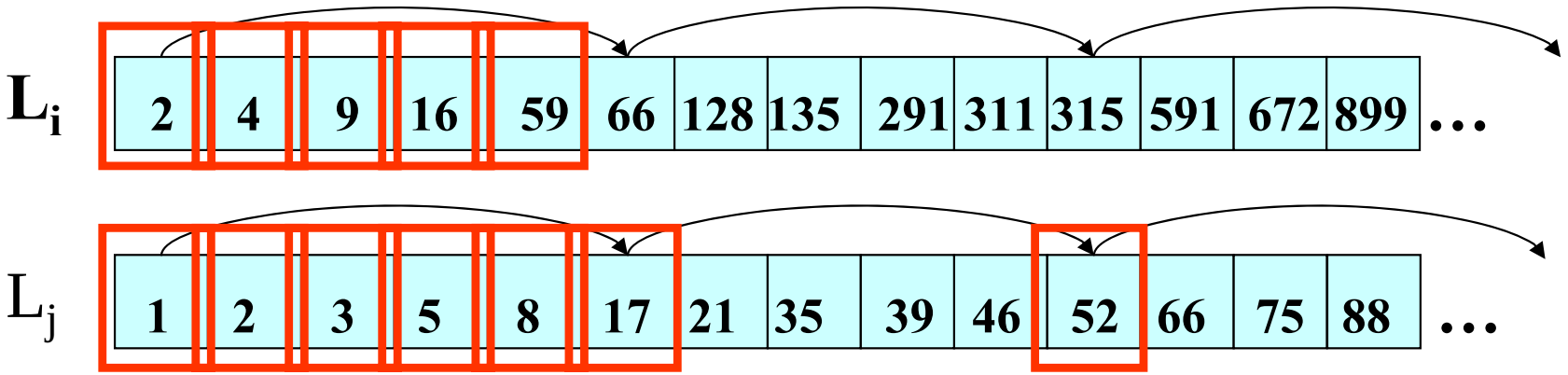
Candidate results need to be looked up in other lists L(j)

To avoid having to uncompress the entire list L(j),

L(j) is encoded into groups of entries

with a skip pointer at the start of each group

→ sqrt(n) evenly spaced skip pointers for list of length n



Computational Model for Top-k Queries

Assume ***local scores*** s_i for query q, data item d, and dimension i, and ***global scores s*** of the form $s(q,d) = aggr\{ s_i(q,d) \mid i = 1..m \}$

with a ***monotonic*** aggregation function $aggr : [0,1]^m \rightarrow [0,1]$

Examples: $s(q,d) = \sum_{i=1}^{m} s_i(q,d)$ $\quad$ $s(q,d) = max\{ s_i(q,d) \mid i = 1..m \}$

Find top-k data items with regard to global scores:

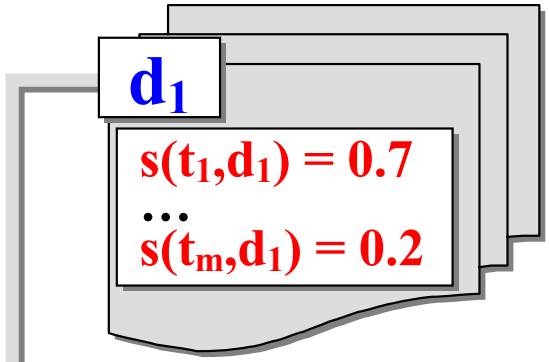
- process m ***index lists*** Li with ***sorted access (SA)*** to entries (d, s_i(q,d)) in ***ascending order of doc ids*** or ***descending order of s_i(q,d)***
- maintain for each candidate d a set E(d) of evaluated dimensions and a ***partial score „accumulator“***
- for candidate d with incomplete E(d) consider looking up d in Li for all i∈R(d) by ***random access (RA)***
- terminate index list scans when enough candidates have been seen
- if necessary sort final candidate list by global score

Efficient Top-k Search

[Buckley85, Güntzer/Balke/Kießling 00, Fagin01]

threshold algorithms: efficient & principled top-k query processing with monotonic score aggr.

