

Information Retrieval

Isa-Ali Kirca

Contents

1	Week 1: Recap IR0	2
1.1	Document Parsing	4
1.2	Tokenizing	4
1.3	Stopping	4
1.4	Stemming	5
1.5	Phrases and N-grams	5
1.6	Ranking with indexes	6
1.7	Slides IR0 recap	7
2	Week 2: Evaluation	9
2.1	Test collections	9
2.2	Metrics	10
3	Week 3: Document representation and matching	13
3.1	Query analysis	13
3.2	Query processing	13
3.3	Termbased Retrieval	13
3.4	Semantic-based Retrieval	16

1 Week 1: Recap IR0

After gathering the text we want to search, the next step is to decide whether it should be modified or restructured to simplify searching. The types of changes that are made at this stage are called **text transformation** or, more often, **text processing**.

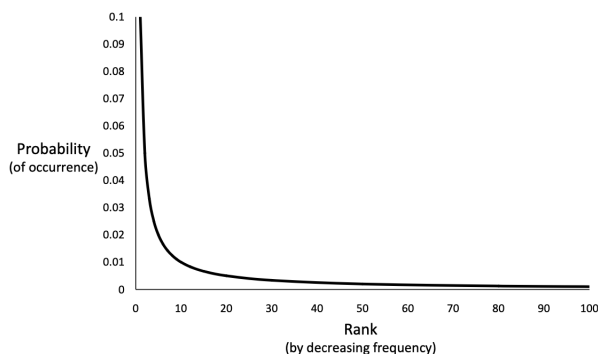
Goal of text processing: convert the many forms in which words can occur into more consistent index terms.

Index terms	representation of the content of a document used for searching.
Tokenization	words are split apart. Process of forming words from the sequence of characters in a document.
Stopping	some words may be ignored entirely in order to make query processing more effective and efficient.
Stemming	allow similar words (like "run" and "running") to match each other.

Statistical models of **word occurrences** are very **important** in information retrieval, and are used in many of the core components of search engines, such as the ranking algorithms, query transformation, and indexing techniques. One of the most obvious features of text is that the **distribution of word frequencies** is very **skewed**.

Zipf's Law: the frequency of the r th most common word is inversely proportional to r or, alternatively, the rank of a word times its frequency (f) is approx. a constant (k): $r \cdot f = k$.

But we want the probability of occurrence of a word, which is the frequency of the word divided by the total number of word occurrences in the text. In this case, **Zipf's law** is: $r \cdot P_r = c$, where P_r is the probability of occurrence for the r th ranked word, and c is a constant.



Hapax Legomena: words that occur once in a text corpus or book.

Heaps' law: Relationship between size of corpus and size of vocabulary.

$$v = k \cdot n^\beta$$

v = vocabulary size for a corpus of size n words
 k and β = parameters that vary for each collection.

Heaps' law predicts that the number of new words will increase very rapidly when the corpus is small and will continue to increase indefinitely, but slower for larger corpora.

Word occurrence statistics can also be used to estimate the size of the results from a web search. A **result** is any document (or web page) that contains all of the query words.

If we assume that words occur *independently* of each other, then the probability of a document containing all the words in the query is simply the **product of the probabilities** of the individual words occurring in a document. For example, if there are three query words a , b , and c , then:

$$P(a \cap b \cap c) = P(a) \cdot P(b) \cdot P(c)$$

$P(a \cap b \cap c) =$ **joint probability**, or the probability that all three words occur in a document

$P(a), P(b), P(c) =$ are the probabilities of each word occurring in a document.

A search engine will always have access to the number of documents that a word occurs in (f_a, f_b , and f_c), and the number of documents in the collection (N), so these probabilities can easily be estimated as $P(a) = f_a/N$, $P(b) = f_b/N$, and $P(c) = f_c/N$:

$$f_{abc} = N \cdot f_a/N \cdot f_b/N \cdot f_c/N = (f_a \cdot f_b \cdot f_c) / N^2 \quad \text{where } f_{abc}: \text{ estimated size of result set.}$$

But, this assumption does **not** lead to good estimates for result size, especially for **combinations of three words**. The problem is that the words in these combinations do not occur independently of each other. If we see the word “fish” in a document, for example, then the word “aquarium” is more likely to occur in this document than in one that does not contain “fish”.

