

Multimedia Retrieval

Chapter 3: Advanced Text Retrieval

Dr. Roger Weber, roger.weber@ubs.com

- [3.1 Introduction](#)
- [3.2 Natural Language Processing](#)
- [3.3 Web Retrieval](#)
- [3.4 Latent Semantic Analysis](#)
- [3.5 Naive Bayes](#)
- [3.6 Literature and Links](#)

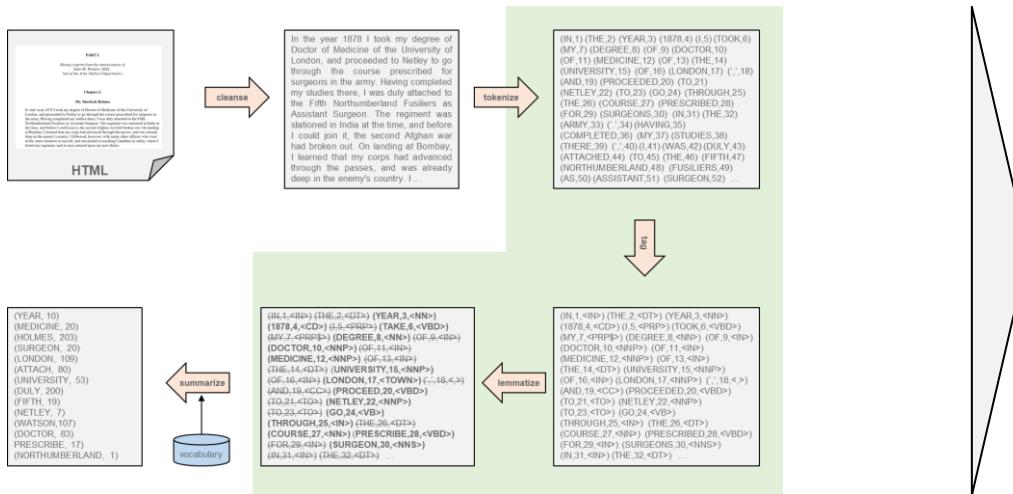


3.1 Introduction

- In the previous chapter, we considered the classical retrieval models. These models have been greatly improved over the past 20 years with the advent of web search, natural language processing, machine learning, and deep learning.
- In this chapter, we focus on the following aspects
 - Natural Language Processing (with NLTK, see below)
 - Web Retrieval with focus on link information (example Google)
 - Latent Semantic Indexing (dimensionality reduction of vocabulary)
 - Naïve Bayes approaches to classify text (language detection, sentiment analysis)
- We are also looking into the python package NLTK which is a good starting point for advanced text processing. To get ready, ensure (as required for your Python environment):

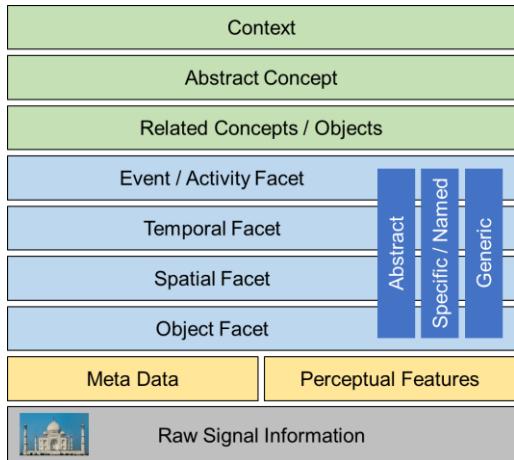
```
sudo pip install -U nltk          # or pip3
sudo pip install -U numpy          # or pip3
python
import nltk
nltk.download()                  # select: popular or all-nltk
```
- Apache OpenNLP is a good package for the Java world (also available through Lucene)

- We focus in this chapter mostly on extraction of higher-level features. First in the classical sense by extending the pipeline from the last chapter



- Extraction of terms beyond simple sequence of characters
- Linguistic transformations (stemming, synonyms, homonyms)
- Structure analysis of sentence (basic part of speech)

- In addition, we apply algorithms / machine learning to infer meta information from the text



- Link analysis to understand importance and relationships of page
- Automated extraction of topics through vocabulary analysis
- Extraction of concept / classifications based on machine learning approaches

3.2 Natural Language Processing

- We extend the feature extraction pipeline from the previous chapter. Step 1 & 5 remain the same, but we extend step 2, 3 and 4 and look into some examples

1. Cleanse document and reduce to sequence of characters
- 2. Create tokens from sequence**
- 3. Tag token stream with additional information**
- 4. Lemmatization, spell checking, and linguistic transformation**
5. Summarize to feature vector (given a vocabulary)

3.2.1 Step 2: Create Tokens

- **Segmentation:** consider a book with several chapters, sections, paragraphs, and sentences. The goal of segmentation is to extract this meta structure from the text (often with the information provided by the previous step). While the broader segmentations (e.g., chapters) require control information from the document, sentence segmentation is possible on the text stream alone:
 - If we observe a ? or a !, a sentence ends (quite unambiguous, but this line is an exception)
 - The observation of a . (period) is rather ambiguous: it is not only used for sentence boundaries, but also in abbreviations, numbers, and ellipses that do not terminate a sentence
 - Some language specifics like ¿ in Spanish
 - Sentence-final particles that do not carry content information but add an effect to the sentence
 - Japanese: カ *ka*: question. It turns a declarative sentence into a question.
 - クセ *kke*: doubt. Used when one is unsure of something.
 - ナ *na*: emotion. Used when one wants to express a personal feeling.
 - English: Don't do it, **man**. The blue one, **right?** The plate isn't broken, **is it?**
 - Spanish: Te gustan los libros, ¿**verdad**? Le toca pasar la aspiradora, ¿**no**?
- A good heuristic works as follows (95% accuracy with English):
 1. If it is a ‘?’ or ‘!', the sentence terminates
 2. If it is a ‘.’, then
 - a. if the word before is a known abbreviation, then the sentence continues
 - b. if the word afterwards starts with capital letter, then the sentence terminates
- The approach in NLTK uses a trained method (Punkt) to determine sentence boundary.

- **Token Generation:** There are different ways to create tokens: a) Fragments of words, b) Words, and c) Phrases (also known as n-grams).
 - **Fragments of words:** an interesting approach in fuzzy retrieval is to split words into sequences of characters (so-called k-grams). For example:

street → str, tre, ree, eet

streets → str, tre, ree, eet, ets

streets → str, tre, ret, ets

An obvious advantage is that different inflections still appear similar at the fragment level. It also compensates for simple misspellings or bad recognition (OCR, speech analysis). Further, no language specific lemmatization is required afterwards. An early example was *EuroSpider* a search engine that used 3-grams to index OCR texts. However, while the technology was compelling, it has become superficial with the increased recognition and correction capabilities. In other retrieval scenarios, the method is still of interest. Music retrieval, DNA retrieval, and Protein Sequencing use fragments to model characteristic features. In linguistic analysis, n-grams of words also play an important role for colocation analysis.

- **Words:** using words as terms is the usual approach. But there are some subtle issues to deal with. For instance, how do you tokenize the following sequences?

Finland's capital → Finland, Finlands, or Finland's?

what're, I'm, isn't → What are, I am, is not?

I'ensemble → le ensemble?

San Francisco → one token or two?

m.p.h., PhD. → ??

\$380.2, 20% → ??

Leuchtrakete → one word or composite word?

- **Words** (contd): In most languages, tokenization can use (space) separators between words. In Japanese and Chinese, words are not separated by spaces. For example:

莎拉波娃现在居住在美国东南部的佛罗里达。

莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

Sharapova now lives in US southeastern Florida

In Japanese, texts can use different formats and alphabets mixed together.

- The conventional approach for tokenization is based on a regular expression to split words. One way to do so is as follows:

1. Match abbreviations with all upper case characters (e.g., U.S.A.)
2. Match sequences of word characters including hyphens (-) and apostrophes (‘)
3. Match numbers, currencies, percentage, and similar (\$2.3, 20%, 0.345)
4. Match special characters and sequences (e.g., ... ; “” () [])

- In addition, we want to consider special expressions/controls in the environment like hashtags (#blowsyourmind), user references (@thebigone), emoticons (☺), or control sequences in the format (e.g., wiki).
- NLTK uses the Treebank tokenizer and the Punkt tokenizer depending on the language. There are a few simpler methods that split sequences on whitespaces or regular expression.
- For Japanese and Chinese, we can identify token boundaries with longest matches in the sequences that form a known word from the dictionary. This approach does not work in other languages.

- **Phrases:** we have seen some examples, where it seems more appropriate to consider subsequent words as a singular term (e.g., New York, San Francisco, Sherlock Holmes). In other examples, the combinations of two or more words can change or add to the meaning beyond the words. Examples include express lane, crystal clear, middle management, thai food, Prime Minister, and other compounds. To capture them, we can extract so-called n-grams from the text stream:

1. Extract the base terms (as discussed before)
2. Iterate through the term sequence
 - Add 2-grams, 3-grams, ..., n-grams over subsequent terms at a given position

However, this leads to many meaningless compounds such as “the house”, “I am”, “we are”, or “it is” which are clearly not interesting to us. More over, we generate thousands of new term groups that are just accidentally together (like “meaningless compounds” or “better control” in this paragraph). To better control the selection of n-grams, various methods have been proposed. We consider here only two simple and intuitive measures:

- A first approach is to reject n-grams that contain at least one so-called stop word. A stop word is a linguistic element that bears little information in itself. Examples include: a, the, I, me, your, by, at, for, not, ... Although very simple, this already eliminates vast amounts of useless n-grams.
- **Pointwise Mutual Information (PMI).** For simplicity, we consider only the case of 2-grams but generalization to n-grams is straightforward. The general idea is that the 2-gram is interesting only if it occurs more frequently than the individual distributions of the two terms would suggest (and assuming they are independent). To this end, we can compute the Pointwise Mutual Information pmi for two terms t_1 and t_2 as follows; $p(t)$ is that probability that term t occurs:

$$pmi(t_1, t_2) = \log \frac{p(t_1, t_2)}{p(t_1) \cdot p(t_2)} = \log \frac{p(t_1|t_2)}{p(t_1)} = \log \frac{p(t_2|t_1)}{p(t_2)} = \log p(t_1, t_2) - \log p(t_1) - \log p(t_2)$$