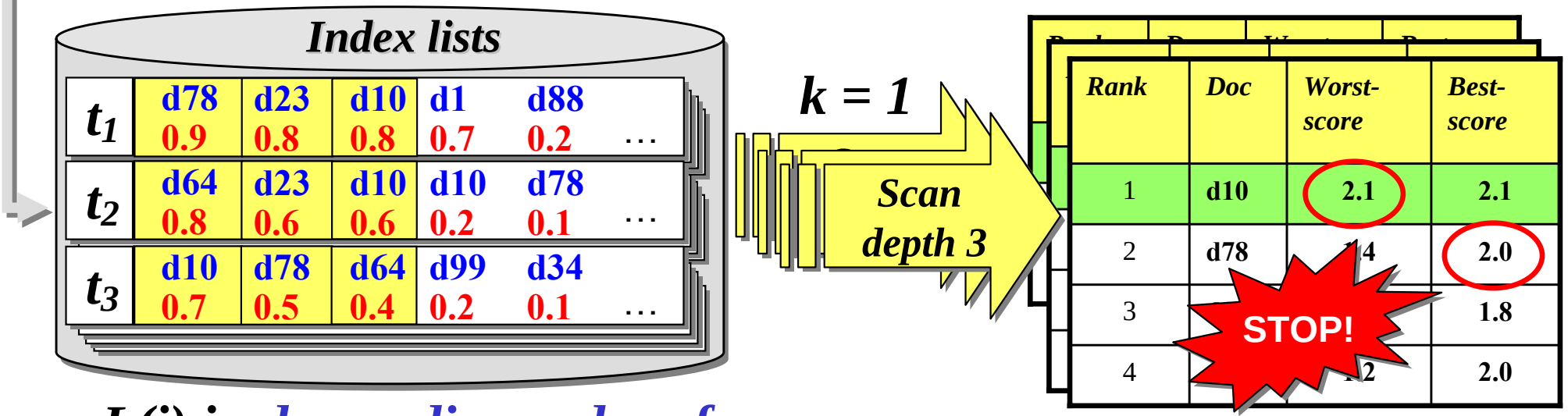
Data items: $d_1, \ldots, d_n$



Query: $q = (t_1, t_2, t_3)$

TA with sorted access only (NRA):

```
can index lists; consider d at pos_i in L_i;
E(d) := E(d) ∪ {i}; high_i := s(t_i,d);
worstscore(d) := aggr{s(t_ν,d) | ν ∈E(d)};
bestscore(d) := aggr{worstscore(d),
                     aggr{high_ν | ν ∉ E(d)}};
if worstscore(d) > min-k then add d to top-k
    min-k := min{worstscore(d') | d' ∈ top-k};
else if bestscore(d) > min-k then
    cand := cand ∪ {d}; s
threshold := max {bestscore(d') | d'∈ cand};
if threshold ≤ min-k then exit;
```



keep L(i) in descending order of scores

Threshold Algorithm (TA, Quick-Combine, MinPro)

(Fagin'01; Güntzer/Balke/Kießling; Nepal/Ramakrishna)

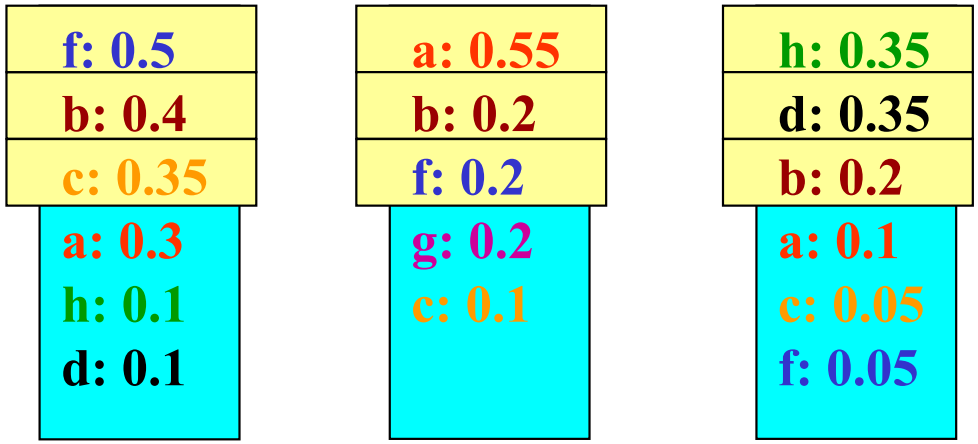
scan all lists L_i (i=1..m) in parallel:
consider dj at position pos_i in Li;
$high_i := s_i(dj)$;
if dj ∉ top-k then {
look up $s_\nu(dj)$ in all lists L_ν with $\nu \neq i$; // random access
compute $s(dj) := aggr \{s_\nu(dj) \mid \nu=1..m\}$;
if s(dj) > min score among top-k then
add dj to top-k and remove min-score d from top-k; };
$threshold := aggr \{high_\nu \mid \nu=1..m\}$;
if min score among top-k ≥ threshold then exit;