Better estimates are possible if **word co-occurrence** information is also available. Obviously, this would give exact answers for two-word queries. For longer queries, we can improve the estimate by not assuming independence. In general, for three words:

$$P(a \cap b) = \text{probability that the words } a \text{ and } b \text{ co-occur in a document}$$

$$P(a \cap b \cap c) = P(a \cap b) \cdot P(c \mid (a \cap b))$$

$P(a), P(b), P(c) =$ probability that the word c occurs in a document given that the words a and b occur in the document.

These estimates are much better than the ones produced assuming independence, but they are **still too low**.

Skipped the last few paragraphs of 4.2, because I don't think this is very important.

1.1 Document Parsing

Document parsing involves recognizing the content and structure of text document.

Metadata is information about a document that is not part of the text content. Metadata content includes document attributes such as date and author, and, most importantly, the **tags** that are used by **markup languages** to identify document components. Most popular are HTML and XML.

The **parser** uses the **tags** and other metadata recognized in the document to interpret the document's structure based on the syntax of the markup language (**syntactic analysis**) and to produce a representation of the document that includes both the structure and content.

1.2 Tokenizing

Tokenizing is the process of forming words from the sequence of characters in a document. Some examples of issues involving tokenizing that can have significant impact on the effectiveness of search are:

- **Small words:** can be important in combination with other words. For example, master, world war II.
- **Hyphenated and non-hyphenated forms of words**
- **Special characters:** important part of the tags, URLs, code, and other important parts of documents that must be correctly tokenized.
- **Capitalized words:** can have different meaning from lowercase words. For example, "Bush" and "Apple".
- **Apostrophes**
- **Numbers, including decimals:** for example, nokia 3250, quicktime 6.5 pro.
- **Periods can occur in numbers, abbreviations** (e.g., "I.B.M.", "Ph.D."), URLs, ends of sentences, and other situations.

Text processing for queries **must** be the **same** as that used for documents. Otherwise, many of the index terms will simply not match the corresponding terms.

1.3 Stopping

Human language is filled with function words: words that have little meaning apart from other words. The most popular—"the," "a," "an," "that," and "those"—are determiners. These words are part of how we describe nouns in text, and express concepts like location or quantity. Prepositions, such as "over," "under," "above," and "below," represent relative position between two nouns.

In information retrieval, these function words have a second name: **stopwords**. We call them **stopwords** because text processing stops when one is seen, and they are thrown out. **Throwing out** these words **decreases index size**, **increases retrieval efficiency**, and generally **improves retrieval effectiveness**.

Constructing a stopwords list must be done with **caution**. Removing too many words will **hurt** retrieval effectiveness.

If storage space requirements allow, it is best to at least index all words in the documents. If stopping is required, the stopwords can always be removed from queries. If keeping stopwords in an index is not possible because of space requirements, as few as possible should be removed in order to maintain maximum flexibility.

1.4 Stemming

Stemming, also called **conflation**, is a component of text processing that captures the relationships between different variations of a word. More precisely, **stemming reduces** the different forms of a word that occur because of **inflection** (e.g., plurals, tenses) or **derivation** (e.g., making a verb into a noun by adding the suffix -ation) to a common stem.

There are two basic types of **stemmers**: **algorithmic** and **dictionary-based**. An **algorithmic** stemmer uses a small program to decide whether **two words** are **related**, usually based on knowledge of word suffixes for a particular language. By contrast, a **dictionary-based** stemmer has no logic of its own, but instead **relies on pre-created dictionaries** of related terms to store term relationships.

Algorithmic stemmers:

- **Suffix-s stemmer**: assumes any word ending in the letter “s” is plural. Cakes → cake.
- **Porter stemmer**: consists of a number of steps, each containing a set of rules for removing suffixes. At each step, the rule for the longest applicable suffix is executed.

Dictionary (Hybrid)-based stemmers:

- **Krovetz stemmer**: makes constant use of a dictionary to check whether the word is valid. The Krovetz stemmer has the additional advantage of producing stems that, in most cases, are full words, whereas the Porter stemmer often produces stems that are word fragments.