- **Pointwise Mutual Information** (contd): Let $p(t_j)$ be the probability that we observe the term t_j in the text. We compute this probability with a maximum likelihood approach. Let M be the number of different terms in the collection, $tf(t_j)$ be the so-called **term frequency** of term t_j (number of its occurrences), and N be the total occurrences of all terms in the text. We then obtain $p(t_j)$ as:

$$p(t_j) = \frac{tf(t_j)}{N} \quad \forall j: 1 \leq j \leq M$$

$$\text{similarly: } p(t_1, t_2) = \frac{tf(t_1, t_2)}{N}$$

Now, assume we have two terms t_1 and t_2 . If they are independent from each other, then the probability $p(t_1, t_2)$ of their co-occurrence is the product of their individual probabilities $p(t_j)$ and the *pmi* becomes 0. If t_2 always follows t_1 , then $p(t_2|t_1) = 1$ and the *pmi* is positive and large. If t_2 never follows t_1 , then $p(t_2|t_1) = 0$ and $pmi = -\infty$. So, we keep 2-grams if their *pmi* is positive and large, and dismiss them otherwise. In addition, we dismiss infrequent 2-grams with $tf(t_1, t_2) < \text{threshold}$ to avoid accidental co-occurrences with high *pmi* (seldom words):

| Bigram | $tf(t_1)$ | $tf(t_2)$ | $tf(t_1, t_2)$ | $pmi(t_1, t_2)$ |
|-------------------|-----------|-----------|----------------|-----------------|
| salt lake | 11 | 10 | 10 | 11.94 |
| halliday private | 5 | 12 | 5 | 11.81 |
| scotland yard | 8 | 9 | 6 | 11.81 |
| lake city | 10 | 23 | 9 | 10.72 |
| private hotel | 12 | 14 | 6 | 10.59 |
| baker street | 6 | 29 | 6 | 10.54 |
| brixton road | 15 | 28 | 13 | 10.38 |
| jefferson hope | 37 | 56 | 34 | 9.47 |
| joseph stangerson | 13 | 47 | 10 | 9.46 |
| enoch drebber | 8 | 62 | 8 | 9.44 |
| old farmer | 39 | 9 | 5 | 9.26 |
| john rance | 39 | 10 | 5 | 9.11 |
| john ferrier | 39 | 62 | 29 | 9.01 |
| sherlock holmes | 52 | 98 | 52 | 8.78 |

3.2.2 Step 3: Tagging of Tokens

- A simple form of tagging is to add position information to the tokens. Usually, this is already done at token generation time (term position in stream).
- For natural language processing, tagging associates a linguistic or lexical category to the term. With **Part of Speech (POS)**, we label terms as nouns, verbs, adjectives, and so on. Based on this information, we can construct tree banks to define the syntactic and semantic structure of a sentence. Tree banks have revolutionized computational linguistic in the 1990s with “The Penn Treebank” as first large-scale empirical data set. It defines the following tags:

| Tag | Description |
|------|--|
| CC | Coordinating conjunction |
| CD | Cardinal number |
| DT | Determiner |
| EX | Existential there |
| FW | Foreign word |
| IN | Preposition or subordinating conjunction |
| JJ | Adjective |
| JJR | Adjective, comparative |
| JJS | Adjective, superlative |
| LS | List item marker |
| MD | Modal |
| NN | Noun, singular or mass |
| NNS | Noun, plural |
| NNP | Proper noun, singular |
| NNPS | Proper noun, plural |
| PDT | Predeterminer |
| POS | Possessive ending |
| PRP | Personal pronoun |

Proper nouns are specific people, places, things.

| Tag | Description |
|-------|---------------------------------------|
| PRP\$ | Possessive pronoun |
| RB | Adverb |
| RBR | Adverb, comparative |
| RBS | Adverb, superlative |
| RP | Particle |
| SYM | Symbol |
| TO | to |
| UH | Interjection |
| VB | Verb, base form |
| VBD | Verb, past tense |
| VBG | Verb, gerund or present participle |
| VBN | Verb, past participle |
| VBP | Verb, non-3rd person singular present |
| VBZ | Verb, 3rd person singular present |
| WDT | Wh-determiner |
| WP | Wh-pronoun |
| WP\$ | Possessive wh-pronoun |
| WRB | Wh-adverb |

with NLTK, use
`nltk.help.upenn_tagset()`

WH-words are: where, what, which, when, ...

- NLTK also provides a simpler variant with the universal POS tagset. It is based on the same (machine learning) approach as the Penn Treebank but maps tags to a smaller/simpler set. Here is an example together with the number of occurrences in the book “A Study in Scarlet”:

| Tag | Description | Freq | Examples |
|------|---------------------|------|--|
| ADJ | adjective | 2812 | new, good, high, special, big, local |
| ADP | adposition | 5572 | on, of, at, with, by, into, under |
| ADV | adverb | 2607 | really, already, still, early, now |
| CONJ | conjunction | 1711 | and, or, but, if, while, although |
| DET | determiner, article | 5307 | the, a, some, most, every, no, which |
| NOUN | noun | 9358 | year, home, costs, time, Africa |
| NUM | numeral | 354 | twenty-four, fourth, 1991, 14:24 |
| PRT | particle | 1535 | at, on, out, over per, that, up, with |
| PRON | pronoun | 5705 | he, their, her, its, my, I, us |
| VERB | verb | 8930 | is, say, told, given, playing, would |
| . | punctuation marks | 7713 | . , ; ! |
| X | other | 36 | ersatz, esprit, dunno, gr8, univeristy |

POS tags are the basis for natural language processing (NLP). They are used to define a parse tree which allows the extraction of context and the transformation of sentences. **Named entities** is one such transformation. Based on the initial POS tagging and with the help of a entity database, individual tokens or groups of tokens are collapsed to a single named entity.

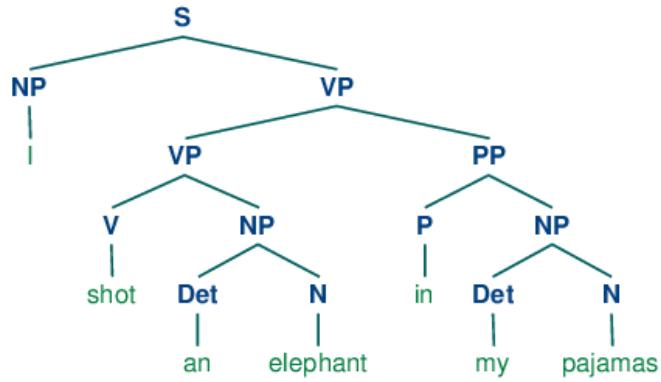
Chunking is the more generic technique. We can define a simple grammar which is used to construct **non-overlapping phrases**. For example, the grammar “NP: {<DT>?<JJ>*<NN>}” collapses a sequence of article, adjectives, and noun into a new group.

- To analyze the structure of sentences, we need a grammar much like for a programming language. Unlike programming languages, natural language grammar is not perfect and contains lots of ambiguities that make it hard (even for humans) to understand the context:

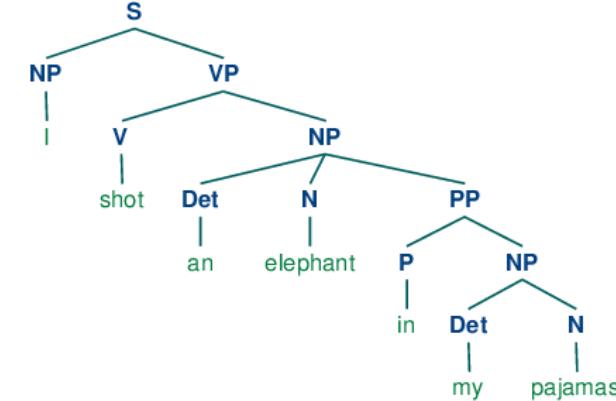
While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know.

- The phrase “in my pajamas” is ambiguous and could relate to the subject “I” who is in pajamas or the object “elephant” being in the pajamas of the subject. Grammar alone cannot resolve ambiguities but the context can help to resolve them (see next sentence above)
- A simple way to analyze sentences is through substitutions of complex phrases into smaller ones. Similar to a grammar of a programming language, we obtain simple productions that allow to parse the sentence through shift-reduce parsers. The labels NP, VP, and PP stand for noun phrase, verb phrase and prepositional phrase respectively

Interpretation 1



Interpretation 2



- Demo: <https://corenlp.run>

3.2.3 Step 4: Lemmatization and Linguistic Transformation

- Lemmatization and linguistic transformation are necessary to match query terms with document terms even if they use different inflections or spellings (colour vs. color). Depending on the scenario, one or several of the following methods can be applied.
- A very common step is **stemming**. In most languages, words appear in many different inflected forms depending on time, case, or gender. Examples:
 - English: go, goes, went, going, house, houses, master, master's
 - German: gehen, gehst, ging, gegangen, Haus, Häuser, Meister, Meisters

As we see from the examples, the inflected forms vary greatly but essentially do mean the same. The idea of stemming is to reduce the term to a common stem and use this stem to describe the context. In many languages, like German, stemming is challenging due to its many irregular forms and the use of strong inflection (gehen → ging). In addition, some languages allow the construction of “new terms” through compound techniques which may lead to arbitrarily long words:

- German (law in Mecklenburg-Vorpommern, 1999-2013): Rinderkennzeichnungs- und Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz. Literally ‘cattle marking and beef labeling supervision duties delegation law’
- Finnish: *atomiydinenergiareaktorigeneraattorilauhduttajaturbiiniratasvaihde*. Literally 'atomic nuclear energy reactor generator condenser turbine cogwheel stage'

In many cases, we want to decompose the compounds to increase chances to match against query terms. Otherwise, we may never find that German cattle law with a query like “Rind Kennzeichnung”. On the other side, breaking a compound may falsify the true meaning

- German: Gartenhaus → Garten, Haus (ok, not too far away from the true meaning)
- German: Wolkenkratzer → Wolke, Kratzer (no, this is completely wrong)

- For English, the **Porter Algorithm** determines a near-stem of words that is not linguistic correct but in most cases, words with the same linguistic stem are reduced to the same near-stem. The algorithm is very efficient and several extensions have been proposed in the past. We consider here the original version of Martin Porter from 1980:
 - Porter defines v as a „vocal“ if
 - it is an **A, E, I, O, U**
 - it is a **Y** and the preceding character is not a „vocal“ (e.g. **RY, BY**)
 - All other characters are consonants (c)
 - Let c be a sequence of consonants, and let v be a sequence of vocals
 - Each word follows the following pattern:
 - $[c] (vc)^m [v]$
 - m is the measure of the word
 - further:
 - $*_o$: stem ends with cvc; second consonant must not be W, X or Y (-WIL, -HOP)
 - $*_d$: stem with double consonant (-TT, -SS)
 - $*_{v*}$: stem contains a vocal
 - The following rules define mappings for words with the help of the forms introduced above. m is used to avoid overstemming of short words.