but random accesses are expensive !

m=3

aggr: sum

k=2



top-k:

~~f: 0.75~~

a: 0.95

b: 0.8

No-Random-Access Algorithm

(NRA, Stream-Combine, TA-Sorted)

scan index lists in parallel:
consider dj at position pos_i in Li;
E(dj) := E(dj) ∪ {i}; $high_i$:= si(q,dj);
bestscore(dj) := aggr{x1, ..., xm)
with xi := si(q,dj) for i∈E(dj), $high_i$ for i ∉E(dj);
worstscore(dj) := aggr{x1, ..., xm)
with xi := si(q,dj) for i∈E(dj), 0 for i ∉E(dj);
top-k := k docs with largest worstscore;
threshold := bestscore{d | d not in top-k};
if min worstscore among top-k ≥ threshold then exit;

m=3
aggr: sum
k=2

f: 0.5	a: 0.55	h: 0.35
b: 0.4	b: 0.2	d: 0.35
c: 0.35	f: 0.2	b: 0.2
a: 0.3	g: 0.2	a: 0.1
h: 0.1	c: 0.1	c: 0.05
d: 0.1		f: 0.05

top-k:
a: 0.95
b: 0.8

candidates:
f: 0.7 + ? ≤ 0.7 + 0.1
h: 0.35 + ? ≤ 0.35 + 0.5
c: 0.35 + ? ≤ 0.35 + 0.3
d: 0.35 + ? ≤ 0.35 + 0.5
g: 0.2 + ? ≤ 0.2 + 0.4

Definition:

For a class $\mathcal{A}$ of algorithms and a class $\mathcal{D}$ of datasets,
let cost(A,D) be the execution cost of $A \in \mathcal{A}$ on $D \in \mathcal{D}$.
Algorithm B is ***instance optimal*** over $\mathcal{A}$ and $\mathcal{D}$ if

for every $A \in \mathcal{A}$ on $D \in \mathcal{D}$: cost(B,D) = O(cost(A,D)),

that is: cost(B,D) ≤ c*O(cost(A,D)) + c‘
with optimality ratio (competitiveness) c.

Theorem:

- TA is instance optimal over all algorithms that are based on sorted and random access to (index) lists (no „wild guesses“).
  TA has optimality ratio $m + m(m-1) C_{RA}/C_{SA}$
  with random-access cost C_{RA} and sorted-access cost C_{SA}
- NRA is instance-optimal over all algorithms with SA only.

if „wild guesses“ are allowed,
then no deterministic algorithm is instance-optimal

Execution Cost of TA Family

Run-time cost is $O\left(n^{\frac{m-1}{m}} \cdot k^{\frac{1}{m}}\right)$ with arbitrarily high probability

(for independently distributed Li lists)

Memory cost is O(k) for TA

and $O(n^{(m-1)/m})$ for NRA (priority queue of candidates)

Approximation TA:

A *θ-approximation* T‘ for top-k query q with θ > 1
is a set T‘ of docs with:

- |T‘|=k and
- for each d‘∈T‘ and each d‘‘∉T‘: θ *score(q,d‘) ≥ score(q,d‘‘)

Modified TA:

...