Incorporating language-specific stemming algorithms is one of the most important aspects of customizing, or **internationalizing**, a search engine for multiple languages.

1.5 Phrases and N-grams

A **phrase** is equivalent to a simple **noun phrase**. This is often restricted even further to include just sequences of nouns, or adjectives followed by nouns. Phrases defined by

these criteria can be identified using a **part-of-speech (POS) tagger**. A POS tagger marks the words in a text with labels corresponding to the part-of-speech of the word in that context. Taggers are based on statistical or rule-based approaches and are trained using large corpora that have been manually labeled.

N-gram: any sequence of n words.

Unigram: single words.

Bigrams: sequences of two words.

Unigram: sequences of three words.

The more frequently a word **N-gram** occurs, the **more likely** it is to correspond to a **meaningful phrase in the language**. N-grams of all lengths form a Zipf distribution, with a few common phrases occurring very frequently and a large number occurring with frequency 1. In fact, the rank-frequency data for n-grams (which includes single words) fits the Zipf distribution better than words alone.

1.6 Ranking with indexes

Unsorted arrays are **slow to search**, and **sorted arrays** are **slow at insertion**. By contrast, **hash tables** and **trees** are **fast** for **both search and insertion**. These structures are more complicated than arrays, but the speed difference is compelling.

Text search is very different from traditional computing tasks, so it calls for its own kind of data structure, the **inverted index**.

1.7 Slides IR0 recap

Outline **text analysis**:

- **Statistical properties of written text** (Zipf's law and Heaps' law)
- **Text analysis pipeline**
 1. Remove white-spaces and punctuation
 2. Convert terms to lower-case
 3. Remove stop-words
 - **Frequency-based**:
 - * Set a frequency threshold f
 - * Remove words with the frequency higher than f
 - **Dictionary-based**:
 - * Create a dictionary of stop-words
 - * Remove words that occur in this dictionary
 4. Convert terms to their stems
 5. Deal with phrases
 6. Apply language-specific processing rules
- **Stemming**
 - **Algorithmic**: Porter-stemmer
 - **Dictionary-based**
 - * Large dictionary of related words
 - * Semi-automatic: run → running, runs, **runned**, **runly**
 - * New-words problem
 - **Hybrid**: produces words, **not stems**. Comparable effectiveness with the Porter stemmer
 - * check the word in a dictionary, **if found**, leave it as is
 - * if not found, apply algorithmic stemming (remove suffixes)
 - * check the dictionary again
 - * if not found, apply rules to modify the ending
- **Phrases**
 1. detect noun phrases using a part-of-speech tagger
 - sequences of nouns
 - adjectives followed by nouns
 2. Detect phrases at the query processing time
Use index with word positions
 3. Use frequent **n-grams**, e.g., bigrams and trigrams

Outline indexing:

1. **Data structures:** Web Graph, Forward index, Page attribute file, Inverted index

2. **Inverted index**

a) Dictionary

- Each entry contains
 - Number of pages containing the term
 - Pointer to the start of the inverted list
 - Other meta-data about the term
- B+ tree, hash table

b) Inverted lists

- Document identifiers
- Frequencies
- Positions
- Weights

3. **Constructing an index**

- Simple indexer

Problems of this indexer:

a) In-memory:

- Two-pass index
- One-pass index with merging

b) Single-threaded

- Distributed indexing (MapReduce)

4. **Updating an index**

Strategies:

- No merge (low index maintenance cost, high query processing cost)
- Incremental update
- Immediate merge (always a single index)
- Lazy merge (trade-off between index maintenance and query processing cost)

Page deletions:

- Maintain identifiers of deleted documents in memory, access during query processing
- Garbage collection (e.g., during index merging)

2 Week 2: Evaluation

Evaluation is the key to making progress in building better search engines.

What is **different** about **evaluation** in IR? Many possible queries (cannot be enumerated) and documents (most cannot be judged) and expect good “zero-shot” performance.