Source: Porter, M.F.: *An Algorithm for Suffix Stripping*. Program, Vol. 14, No. 3, 1980

- Porter algorithm - extracts (1)

| Rule | Examples |
|--|--|
| Step 1 | |
| a) SSES → SS IES → I | caresses → caress ponies → poni |
| SS → SS S → | caress → caress cats → cat |
| b) (m>0) EED → EE (*v*) ED → (*v*) ING → | feed → feed plastered → plaster motoring → motor |
| ... (further rules) | |
| Step 2 | |
| (m>0) ATIONAL → ATE (m>0) TIONAL → TION | relational → relate conditional → condition |
| (m>0) ENCI → ENCE (m>0) IZER → IZE | valenci → valence digitizer → digitize |
| ... (further rules) | |

- Porter algorithm - extracts (2)

| Rule | Examples |
|--|----------------------------------|
| Step 3 | |
| $(m>0)$ ICATE \rightarrow IC | triplicate \rightarrow triplic |
| $(m>0)$ ATIVE \rightarrow | formative \rightarrow form |
| $(m>0)$ ALIZE \rightarrow AL | formalize \rightarrow formal |
| ... (further rules) | |
| Step 4 | |
| $(m>1)$ and $(^S \text{ or } ^T)$ ION \rightarrow | adoption \rightarrow adopt |
| $(m>1)$ OU \rightarrow | homologou \rightarrow homolog |
| $(m>1)$ ISM \rightarrow | platonism \rightarrow platon |
| ... (further rules) | |
| Step 5 | |
| a) $(m>1)$ E \rightarrow | rate \rightarrow rate |
| $(m=1)$ and $(\text{not } ^o)$ E \rightarrow | cease \rightarrow ceas |
| b) $(m>1 \text{ and } ^d \text{ and } ^L)$ \rightarrow single letter | controll \rightarrow control |

- There are several variants and extensions of the Porter Algorithm. **Lancaster** uses a more aggressive stemming algorithm that can result in almost obfuscated stems but at increased performance. **Snowball** is a set of rule based stemmers for many languages. An interesting aspect is the domain specific language to define stemmers, and compilers to generate code in many computer languages.
- In contrast to the rule based stemmers, a dictionary based stemmer reduces terms to a linguistic correct stem. This comes at additional stemming costs and the need to maintain a dictionary. The EuroWordNet initiative develops a semantic dictionary for many of the European languages. Next to words, the dictionary also contain all inflected forms, a simplified rule-based stemmer for regular inflections, and semantic relations between words (so-called ontologies).
 - Examples of such dictionaries / ontologies:
 - EuroWordNet: <http://www.illc.uva.nl/EuroWordNet/>
 - GermaNet: <http://www.sfs.uni-tuebingen.de/lsd/>
 - WordNet: <http://wordnet.princeton.edu/>
 - We consider in the following the English version of WordNet with its stemmer Morphy. It consists of three parts
 - a simple rule-based stemmer for regular inflections (-ing, -ed, ...)
 - an exception list for irregular inflections
 - a dictionary of all possible stems of the language

- The rule-based approach is quite similar to the Porter rules but they only apply to certain word types (noun, verb, adjective).
- The stemming works as follows:

1. Search the current term in the dictionary. If found, return the term as its own stem (no stemming required)
2. Search the current term in the exception lists. If found, return the associated linguistic stem (see table below)
3. Try all rules as per the table on the right. Replace the suffix with the ending (we may not know the word type, so we try all of them)
 - a. If a rule matches, search in the indicated dictionary for the reduced stem. If found, return it as the stem
 - b. If several rules succeed, choose the more likely stem

Example: axes → axis, axe
4. If no stem is found, return the term as its own stem

| Type | Suffix | Ending |
|------|--------|--------|
| NOUN | s | s |
| NOUN | ses | s |
| NOUN | xes | x |
| NOUN | zes | z |
| NOUN | ches | ch |
| NOUN | shes | sh |
| NOUN | men | man |
| NOUN | ies | y |
| VERB | s | |
| VERB | ies | y |
| VERB | es | e |
| VERB | es | |
| VERB | ed | e |
| VERB | ed | |
| VERB | ing | e |
| VERB | ing | |
| ADJ | er | |
| ADJ | est | |
| ADJ | er | e |
| ADJ | est | e |

adj.exc (1500):

...
 stagiest stagy
 stalkier stalky
 stalkiest stalky
 stapler stapler
 starchier starchy
 starchiest starchy
 starer starer
 starest starest
 starrier starry
 starriest starry
 statelier stately
 stateliest stately
 ...

verb.exc (2400):

...
 ate eat
 atrophied atrophy
 averred aver
 averring aver
 awoke awake
 awoken awake
 babied baby
 baby-sat baby-sit
 baby-sitting baby-sit
 back-pedalled back-pedal
 back-pedalling back-pedal
 backbit backbite
 ...

noun.exc (2000):

...
 neuromata neuroma
 neuroptera neuropteron
 neuroses neurosis
 nevi nevus
 nibelungen nibelung
 nidi nidus
 nielli niello
 nilgai nilgai
 nimbi nimbus
 nimbostrati nimbostratus
 noctilucae noctiluca
 ...

- NLTK supports Porter, Lancaster, Snowball and WordNet stemmers. The table below shows examples for all stemmers. Note that the Morphy implementation in NLTK requires a hint for the word type, otherwise it considers the term as a noun.

| Term | Porter Stem | Lancaster Stem | Snowball Stem | WordNet Stem |
|------------|-------------|----------------|---------------|--------------|
| took | took | took | took | take |
| degree | degre | degr | degre | degree |
| doctor | doctor | doct | doctor | doctor |
| medicine | medicin | medicin | medicin | medicine |
| university | univers | univers | univers | university |
| proceeded | proceed | process | proceed | proceed |
| course | cours | cours | cours | course |
| surgeons | surgeon | surgeon | surgeon | surgeon |
| army | armi | army | armi | army |
| completed | complet | complet | complet | complete |
| studies | studi | study | studi | study |
| there | there | ther | there | there |
| was | wa | was | was | be |
| duly | duli | duly | duli | duly |
| fifth | fifth | fif | fifth | fifth |
| fusiliers | fusili | fusy | fusili | fusiliers |
| assistant | assist | assist | assist | assistant |
| regiment | regiment | regy | regiment | regiment |
| stationed | station | stat | station | station |
| time | time | tim | time | time |
| afghan | afghan | afgh | afghan | afghan |
| had | had | had | had | have |
| broken | broken | brok | broken | break |

- When analyzing text or parsing a user query, we will come across **homonyms** (equal terms but different semantics) and **synonyms** (different terms but equal semantics). Homonyms may require additional annotations from the context to extract the proper meaning. Synonyms are useful to expand a user query if the original search is not (that) successful. Examples:

- **Homonyms** (equal terms but different semantics):

- bank (shore vs. financial institute)

- **Synonyms** (different terms but equal semantics):

- walk, go, pace, run, sprint

WordNet groups English words into so-called synsets or synonym sets and provides short definitions for their usage. It also contains further relations among synsets:

- **Hypernyms** (umbrella term) / **Hyponym** (species)

- **Animal** ← dog, cat, bird, ...

- **Holonyms** (is part of) / **Meronyms** (has parts)

- **door** ← lock

These relationships define a knowledge structure. The hypernym/hyponym relationship defines a hierarchy with synsets at each level and the unique top synset “entity”. We can use this structure to derive further information or context data for our annotations. For instance, if we find the term horse, we can try to derive whether the text is about an animal or about a chess figure.

- NLTK provides the corpus `nltk.corpus.wordnet` which provides access to the WordNet knowledge base. You can also browse through the structure online.

- **Spell checking:** for user queries, we often use spell checkers to fix simple misspellings or to suggest corrected versions of the terms. Most systems provide a fuzzy search which automatically looks for similar terms and adds them to the query if necessary (see Lucene later on)

3.3 Web Retrieval

- Web Retrieval was first performed like ordinary text retrieval. But soon it was clear that web retrieval is entirely different. At the time Google started, the earlier search engines all used vector space retrieval or some form of probabilistic retrieval. Google was the first engine to use ordinary Boolean retrieval but enhanced with a clever ranking system that we will consider in the following. Although the mechanics of the Google search are well kept secrets, we know from the early prototypes of Brin and Page at the Stanford University how the search engine works.
- We first consider the differences between classical and web retrieval. Not only the size varies, but also the quality and how people are searching for information:

| | Classical Retrieval | Web Retrieval |
|-------------------|---------------------------------------|--|
| Collection | controlled set | uncontrolled, incomplete |
| Size | small to large (20 GB) | extremely large (>10PB) |
| Documents | homogenous | heterogeneous (HTML, PDF, ASCII) |
| Structure | homogenous | heterogeneous |
| Links | seldom (citations of other documents) | lots of links among documents |
| Quality | good to excellent | broad range of quality: poor grammar, wrong contents, incorrect, spamming, misspellings, click baits |
| Queries | precise and structures | short and imprecise, names! |
| Results | small number of hits (<100) | large numbers of hits (>1,000,000) |

- These days, a web search engine has to deal with **40+ billion pages, 60+ trillion unique URLs**, and an **index size of 100+PB**. A typical query returns several millions of hits but users expect the top page (or the first link) to be the most relevant for them. But how can we find the most relevant documents for queries with one or two terms given that millions of pages contain them?
 - Example query=“ford”: what do you expect at the top of the list? The car manufacturer, the president, or a ford to cross a river?
 - Example query =“uni basel”: what should be at the top? this course? the main page of the university?
 - Example query=“it”: is the movie the best answer? the book by Stephen King? an IT company? the definition of “it” as a pronoun?
- With all the examples above, it is clear that the short queries are not sufficient to define what is relevant. So Brin and Page considered what users actually want to see and designed their search algorithms to optimize towards this most common information need. With all the queries above, the average user is expecting the page he/she most likely wants to visit. Hence, if more people are interested in ford as the car manufacturer, than that page should be at top. The answers may change over time! As we see with “it”, a recently released movie, the results can depend very much on current events and rankings can drastically change over time.
- In summary: when the context is not clear, and when the query is ambiguous, a web search should return the page at the top that most people consider relevant.
 - This may not be your interpretation of “what is best”. But it is the average interpretation of all internet users.
 - This concept is not entirely new: broadcast stations have always played those songs that most people (in the area) like best. The term “pop song” indicates an entire music industry that is chasing the mainstream listeners.