Stop when $min_k \geq aggr(high_1, ..., high_m) / \theta$

General heuristics:

- **disregard index lists with idf below threshold**
- **for index scans give priority to index lists that are short and have high idf**

String similarity

String Similarity

Hamming-Distanz von Strings s1, s2 $\in \Sigma^*$ mit |s1|=|s2|:

Anzahl verschiedener Zeichen (Kardinalität von {i: s1i ≠ s2i})

Levenshtein-Distanz (Editierdistanz) von Strings s1, s2 $\in \Sigma^*$:

minimale Anzahl der Editieroperationen auf s1

(Ersetzen, Löschen oder Einfügen eines Zeichens),

um s1 in s2 zu ändern

Für

edit (i, j): Levenshtein-Distanz von s1[1..i] und s2[1..j]

gilt: edit (0, 0) = 0, edit (i, 0) = i, edit (0, j) = j

edit (i, j) = min { edit (i-1, j) + 1,

edit (i, j-1) + 1,

edit (i-1, j-1) + diff (i, j) }

mit diff (i, j) = 1 falls s1i ≠ s2j, 0 sonst

→Berechnung durch dynamische Programmierung

String similarity : Damerau-Levenshtein

Damerau-Levenshtein-Distanz von Strings s1, s2 $\in \Sigma^*$:

minimale Anzahl an Ersetzungs-, Einfüge-, Lösch- oder Transpositionsoperationen (Vertauschen benachbarter Zeichen), um s1 in s2 zu ändern

Für edit (i, j): Damerau-Levenshtein-Distanz von s1[1..i] und s2[1..j] gilt:

$$\text{edit}(0, 0) = 0,\ \text{edit}(i, 0) = i,\ \text{edit}(0, j) = j$$

$$\text{edit}(i, j) = \min \{\ \text{edit}(i-1, j) + 1,$$
$$\text{edit}(i, j-1) + 1,$$
$$\text{edit}(i-1, j-1) + \text{diff}(i, j),$$
$$\text{edit}(i-2, j-2) + \text{diff}(i-1, j) + \text{diff}(i, j-1) + 1\ \}$$

mit $\text{diff}(i, j) = 1$ falls $s1_i \neq s2_j$, 0 sonst

String Similarity: N-Grams

Bestimme für String s die Menge seiner N-Gramme:

G(s) = {Substrings von s mit der Länge N}

(häufig werden Trigramme betrachtet mit N=3)

Ähnlichkeit von Strings s1 und s2:

$|G(s1)| + |G(s2)| - 2|G(s1) \cap G(s2)|$

Beispiel:

G(rodney) = {rod, odn, dne, ney}

G(rhodnee) = {rho, hod, odn, dne, nee}

Ähnlichkeit (rodney, rhodnee) = 4 + 5 – 2*2 = 5

Phonetic Similarity (1)

Soundex-Code:

Abbildung von Wörtern (speziell: Nachnamen) auf 4-Zeichen-Codes,

so daß ähnlich ausgesprochene Wörter denselben Code haben

• Erstes Codezeichen = erstes Zeichen des Wortes

• Codezeichen zwei bis vier (a, e, i, o, u, y, h, w werden ignoriert):

b, p, f, v →1 c, s, g, j, k, q, x, z →2

d, t →3 l →4

m, n →5 r →6

• Aufeinanderfolgende identische Codes werden zusammengefasst

(es sei denn, sie sind durch h getrennt)

Beispiele:

Powers → P620 , Perez →P620

Penny→ P500, Penee→ P500

Tymczak → T522, Tanshik → T522

Phonetic Similarity: Editex

Idea: Editierdistanz mit Berücksichtigung phonetischer Codes

Für editex (i, j): Editex-Distanz von s1[1..i] und s2[1..j]

gilt: editex (0, 0) = 0,

editex (i, 0) = editex (i-1, 0) + d(s1[i-1], s1[i]),

editex (0, j) = editex (0, j-1) + d(s2[j-1], s2[j]),

editex (i, j) = min { editex (i-1, j) + d(s1[i-1], s1[i]),
editex (i, j-1) + d(s2[j-1], s2[j]),
edit (i-1, j-1) + diffcode (i, j) }

mit diffcode (i, j) = 0 falls s1i = s2j,
1 falls group(s1i)= group(s2j), 2 sonst

und d(X, Y) = 1 falls X ≠ Y und X ist h oder w,
diffcode (X, Y) sonst

mit group:

{a e i o u y}, {b p}, {c k q}, {d t}, {l r},

{m n}, {g j}, {f p v}, {s x z}, {c s z}

Demos: advanced models for text IR