- **Boolean retrieval**: considers only whether a document is relevant
 - Does not estimate a **degree** of relevance (i.e., no ranking)
 - Acceptable for some use cases, especially recall oriented
 - Limitations for large result sets
- **Ranked retrieval**: sorts (ranks) documents by estimated relevance
 - Relevance score estimated for each document
 - Common setting (e.g., Web search)
 - No expectation that user consumes entire ranked list
 - Evaluation metrics characterize different trade-offs

Outline **offline evaluation**:

2.1 Test collections

- **Components of test collections**
 - **Test documents**: repr. for appl. in terms of the nr., size, and type.
 - **Test queries**: Queries from pot. users (query log). More is better, at least 50.
 - **Ground Truth/Relevance judgements**: Search result relevant?
 - * **Where to get**: users, independent judges, crowdsourcing.
 - * **How many**: more the better. More judged queries, fewer judgements per query. Multiple judges.
 - * **Graded relevance**: 4 - perfect, 3 - excellent, 2 - good, 1 - fair, 0 - bad.
 - **But, impossible to obtain judgements for all documents. So:**
 - Depth-k pooling**: Produces a large number of judgments for each query
 1. consider multiple search systems (by participants)
 2. consider top-k results from each system
 3. remove duplicates
 4. present documents to judges in a random order
 - still **incomplete**.

- **Multiple assessors**

- * **Inter-assessor agreement**, Cohen's kappa coefficient:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \quad P(E) = \text{Expected chance agreement}$$

- * Values:

- > 0.8 : high
 - $0.67 - 0.8$: acceptable
 - < 0.67 : low

- * For more than two assessors, average pair-wise coefficients

- **Evaluation campaigns**

- Text REtrieval Conference (TREC)
 - Cross-Language Education and Function (CLEF)
 - NII Test Collections for IR (NTCIR)

2.2 Metrics

- **Unranked evaluation**

- **Precision**: is the fraction of retrieved items that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**: is the fraction of relevant items that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

	Relevant	Non-relevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

- **F-measure**

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1) PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1 - \alpha}{\alpha}$$

- **F1-measure** ($\alpha = 0.5, \beta^2 = 1$)

$$F_1 = \frac{2PR}{P + R}$$

- **Ranking of items is not taken into account!**

- **Ranked evaluation**

- **Precision** at rank k $P@k = \frac{\#(\text{relevant items at } k)}{k}$
- **Recall** at rank k $R@k = \frac{\#(\text{relevant items at } k)}{\#(\text{relevant items})}$
- **Reciprocal rank (RR)** $RR = \frac{1}{\text{rank of first relevant item}}$
- **Average precision (AP)** $AP = \frac{\sum_{d \in \text{rel}} P@k_d}{\#(\text{relevant items})}$
- **Average over multiple queries**
 - * mean $P@k$ * MRR
 - * mean $R@k$ * MAP
- **User search behavior is not taken into account!**

- **User-oriented evaluation**

- **Discounted cumulative gain (DCG)**
 - * Graded relevance $R_k \in \{0, 1, 2, 3, 4\}$
 - * **Cumulative Gain**: $CG = \sum_{k=1}^N (2^{R_k} - 1)$
 - * Gain is **discounted** by rank: $D(k) = \frac{1}{\log(k+1)}$
 - * Discounted cumulative gain: $DCG = \sum_{k=1}^N \frac{2^{R_k} - 1}{\log(k+1)}$
 - * Normalized DCG: $NDCG = \frac{DCG}{DCG_{ideal}}$
- **Rank-biased precision (RBP)**
 - * view next item with probability θ
 - * stop with probability $1 - \theta$
 - * Probability of looking at rank k $P(\text{look at } k) = \theta^{k-1}$
 - * Average number of examined items

$$\begin{aligned} \text{Avg. exam} &= \sum_{k=1}^{\infty} k \cdot P(\text{look at } k) \cdot P(\text{stop at } k) \\ &= \sum_{k=1}^{\infty} k \cdot \theta^{k-1} \cdot (1 - \theta) = \frac{1}{1 - \theta} \end{aligned}$$