3.3.1 Proximity of Terms

- Assume we are search with “White House” and we have the following documents:
 - “The white car stands in front of the house”
 - “The president entered the White House”
- Which one would you expect to match the query better?
- Brin and Page, with the example of Bill Clinton, realized that most people implicitly assume proximity between query terms in the documents. Especially, with the most popular search type (celebrities, brands, names), the implicit proximity assumption is key. If you are looking for “Bill Clinton”, you do not want to find:
 - “....Bill Rosweld was winning....and John Clinton raised his hand...”
 - “...the dollar bill was on the floor ... Mrs. Clinton went home...”
- The average user is expecting that the query terms are next to each other (or at least very close) and that the order is as given in the query. Try it yourself:
 - “White House” → returns the official homepage for the White House
 - “House White” → returns another page at the top with the name “House White”
- To enable a similarity value based on proximity, Google uses two options:
 - n-grams: add “white house” as a new n-gram term and use it for searches. This ensures that hits have both words in proximity
 - extract position information from the document, calculate proximity for terms in the document, and push similarity values if proximities are high

- With the position information, we can evaluate a simple metric for proximity. The following is a rough sketch of what Google's early search engines did, but still applies in one or the other way in today's version. The basic idea is to store not only term frequencies in the inverted lists but the positions of occurrences in the documents (so-called hit-lists). For example: consider the query "White House".
 - We read the hit lists for each of the terms and a given document from the inverted file:

```
hitlist['white'] = [1, 13, 81, 109, 156, 195]
hitlist['house'] = [2, 82, 112, 157, 189, 226]
```

The hit lists are then combined pairwise to obtain all interesting combinations. This leads to the following pairs that bear some proximity information between "white" and "house"

```
pairs = [(1,2), (81,82), (109, 112), (156, 157), (189,195)]
```

- Proximity is expressed as the distance between the elements of pairs and is quantized to a small set proximity values (in this example we use values between 1 and 10):

```
proximity = [1,1,3,1,6]
```

In this simplified approach, '1' denotes adjacent terms, '3' close-by, and '7' distant. Any quantization is thinkable at this point as long as it matches user expectations. Based on these proximity values, a histogram is built, i.e., counting how often a proximity values occurs.

```
pbins = [3,0,1,0,0,1,0,0,0,0]
```

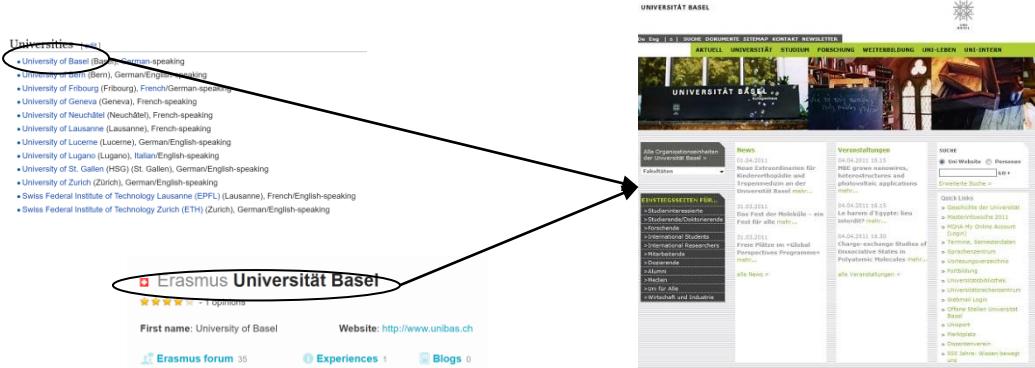
- Finally, the bins are weighted and summed up to a proximity score:

```
weights = [89,55,34,21,13,8,5,3,2,1]
score_proximity = Σi pbins[i] * weights[i]
```

3.3.2 Term Frequencies and HTML Attributes

- Classical retrieval was simply counting term frequencies regardless of where they occur in the text. For HTML documents, we may want to take the surrounding tags into account and add more weight to occurrences if the term is part of `<title>`, `<h1>`, `` or `<i>`.
- Brin and Page went a step further: they realized in their research that hyperlinks not only describe the document containing the anchor text but also provide additional keywords for the referenced web pages. Even more, anchor texts tend to be short and concise and so we obtain very relevant keywords that describe the referenced document most accurately. Nobody is writing a hyperlink with a lengthy anchor text about what is expected at the referenced location. On the other side, we frequently encounter anchor texts with low relevance like “click here”, “back to start” or “follow me”.
- The very first search engines were plagued by spammers: the authors wrote documents containing numerous key words, sometimes repeated 1'000 times, to push their web pages to the top of the search results. Google stopped these spamming approaches by firstly weighting terms in (external) anchor texts much more (what others say about you), and secondly ceiling the number of occurrences at a low number. In contrast to the classical vector space retrieval, Google was not ranking based on term frequencies and idf-weights but used a more elaborated scheme to assess the significance of a term to describe the content.
- Again, we only know what Brin and Page did as part of their research. Meanwhile, Google has extended its scheme to describe documents better and to prevent spammers, click baits, and other dubious pages to appear at the top of searches. Their original approach had 3 steps:
 - Describe the document with the key words and their tags
 - Add keywords of anchor texts to the referenced document
 - When creating the index, sum up the weighted occurrences to a single term score

- Consider a web page and a term “university”. We extract all the term occurrences and their surrounding tags, and associate a term occurrences whenever an anchor text contains “university” and points to the page:



```

...<title> ... university ...</title>
...<h1> ... university ...</h1>
...<b> ... university ...</b>
...<p> ... university ...</p>
...<td> ... university ...</td>
...<i> ... university ...</i>
...<h1> ... university ...</h1>
...<b> ... university ...</b>
...<h1> ... university ...</h1>

```

- Terms are extracted as usual but we keep `<tag>`-information and count how often a term-tag pair occurs (if multiple tags are active, for each tag a separate pair is added). Whenever we encounter a hyper link to our web page, we add the terms of the anchor text:

```

terms = [...(university, <title>,1),...(university, <h1>,2),...(university, <b>,10),
...(university, <p>,55),...(university, <td>, 2),...(university, link, 23)]

```

- Upon creating the index, a final score for a term is computed using an upper limit (e.g. 100) for the occurrences and weights depending on the tag:

```

weights[tag → weight] = [<title> → 13, <h1> → 5, <p> → 1, link → 55]
score[university] = Σterms[i,1]=university min(100,terms[i,3]) * weights[terms[i,2]]

```

- Interestingly, we can now search for documents we have never seen (but only heard about). The scores of the query terms are added up for the ranking (together with all other scores).

3.3.3 PageRank

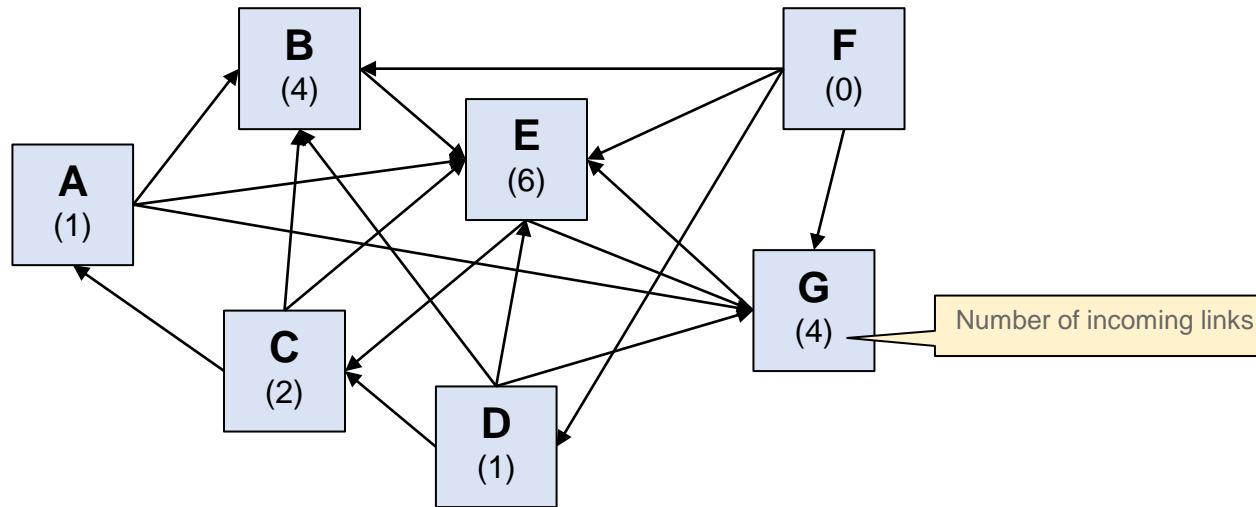
- Assume we search with the key words “uni basel”. What would you expect to be at the top of the ranking? The two pages below both qualify as they have the keywords in prominent places.
 - As a student of this course, you are obviously visiting the course page more often and hence it is more important than the home page of uni basel.
 - However, the average student (or the average web surfer) is more interested in the home page than in a single course.
 - Looking only at key words is not sufficient. Yes, we can take hyperlinks into account and consider the page with most matching keywords in hyperlinks as being more relevant. But that would not help with very popular brands like Apple and Google and with 1000s of websites discussing various aspects of the brands (and hence competing for the top spots in searches).

The screenshot shows a course page for 'Vorlesung: Multimedia Retrieval'. At the top, there is a navigation bar with links for 'Studium/Education', 'Forschung/Research', 'Lehre/Teaching', 'Team', 'Kolloquien', and 'FAQ'. The main content area has a heading 'Vorlesung: Multimedia Retrieval'. Below this, there is a table with course details:

| | |
|----------------------|--|
| VV-Nr. | 15731-01 |
| Dozierende | Roger Weber |
| Zeit und Ort | Mi 08.15 - 12.00: Seminarraum 05 001, Spiegelgasse 5 |
| Start | 20.09.2017 |
| Voraussetzungen | Basics of programming |
| Inhalte | Introduction to information retrieval with a focus on text documents and images. Main topics comprise extraction of characteristic features from documents, index structures, retrieval models, search algorithms, benchmarking, and feedback mechanisms. Searching the web, images and XML collections demonstrate recent applications of information retrieval and their implementation. |
| Leistungsüberprüfung | Lehrveranst.-begleitend Bitte beachten: Oral exam Tuesday, 9 January 2018 (3-6 p.m.) and Tuesday, 16 January (3-6 p.m.) Room: 00 003, Spiegelgasse 1 |
| Kreditpunkte | 6 |
| Skala | 1-6 0.5 |
| Module | Wahlbereich Master Informatik: Empfehlungen (Master Informatik 10) Modul Praxis aktueller Informatikmethoden (MSF - Informatik (Studienbeginn vor 01.08.2016)) Modul Applications of Distributed Systems (Master Computer Science 16) Modul Applications of Machine Intelligence (Master Computer Science 16) Modul Concepts of Distributed Systems (MSF - Computer Science) |
| Belegen | Services (Anmeldung mit Passwort) |

The screenshot shows the University of Basel homepage. At the top, there is a navigation bar with links for 'English', 'Organisationseinheiten', 'Dokumente', 'Informationen...', 'Standorte', 'Suche', and 'Personen'. The main content area features a large image of Michael Hall holding a trophy, with the text 'Lasker-Preis Grosser Empfang für Michael Hall'. Below the image, there are navigation links for 'Aktuell', 'Studium', 'Forschung', 'Innovation', 'Weiterbildung', and 'Universität'. At the bottom, there are video control buttons for a video player.