- * Utility at rank k $U@k = P(\text{look at } k) \cdot R_k = \theta^{k-1} \cdot R_k$
- * Avg. utility of all results

$$RBP = \frac{\sum_{k=1}^N U@k}{\text{Avg. exam}} = (1 - \theta) \cdot \sum_{k=1}^N \theta^{k-1} \cdot R_k$$

- * θ is usually close to 1

– **Expected reciprocal rank (ERR)**

- * Reciprocal rank $RR = \frac{1}{\text{rank of first relevant item}}$
- * If an item is relevant (R_k) then stop
- * Otherwise ($1 - R_k$), continue with probability θ
- * Probability of looking at rank k $P(\text{look at } k) = \prod_{i=1}^{k-1} ((1 - R_i) \cdot \theta)$
- * Probability of reciprocal rank $= \frac{1}{k}$ $P(RR = \frac{1}{k}) = R_k \cdot \prod_{i=1}^{k-1} ((1 - R_i) \cdot \theta)$
- * Expected reciprocal rank

$$\begin{aligned}
 ERR &= \sum_{k=1}^N \frac{1}{k} \cdot P(RR = \frac{1}{k}) \\
 &= \sum_{k=1}^N \frac{1}{k} \cdot \theta^{k-1} \cdot R_k \cdot \prod_{i=1}^{k-1} (1 - R_i)
 \end{aligned}$$

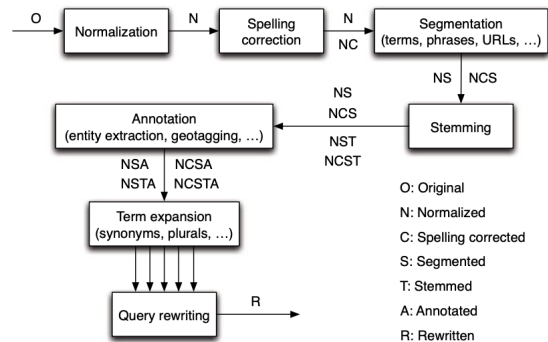
- * θ is usually close to 1

3 Week 3: Document representation and matching

3.1 Query analysis

Pipeline: should be the same as document processing pipeline.

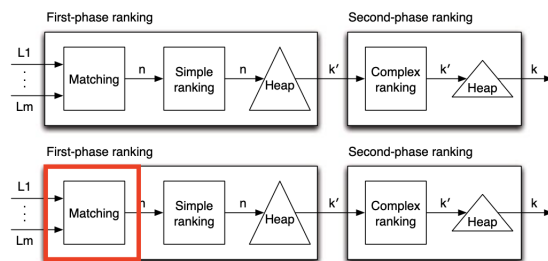
- Normalization
- Spelling correction
- Segmentation
- Stemming
- Term expansion, etc..



3.2 Query processing

Practical considerations:

- Conjunctive mode (AND)
- Document-at-a-time
- A score is usually computed as a linear combination of query-dependent and query-independent scores



3.3 Termbased Retrieval

1. Vector space model

- **Document as vector:** binary occurrence of a term in a document (each document is a vector)
- **Match using Cosine similarity**

$$\text{sim}(d, q) = \cos(\vec{v}(d), \vec{v}(q)) = \frac{\vec{v}(d) \cdot \vec{v}(q)}{\|\vec{v}(d)\| \cdot \|\vec{v}(q)\|} = \frac{\sum_{i=1}^{|V|} d_i \cdot q_i}{\sqrt{\sum_{i=1}^{|V|} d_i^2} \cdot \sqrt{\sum_{i=1}^{|V|} q_i^2}}$$

Where: V is the size of the vocabulary.

- **Instead** of using the binary occurrence, we can use these weights:

– **Term frequency:**

* **Raw term frequency:**

$$tf(t, d)$$

* **Long term frequency:**

$$\begin{cases} 1 + \log tf(t, d) & \text{if } tf(t, d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

– **TF-IDF:**

- * **Inverse document frequency (1):** $idf(t) = \log \frac{N}{df(t)}$
- * **Inverse document frequency (2):** $\max \left\{ 0, \log \frac{N-df(t)}{df(t)} \right\}$

Where:

$$\begin{aligned} df(t) &= \text{document frequency of term } t \\ N &= \text{total number of documents in a collection} \end{aligned}$$

* **TF-IDF:**

$$\text{TF-IDF}(t, d) = \text{tf}(t, d) \cdot \text{idf}(t)$$

2. Language modelling (LM) in IR:

- **Method**

Statistical language model is a probability distr. over sequences of words.

- given a **sequence of length m**
- a LM assigns probability $P(w_1, \dots, w_m)$ to this sequence
- Unigram LM: $P(w_1, \dots, w_m) = P(w_1) \dots P(w_m)$
- Bi-gram LM: $P(w_1, \dots, w_m) = P(w_1) P(w_2 | w_1) \dots P(w_m | w_{m-1})$
- Documents as distributions (1):
 - * **Unigram LM:** $P(t | M_d) = \frac{tf(t, d)}{dl(d)}$
 - * $dl(d)$ is total number of terms in the document (length of document)
 - * this is the **maximum likelihood estimation**
 - * a document is a multinomial distribution over words
 - * if some vocabulary terms do not in document d, then $P(t | M_d) = 0$
 - * addressed by **smoothing**
- How to match the distributions (using query likelihood model QLM) (2)?
 - * **Likelihood** of a doc given a query: $P(d | q) = \frac{P(q|d)P(d)}{P(q)}$
 - * **Prior distr.** over queries $P(q)$ does **not affect** matching a particular query: $P(d | q) \stackrel{\text{rank}}{=} P(q | d)P(d)$
 - * Usually, the **prior distr** over docs $P(d)$ is assumed to be **uniform**: $P(d | q) \stackrel{\text{rank}}{=} P(q | d) = P(q | M_d)$
 - * **"Bag of words"** assumption: terms are **independent**: $P(q | M_d) = \prod_{t \in q} P(t | M_d) = \prod_{t \in q} \frac{tf(t, d)}{dl(d)}$
 - * Match using **KL-divergence**

$$KL(M_d || M_q) = \sum_{t \in V} P(t | M_q) \log \frac{P(t | M_q)}{P(t | M_d)}$$

- **Smoothing** (3)

- **Jelinek-Mercer smoothing**

$$P_s(t | M_d) = \lambda P(t | M_d) + (1 - \lambda) P(t | M_c) \\ = \lambda \frac{tf(t, d)}{dl(d)} + (1 - \lambda) \frac{cf(t)}{cl}$$

Where: $cf(t)$ = collection frequency of term t
 cl = collection length

- **Dirichlet smoothing**

- * A **unigram language model** can be seen as a **multinomial distr.** over words $\mathcal{L}_d(n_1, \dots, n_k | p_1, \dots, p_k)$:

- $n_i = tf(t_i, d)$
- $p_i = P(t_i | M_d)$

- * The **conjugate prior** for **multinomial** is the **Dirichlet distr.**

$P_{\text{prior}}(p_1, \dots, p_k; \alpha_1^{pr}, \dots, \alpha_k^{pr})$:

- $\alpha_i^{pr} = \mu P(t_i | M_c)$
- μ is a smoothing parameter $\left(\lambda = \frac{dl}{dl + \mu}\right)$

- * The **posterior** is the **Dirichlet distr.** with parameters

$$\alpha_i^{po} = n_i + \alpha_i^{pr} = tf(t_i, d) + \mu P(t_i | M_c)$$

- * **Dirichlet smoothing:**

$$P_s(t | M_d) = \frac{tf(t_i, d) + \mu P(t_i | M_c)}{dl(d) + \mu}$$

3. BM25

$$BM25 = \sum_{t \in q} \log \left[\frac{N}{df(t)} \right] \cdot \frac{(k_1 + 1) \cdot tf(t, d)}{k_1 \cdot \left[(1 - b) + b \cdot \frac{dl(d)}{dl_{avg}} \right] + tf(t, d)}$$

- k_1, b - parameters
- $dl(d)$ - length doc d
- dl_{avg} - avg. doc length
- $k_1 = 0 \rightarrow$ summation of idfs
- $k_1 = \infty \rightarrow$ summation of tf-idfs
- $b = 0 \rightarrow$ no normalization by doc length
- $b = 1 \rightarrow$ doc length full effect
- $tf \rightarrow$ compensates (if it's too large for example)