- PageRank was invented by Larry Page (one of the Google founders) during his research time at Stanford. His preliminary idea was to consider a page more relevant (for the average user) if it has many incoming links. Consider the following example:



- PageRank assigns an absolute ranking to all pages in a graph like the one above. A naïve approach considers only the incoming links. In the running example, E would be top ranked as it has the largest number (6) of incoming links. B and G follow with 4 links, then C with 2 links, D with 1 link, and finally F with no incoming links. This also would reflect our intuitive understanding of importance as E appears to be indeed the center of the graph.
- Consider B and G: both have 4 incoming links and hence tie on 2nd place. If we look closer, we see that G is referenced by E while B is referenced by less important pages. In other words, simply considering incoming links is not enough, we want to also weight the link based on the quality of the source.
- Note that incoming links as a ranking measure is not very robust. A web page author can easily create thousands of incoming links and thus optimize the ranking of the page (link farms).

- PageRank is based on a random surfer model: a user navigates through the web by clicking on links. At some point, the user switches randomly to another page (e.g., picked from the bookmarks). We make the following assumptions for the random surfer model:
 - When on a page, the user can perform two actions: 1) with a probability α he follows a link, and 2) with a probability $1 - \alpha$ he enters a random URL (from his or her bookmarks, or by search)
 - For 1) user navigates by picking a random link (all links are equally probable)
 - For 2) if a page has no outbound links (sink), the user picks a random URL
- We can interpret PageRank as a Markov chain in which the states are pages and the transitions are the links between pages. The PageRank of a page is then the probability that a random surfer is visiting that page. The higher the value, the more popular the page and we expect it to be more often at the top of the search results.
 - To compute the probability equations, we consider two aspects: 1) all incoming links, and 2) a random switch to the page. Let $q \rightarrow p$ denote that q contains a link to p , and let $L(q)$ be the number of outgoing links from q . The set of all pages \mathbb{P} contains $N = |\mathbb{P}|$ elements. We can then express the PageRank with the given probability α that the user follows a link as:

$$PR(p) = \frac{1 - \alpha}{N} + \alpha \cdot \sum_{q \rightarrow p} \frac{PR(q)}{L(q)} \quad \forall p \in \mathbb{P}$$

- Interpretation: to obtain a high PageRank, the number of incoming links is still important but each incoming link is weighted by the PageRank (aka importance) of the source page. If a page has several outgoing links, its PageRank is evenly distributed to all referenced pages. The method is more robust and adding artificial links does not help to push the PageRank. On the other hand, it favors older pages that are well connected while new pages, even very good ones, lack the number of links necessary to obtain high ranks.

- Evaluation: the PageRank equation defines an implicit equation system that can be solved iteratively. Let $\mathbf{r} \in \mathbb{R}^N$ be the vector holding all PageRanks of documents in \mathbb{P} . We represent the links between pages with a matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$:

$$M_{i,j} = \begin{cases} \frac{1}{L(p_j)} & \text{if } p_j \rightarrow p_i \\ \frac{1}{N} & \text{if } p_j \text{ has no outgoing links} \\ 0 & \text{otherwise} \end{cases}$$

With this, we can rewrite the PageRank equation as follows:

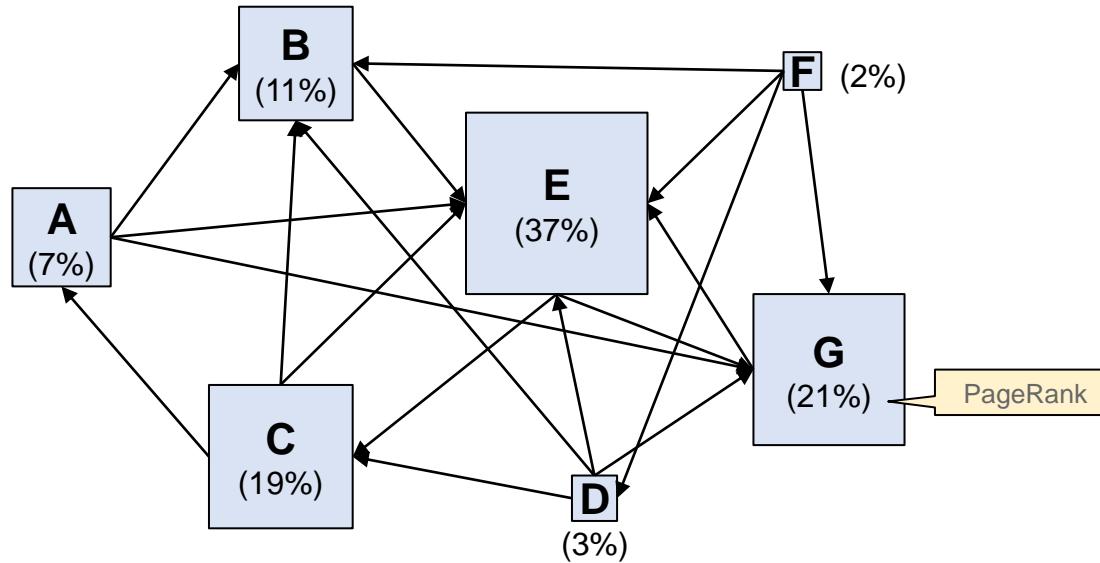
$$\mathbf{r} = \frac{1 - \alpha}{N} \cdot \mathbf{1} + \alpha \cdot \mathbf{M}\mathbf{r} \quad \text{with } \mathbf{1} \text{ being a column vector of length } N \text{ with only ones}$$

We now can describe the iterative process to solve the above equation system. Let $\mathbf{r}^{(t)}$ denote the PageRank values of pages after the t -th iteration:

- Initialization: $\mathbf{r}^{(0)} = \frac{1}{N} \cdot \mathbf{1}$, $\alpha = 0.85$
- Iteration:
 - $\mathbf{r}^{(t+1)} = \frac{1 - \alpha}{N} \cdot \mathbf{1} + \alpha \cdot \mathbf{M}\mathbf{r}^{(t)}$
 - stop if $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}| < \epsilon$

Because of the sparse link matrix, the iteration converges rather quickly and it can easily scale to larger document sets. In their original study, Brin and Page reported 52 iterations of a network with 322 millions of links, and 45 iterations for 161 millions of links. They concluded that the number of iterations is linear to $\log(n)$ with n being the number of edges. Due to the sparse matrix, compressed representations are used to minimize memory consumption.

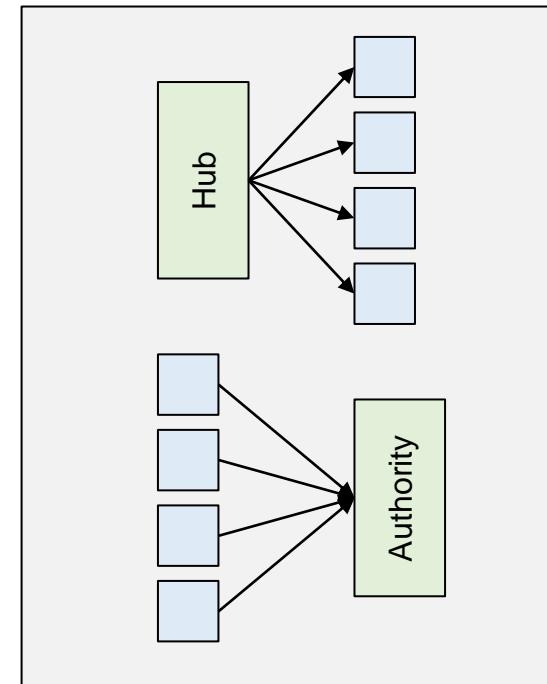
- Example from before: let us apply the PageRank formula to the graph below. The size of the nodes represent now the PageRank and the values the probabilities that a random surfer visits the page:



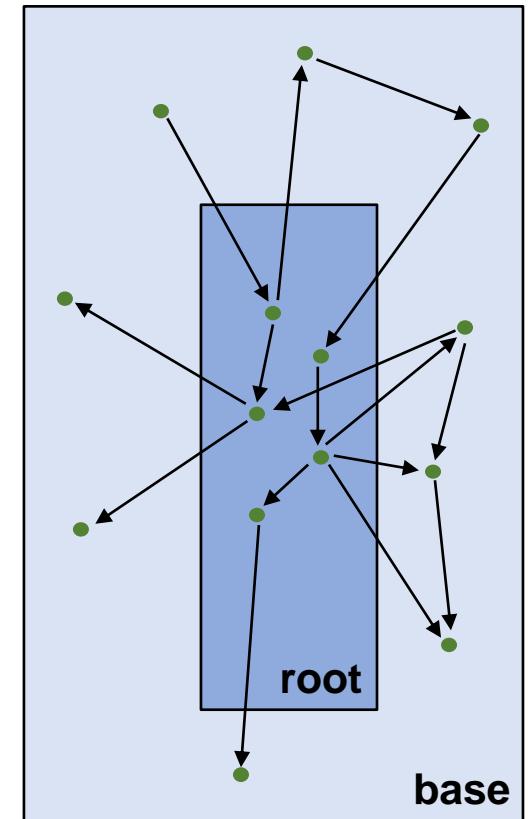
- Discussion: E is still the center of the network but G and C are now more important than B. Even though B has 4 incoming links, two of them come from the least important pages D and F. On the other side, E has only two outgoing links and hence both C and G receive about 43% (with $d = 0.85$) of the PageRank of E.
- Usage for ranking: the PageRank is an absolute measure for the importance of a page regardless of the query. It is computed once after a complete crawl and used for all queries. Even though PageRank is an important measure, it is only one of many criteria. If we would emphasize too much on PageRank, we would only see the same sites in our search results. Terms and proximity are equally important but PageRank helps to favor pages that are more likely visited by users (and hence requested in the search results to be at the top). However, negative publicity pushes pages to the top as well.

3.3.4 Hyperlink-Induced Topic Search (HITS)

- There are many ways to interpret link information during feature extraction. A common observation is that there are two prototypes of web pages
 - **Authorities** are web pages that discuss a topic and are recognized by the community as the authoritative source for information. A good example is Wikipedia, IMBD, MusicBrainz, etc.
 - **Hubs** are web pages that group authorities in a directory like style without actually discussing the topics. Your bookmarks are your personal hub, but also web sites like Yahoo, Yellow Pages, etc.
- Note that PageRank was only considering how likely a user would visit the page but not whether it contains any authoritative content. PageRank is also a random walk across the entire web. The methods we consider in this section only look at the current topic of the query, hence the terms **Topic Search** is often used with these methods.
- How can we recognize a good hub and a good authority?
 - A hub is a web page with many links to authorities. We observe the typical hub structure depicted on the right side
 - An authority is web page with many incoming links from hubs. We observe a typical (sink) structure as depicted on the right side
 - To be a good hub, it must link to good authorities on the topic. To be a good authority, it must be linked by many good hubs on the topic.
 - Note: “on the topic” means that we are not just interested in the number of incoming / outgoing links, but they have to be related with the current topic. This is the biggest difference to PageRank where all links regardless of any topic are considered.