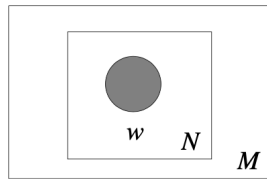
BM25 for long queries

$$BM25 = \sum_{t \in q} \log \left[\frac{N}{df(t)} \right] \cdot \frac{(k_1 + 1) \cdot tf(t, d)}{k_1 \cdot \left[(1 - b) + b \cdot \frac{dl(d)}{dl_{avg}} \right] + tf(t, d)} \cdot \frac{(k_3 + 1) tf(t, q)}{k_3 + tf(t, q)}$$

3.4 Semantic-based Retrieval

1. Topic modelling

Unigram language model



$$W_{ij} \sim \text{Mult}(d_i)$$

$$i \in \{1, \dots, M\}$$

$$j \in \{1, \dots, N_i\}$$

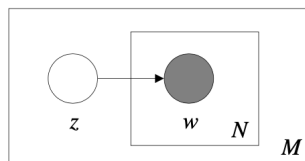
– The circle is a **random variable**.

– Shaded circle is observed, empty is not observed.

– **RV** is occurrence of word

So: we have a collection of M documents, each document has a length of N . On every n th position the word w may either occur or not.

Mixture of unigrams



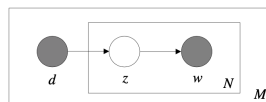
$$z_i \sim \text{Mult}(\theta)$$

$$w_{ij} \sim \text{Mult}(\phi_{z_i})$$

"Arts"	"Budgets"	"Children"
NEW	MILLION	CHILDREN
FILM	TAX	WOMEN
SHOW	PROGRAM	PEOPLE
MUSIC	BUDGET	CHILD
MOVIE	BILLION	YEARS

This time: we first added an unobserved hidden variable z (topic). Now we pick a topic M times. From that topic we sample N times. Topics are for example as depicted in the right figure above: "arts", "budgets" and "children".

Probabilistic latent semantic analysis (pLSA)



$$z_{ij} \sim \text{Mult}(\theta_i)$$

$$w_{ij} \sim \text{Mult}(\phi_{z_{ij}})$$

The William Randolph Hearst Foundation will give \$1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be \$200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive \$400,000 each. The Juilliard School, where music and the performing arts are taught, will get \$250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual \$100,000 donation, too.

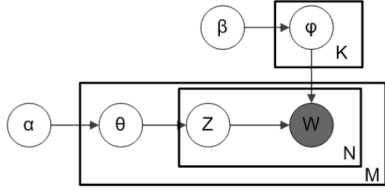
In this case: we assume that every word in the document comes from a **certain topic**. The document is observed and we have M documents. For every position in the document (that is observed) we randomly sample a topic from the multinomial distr. (which is not known). From that topic we sample a word.

So the **probability of a word given a document:**

$$P(w | d) = \sum_z P(w | \phi_z) P(z | \theta_d)$$

This is **better** than before, because we are semantically matching. **Before** we were matching exactly (terms in a query and terms in a document). If the query term did not occur in the document, we used smoothing. **Now**, even if the query term does not occur in the document, but the document is in general about the topic of the query, we still want this document to be ranked **high**.

Latent Dirichlet allocation (LDA)



- Choose $\theta_i \sim \text{Dir}(\alpha)$, where $i \in \{1, \dots, M\}$
- Choose $\phi_k \sim \text{Dir}(\beta)$, where $k \in \{1, \dots, K\}$
- For each position j , where $j \in \{1, \dots, N_i\}$
 - Choose a topic $z_{ij} \sim \text{Mult}(\theta_i)$
 - Choose a word $w_{ij} \sim \text{Mult}(\phi_{z_{ij}})$

This one is basically the same as pLSA, but with priors added.

Estimating LDA: expectation-maximization **[BEYOND IR1!!]**

- E-step:** define the expected value of the log-likelihood function, with respect to the current estimates of the parameters $\theta^{(t)}, \phi^{(t)}$:

$$Q(\theta, \phi \mid \theta^{(t)}, \phi^{(t)}) = \mathbb{E}_{Z|W, \theta^{(t)}, \phi^{(t)}} [\log L(\theta, \phi; W, Z)]$$

- M-step:** find the parameters that maximize this quantity

$$\theta^{(t+1)}, \phi^{(t+1)} = \arg \max_{\theta, \phi} Q(\theta, \phi \mid \theta^{(t)}, \phi^{(t)})$$

- Repeat until convergence

2. Latent semantic indexing/analysis

- Singular Value Decomposition (SVD)**
 - $C = U \Sigma V^T$: $m \times n$ (term document) Σ : diag. $m \times n$ with singular values
 - U : $m \times m$ unitary matrix V^T : $n \times n$ unitary matrix
- Low-rank approximation**

$$C = U \Sigma V^T = \sum_{i=1}^{\min(m,n)} \sigma_i \vec{u}_i \vec{v}_i^T \approx \sum_{i=1}^k \sigma_i \vec{u}_i \vec{v}_i^T = U_k \Sigma_k V_k^T$$

- Latent semantic indexing/analysis**

$$\begin{array}{ccccccc} C & & U_k & & \Sigma_k & & V_k^T \\ (\mathbf{d}_j) & & & & & & (\hat{\mathbf{d}}_j) \\ \downarrow & & & & & & \downarrow \\ (\mathbf{t}_i^T) \rightarrow \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = & (\hat{\mathbf{t}}_i^T) \rightarrow \begin{bmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_k \end{bmatrix} \dots \begin{bmatrix} \mathbf{u}_k \end{bmatrix} & \cdot & \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_k \end{bmatrix} & \cdot & \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_k \end{bmatrix} \end{array}$$

$$d_j = U_k \Sigma_k \hat{d}_j \implies \hat{d}_j = \Sigma_k^{-1} U_k^T d_j$$

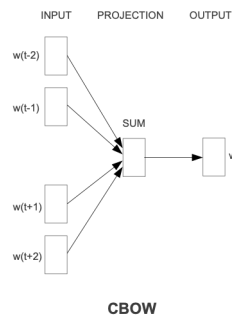
- **Documents as vectors:**

- Given a collection of documents, perform SVD and low-rank approximation to obtain Σ_k and U_k
- Given a document and a query, represent them as vectors in the obtained “semantic” vector space $\hat{d} = \Sigma_k^{-1} U_k^T d$ $\hat{q} = \Sigma_k^{-1} U_k^T q$
- Match the obtained “semantic” vector representations \hat{d} and \hat{q} using cosine similarity

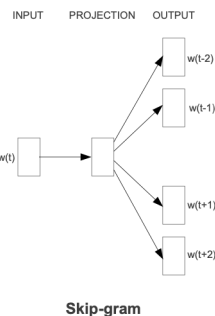
3. Neural models

- **Word embeddings**

Word2Vec



CBOW



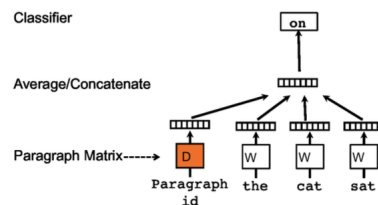
Skip-gram

Documents as vectors

- Compute word embeddings
- Given a document and a query, compute their vector representations as average word embeddings (AWEs)
- Match using cosine similarity

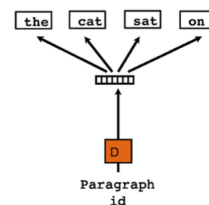
- **Document embeddings**

- **Paragraph vector**



Paragraph vector

- * At every step of stochastic gradient descent, sample a fixed-length context from a random paragraph
- * Compute the error gradient from the network
- * Use the gradient to update parameters



Documents as vectors

- * Compute document embeddings
- * Given a query
 - a) Fix the word matrix W
 - b) Add a (random) column to the document matrix D corresponding to the query repr.
 - c) Update D using gradient descent
 - d) Get the vector repr. of the query from the updated matrix D
- * Match using cosine similarity