- Jon Kleinberg developed the HITS algorithm in 1997. He observed the concepts of hubs and authorities in the emerging web where directories were the pre-dominant way to find information on the Internet (search engines existed but lacked the sufficient quality). To better guide searches, he introduced two metrics for a web page p :
 - $h(p)$ denotes the hub value of the page p , i.e., its ability to serve as a hub for the user
 - $a(p)$ denotes the authority value of page p , i.e., its ability to provide content to the user
- As we are only interested in a single topic, not the entire web structure is used. Instead, we create a base set with the following two steps:
 1. For a query / topic Q determine the top results with the help of a search engine. This set is called the root set. It already contains a fair number of hubs and authorities, but not yet all relevant ones
 2. Extend the root set with a) web pages that link to a page in the root set, and b) pages that are referenced by a page in the root set. We can remove links within the same domain (navigation links) and can limit the number of incoming / outgoing links to keep the graph small. This set is called the base set
- In practice, we need to execute several searches and downloads to compute the base set. 2b) requires downloading the pages of the root set, extracting link information, and adding the referenced pages. Step 2a) requires a search with a `link:-`-clause to obtain pages that link to a member of the root set. A previous crawl of the web to obtain the link structure greatly reduces the costs for constructing the base set.



- We use the notation $p \rightarrow q$ to denote that p contains a link to q . We now can formulate the HITS algorithm as an iterative process. Assume that the base set \mathbb{P} contains N pages:

1. Initialization: $h^{(0)}(p) = a^{(0)}(p) = \sqrt{1/N} \quad \forall p \in \mathbb{P}$

2. Iteration:

- Update: $a^{(t+1)}(p) = \sum_{q \rightarrow p} h^{(t)}(q) \quad h^{(t+1)}(p) = \sum_{p \rightarrow q} a^{(t)}(q)$

- Normalize $a(p)$ and $h(p)$ such that: $\sum_p a^{(t+1)}(p)^2 = \sum_p h^{(t+1)}(p)^2 = 1$

- Stop if $\sum_p |a^{(t+1)}(p) - a^{(t)}(p)| + \sum_p |h^{(t+1)}(p) - h^{(t)}(p)| < \epsilon$

- Once computed, we can return the top hubs (highest $h(p)$ values) and the top authorities (highest $a(p)$ values) to the searcher.
- We can rewrite the equations in matrix notation. Let \mathbf{h} be the vector of hub values for all pages, and let \mathbf{a} be the vector of authority values. We can construct the adjacency matrix \mathbf{A} from the graph:

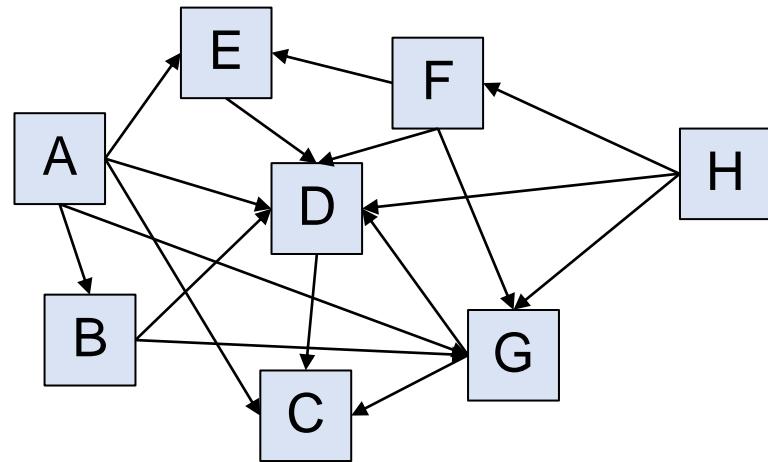
$$A_{i,j} = \begin{cases} 1 & \text{if } p_i \rightarrow p_j \\ 0 & \text{otherwise} \end{cases}$$

The rows of \mathbf{A} contain all outgoing links while the columns contain all incoming links. With this the computational scheme for the iteration becomes

$$\mathbf{h}^{(t+1)} = \mathbf{A}\mathbf{a}^{(t)}$$

$$\mathbf{a}^{(t+1)} = \mathbf{A}^T \mathbf{h}^{(t)}$$

- Example: consider the following graph



Adjacency Matrix

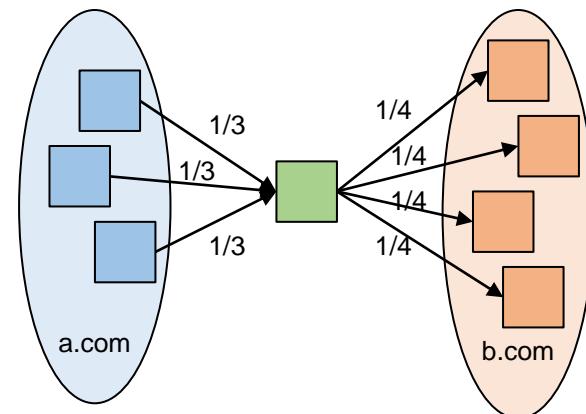
| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 1 | 1 | 1 | | | 1 | |
| B | | | | 1 | | | 1 | |
| C | | | | | | | | |
| D | | | | 1 | | | | |
| E | | | | | 1 | | | |
| F | | | | | | 1 | 1 | 1 |
| G | | | 1 | 1 | | | | |
| H | | | | | 1 | | 1 | 1 |

- We observe that A is the best hub. It links to the best authorities D, G, and E. E is a slightly better authority than C despite having only 2 (compared to 3) incoming links. But it is referenced by the good hubs A and F, while C is referenced by the good hub A, the average hub G and the bad hub D.
- Note that it's not always clear whether a page is a hub or an authority. B for instance is a bad authority and an average hub. C is not in the top authorities and a bad hub. E is a top authority but also has some hub value.
- Finally, remember that we are only looking at the base set of nodes, that is, only at pages that are somewhat related to the topic. Hence, in contrast to PageRank, the hub and authority values of pages change with different queries / topics.

3.3.5 Extensions of HITS (Henzinger, 1998)

- The HITS algorithm suffers from three fundamental problems:
 1. If all pages in a domain reference the same external page, that page becomes too strong an authority. Similarly, if a page links to many different pages in the same domain, that page becomes too strong a hub.
 2. Automatically established links, e.g., advertisements or links to the provider/host/designer of a web site, provide the linked sites a too high authority value (even though they are off topic)
 3. Queries such as "jaguar car" tend to favor the more frequent term (here "car") over the less frequent term. The idea of the query, however, was to distinguish from the animal.
- The first improvement addresses domains. Instead of every page having a single vote on the authority (hub) of an external page, the entire domain gets a single vote.
 - Assume that k pages q_i in a domain link a page p , then we weigh the hub values in the authority formula for page p with $aw(q_i, p) = \frac{1}{k}$.
 - Similarly, assume that a page p links to l pages q_i in the same domain, then we weigh the authority values in the hub formula for page p with $hw(p, q_i) = \frac{1}{l}$.
 - With these weights, we adjust the iteration of HITS as follows:

$$a^{(t+1)}(p) = \sum_{q \rightarrow p} aw(q, p) \cdot h^{(t)}(q) \quad h^{(t+1)}(p) = \sum_{p \rightarrow q} hw(p, q) \cdot a^{(t)}(q)$$



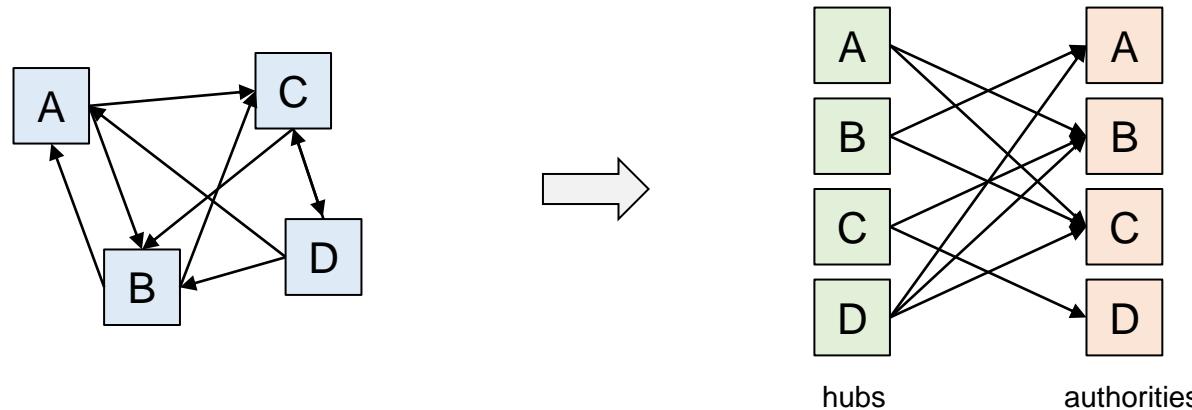
- The second improvement focuses on the topic and applies penalties for pages that are not following the topic. This helps to sort out advertisements. To push less frequent terms, and $tf * idf$ scheme is chosen similar to the methods in vector space retrieval.
 - We construct a reference document C from all documents in the root set (e.g., taking from each document the terms with highest $tf * idf$ values)
 - Compute a similarity value $s(p)$ for page p using the $tf * idf$ vectors of p and the reference document C , i.e., $s(p) = \frac{c^T p}{\|c\| \cdot \|p\|}$
 - For a given threshold t , eliminate all pages p with $s(p) < t$ from the base set. To get a good threshold, we can use the median of all $s(p)$ values, i.e., eliminate 50% from the base set.
 - Use the similarity values $s(p)$ to adjust how much authority and hub value is passed to a page. We adjust the iteration of the HITS algorithm as follows:

$$a^{(t+1)}(p) = \sum_{q \rightarrow p} aw(q, p) \cdot s(q) \cdot h^{(t)}(q) \quad h^{(t+1)}(p) = \sum_{p \rightarrow q} hw(p, q) \cdot s(q) \cdot a^{(t)}(q)$$

- This extension has resulted in a 45% improvement over the original HITS algorithm.

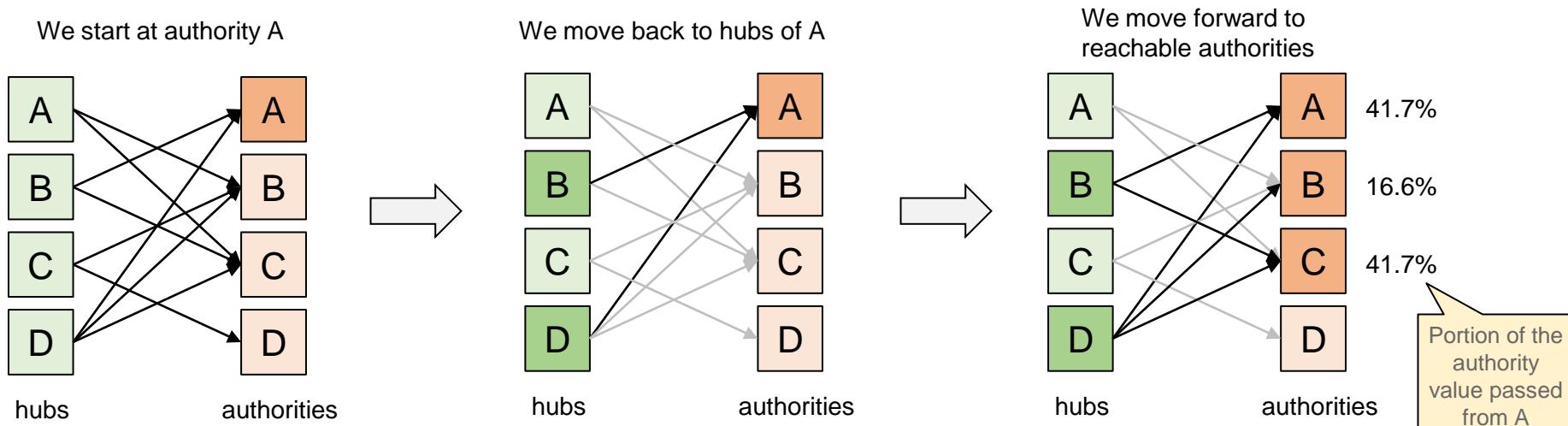
3.3.6 SALSA Algorithm

- The Stochastic Approach for Link Structure Analysis – SALSA is a further extension of the HITS algorithm. Similar to PageRank, it considers the transitions from one page to the other and models it with a Markov chain. However, it only considers two steps in the network, and not an infinite walk across the entire web. Similar to HITS, it only considers a base set of pages obtained with the same approach as with HITS and given a query / topic.
- SALSA considers the graph of the base set as a bipartite graph with pages having a double identity, once as a hub identity and once as a authority identity.

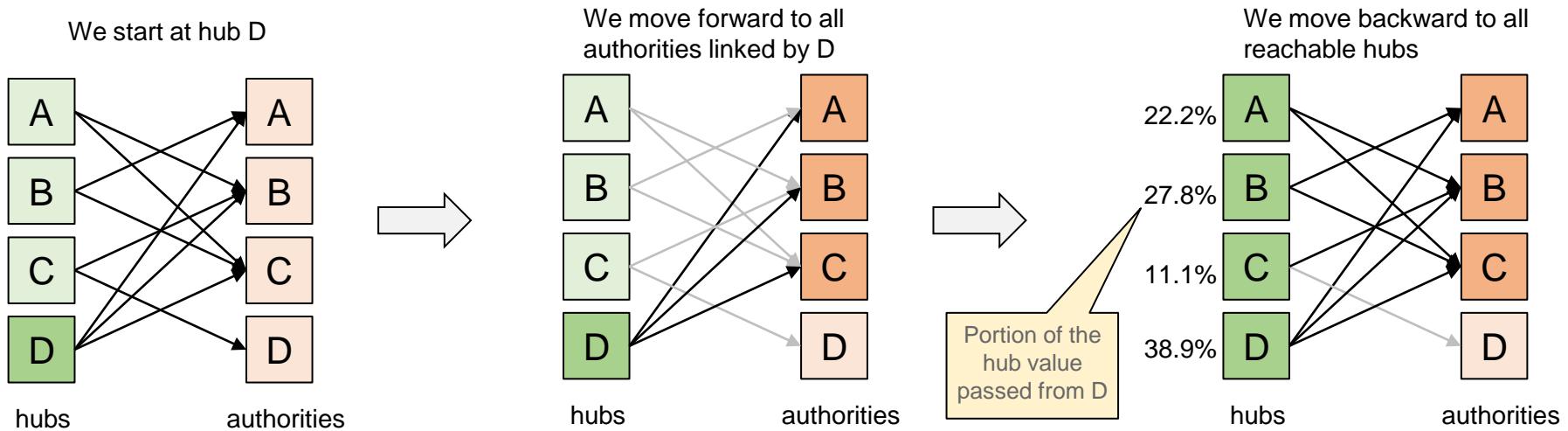


- To compute authority values, the algorithm is performing a random walk with two steps. Starting from a page p , it goes backward to all hubs that link to p and then walks forward to all pages reachable from these hubs. To determine how much authority value is passed from page p to a such reachable page q , we consider a random walk with two steps starting at p and use the probability of arriving at q as the fraction of authority passed from p to q .
- In contrast to HITS, authority values only depend on the authority value of other reachable authorities but not on the hub values.

- Example: consider the example before. We want to compute how much of the authority value of A is passed to C. We now consider the steps of the random walk starting at A:
 - We first walk backwards to all hubs that link to A: there are two hubs B and D each with a 50% chance for the random walk to select.
 - Walking forward: 1) From B, there are two links to A and C, again each path with a 50% chance to be taken. We note a first path from A to C with a 25% chance to be taken. 2) From D, there are three links to A, B and C, each path with a 33.3% chance to be taken. We note a second path from A to C with a 16.7% chance to be taken
 - Summing up, the two paths yield a 41.7% chance to get from A to C. This is the portion of the authority value of A passed to the authority value of C.



- Similarly, we can compute hub values. But this time, the random walk is first forward to all authorities linked by the starting hub, and then backwards to all hubs linking these authorities. The probability of reaching hub q from a hub p determines how much hub value is passed from p to q .
- Example: consider the same example as before. We want to compute how much of the hub value of D is passed to B. We now consider the steps of the random walk starting at D:
 - We first walk forwards to all authorities linked by D: there are three authorities A, B and C each with a 33% chance for the random walk to select.
 - Walking backwards: 1) The two hubs B and D link to A, each hub selected with 50% probability. We note a first path from D to B with a 16.7% chance to be taken. 2) The three hubs A, C and D link to B, each hub selected with 33% probability. There is no path to B. 3) The three hubs A, B and D link to C, each hub selected with 33% probability. We note a second path from D to B with a 11.1% chance to be taken.
 - Summing up, the two paths yield a 27.8% chance to get from D to B. This is the portion of the hub value of D passed to the hub value of B.



- More formally, we can compute hub and authority values as follows. Let \mathbf{A} be the authority-matrix and \mathbf{H} be the hub-matrix. Further, let $L_{in}(p)$ denote the number of incoming links to page p , and $L_{out}(p)$ denote the number of outgoing links of page p . We can determine the matrix elements with the two steps as described before as follows:

$$A_{j,i} = \sum_{q: q \rightarrow p_i \wedge q \rightarrow p_j} \frac{1}{L_{in}(p_i)} \cdot \frac{1}{L_{out}(q)}$$

$$H_{j,i} = \sum_{q: p_i \rightarrow q \wedge p_j \rightarrow q} \frac{1}{L_{out}(p_i)} \cdot \frac{1}{L_{in}(q)}$$

- We now can compute the hub and authority value using an iterative approach. Again, \mathbb{P} denotes the set of all pages in the base set with $N = |\mathbb{P}|$ being the number of pages.
 - Initialization: $h_i^{(0)} = a_i^{(0)} = 1/N \quad \forall i: 1 \leq i \leq N$
 - Iteration:
 - $\mathbf{a}^{(t+1)} = \mathbf{Aa}^{(t)}$
 - $\mathbf{h}^{(t+1)} = \mathbf{Hh}^{(t)}$
 - stop if $\|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}\| + \|\mathbf{h}^{(t+1)} - \mathbf{h}^{(t)}\| < \epsilon$
- A variant of the SALSA algorithm is used at Twitter to recommend “whom to follow”. The twitter example is a very good fit as the graph is usually uni-directional (you follow someone but that person is not necessarily following you).

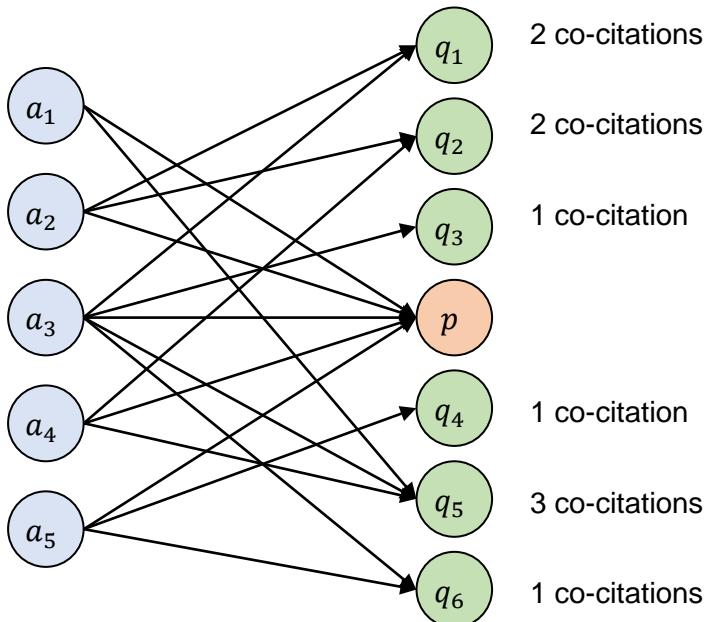
3.3.7 Co-citations and Similar Pages

- The basic idea of **Alexa's „What's Related“** was to identify similar or related documents for a given document. As an example, if you are on the homepage of a car manufacturer (e.g., Ford Motor), a related page would be the one of another car manufacturer (e.g., Honda, GM, VW, Nissan). How can we compute such related pages from the web graph?
 - Surf History/Bookmarks Analysis: users often browse and keep pages on similar topics together. If you want to buy a car, you check several similar pages in succession
 - Co-citations: if two pages are often linked together by pages, we can assume some relationship
 - Deduce relationships from link structure and an implicit similarity score definition (similar pages are linked by similar pages)
- Alexa Toolbars observe surfers, record the history, and performs a static analysis on the surf behavior of users to compute various things such as the Alexa Ranking and suggestions for similar sites. The analysis follows typical data mining approaches for affinity analysis. It is not taking the link structure of the web into account.
- In the following, we look at the other two methods in more details:

- Co-Citations consider only the web graph and count how often two pages are linked together. The first publication by Dean and Henzinger in 1999 suggested the following simple algorithm. We start with a page p and are looking for related pages q_i :

1. Determine at most k parent pages a_j of starting page p
2. Extract for each parent page a_j at most l links to pages q_i that are in the proximity of the link to p
3. Count how often a page q_i is obtained by step 2
4. If we found less than 15 pages q_i with at least 2 co-citations with p then reduce URL of p and start again.
5. Related pages q_i to p are the ones with most co-citations

- Note that not all links are extracted from parent pages a_i but only the ones that appear close to the starting page p . “Close” means the l links which are nearest to the link to p in the HTML file.
- The figure on the right hand shows a simple example with a starting page p , its parent pages a_1, \dots, a_5 and their linked pages q_1, \dots, q_6 . In this example, we find q_5 as the most co-cited page to p .



- Another co-citation method by Dean and Henzinger was published in 1999: the **companion algorithm**. It is a more complex variant of co-citation using similar techniques as with HITS. We start again with a page p and look for related pages q_i :

1. Build a neighborhood graph around page p as follows:
 - add starting page p to graph
 - add at most k parent pages a_i that link to p , and for each page a_i , add at most l links to child pages (around the link to p)
 - add at most m child pages c_j linked by p , and for each page c_j , add at most n parent pages that have a link to c_j
 - add edges between nodes based on the links in the web
2. Merge duplicates and near-duplicates
 - Two documents are near-duplicates if they contain more than 10 links and 95% of their links are the same (occur in both documents)
3. Assign weights to edges in graph based on domain linked
 - Assume that k pages q_i in a domain link a page p , then we weight the edges with $aw(q_i, p) = \frac{1}{k}$
 - Assume that a page p links to k pages q_i in a domain, then we weight edges with $hw(p, q_i) = \frac{1}{l}$
4. Compute hub and authority values for all nodes in the graph with the following iteration

$$a^{(t+1)}(p) = \sum_{q \rightarrow p} aw(q, p) \cdot h^{(t)}(q) \quad h^{(t+1)}(p) = \sum_{p \rightarrow q} hw(p, q) \cdot a^{(t)}(q)$$

5. The pages q_i with the highest authority values are the related pages to p .

- **SimRank** is a method that defines a similarity score $\sigma(p, q)$ between two pages p and q in an implicit way: p and q are similar if they are linked by similar pages. Also, a page p is maximal similar with itself, i.e., $\sigma(p, p) = 1$. Given a “neighborhood” graph around p (use any method to construct such a neighborhood), we select those pages q_i with highest scores $\sigma(p, q_i)$.
 - Let $L_{in}(p)$ denote the number of incoming links to page p . Similarity is defined as follows:

$$\sigma(p, q) = \begin{cases} 1 & \text{if } p = q \\ \frac{C}{L_{in}(p) \cdot L_{in}(q)} \cdot \sum_{a \rightarrow p} \sum_{b \rightarrow q} \sigma(a, b) & \text{otherwise} \end{cases}$$

with C a decay factor, for instance $C = 0.8$.

- We can compute the similarity values with a simple iterative approach for all pages $p \in \mathbb{P}$:

1. Initialization: $\sigma^{(0)}(p, q) = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases} \quad \forall p, q \in \mathbb{P}$

2. Iteration:

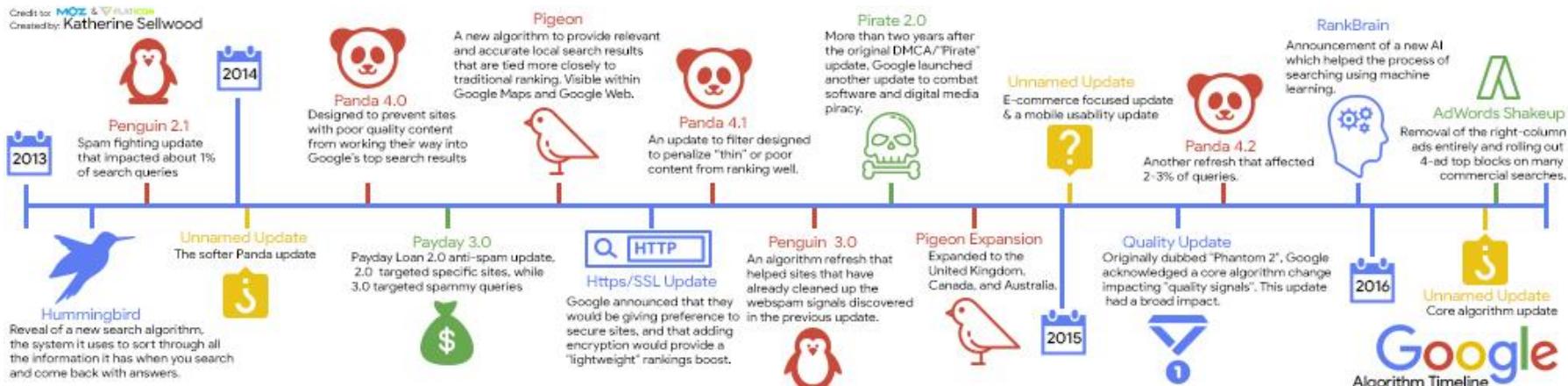
- Update: $\sigma^{(t+1)}(p, q) = \begin{cases} 1 & \text{if } p = q \\ \frac{C}{L_{in}(p) \cdot L_{in}(q)} \cdot \sum_{a \rightarrow p} \sum_{b \rightarrow q} \sigma^{(t)}(a, b) & \text{otherwise} \end{cases}$

- Stop if $\sum_{p,q} |\sigma^{(t+1)}(p, q) - \sigma^{(t)}(p, q)| < \epsilon$

3. Return pages q_i with highest similarity $\sigma(p, q_i)$

3.3.8 Further Improvements (Example Google)

- Google is using more than 200 criteria to compute the top results. We consider here a few of the published criteria but much is hidden within the Google algorithms as a trade secret. Other search engines like Bing use similar approaches, but Google is the best documented search engine.
- Hummingbird (2013): a series of changes themed on conversational queries and synonyms
 - First major update since 2001 focusing on data retrieval, artificial intelligence, and how data is accessed and presented to users. Especially, the integration into an immersive mobile experience was key concept. Users would no longer provide keywords but ask entire questions. Rather than searching for keywords, Hummingbird takes the context of the sentence into account. It uses synonyms to find more relevant keywords.
 - Rewards content pages over click baits, link farms, and pages with lots of advertisements. The reward helps to find relevant content related to the user's intent.
 - Considers co-citations of web pages to boost popular pages in niche topics (where PageRank is limited). Also consider keywords around anchor text to describe the referenced page.
 - Keeps users longer on Google pages by presenting integrated result tables.



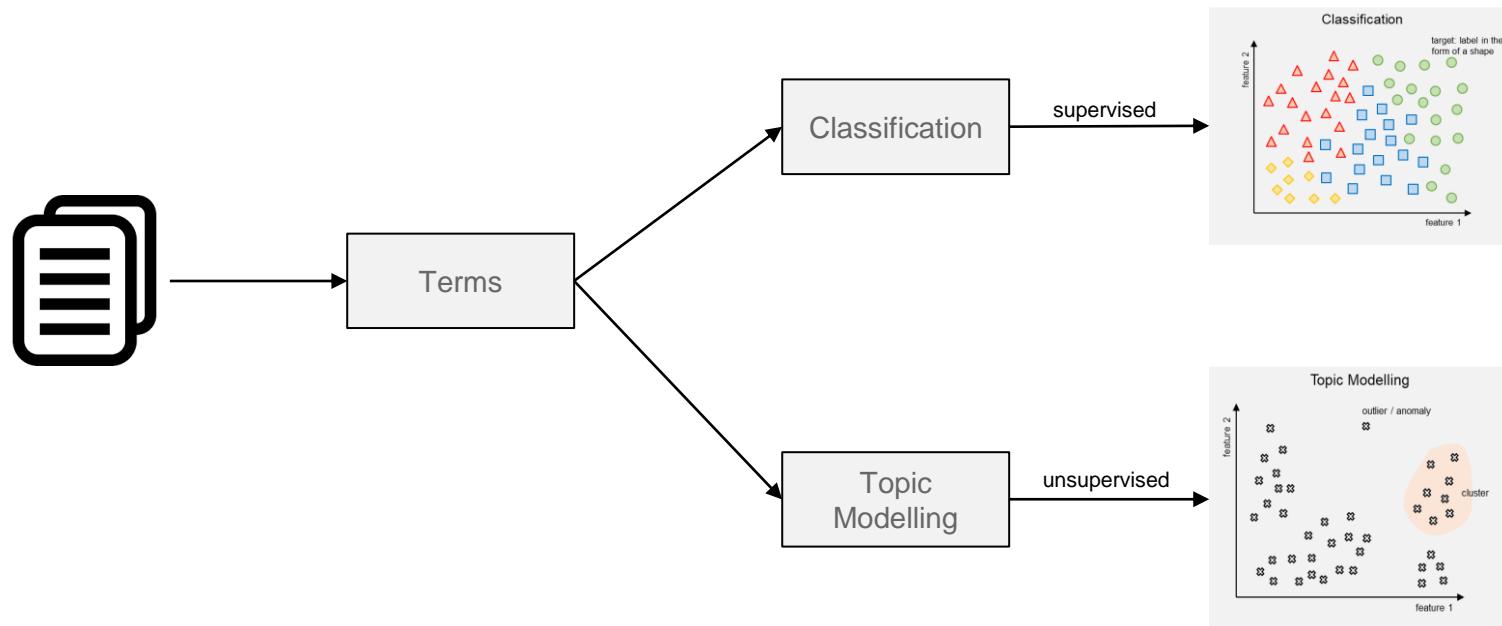
- Pigeon (2014): prefers local (to the user) search results taking location and distance to business into account. Not available everywhere but continuous roll-out to more countries.
- Penguin (2012): series of updates that penalize sites for not following rules (so-called black-hat search engine optimizations) using manipulative techniques to achieve high rankings.
 - Keyword spams, i.e., excessive use of some key words
 - Sites using link farms or other techniques to push PageRank
 - Doorway pages: built to attract search engine traffic but do not provide any content
 - Page Layout Algorithm: targets web sites with too many ads or too little content in the upper part of the page (above the fold)
 - Since 2012, 7 updates of Penguin were released. Affected a small percentage (<3%) of queries. Only recently added to the core search engine
- Panda (2011): updates to lower the rank of low-quality or thin sites, especially content farms. High quality pages should return higher in the rankings.
 - Thin content: web pages with very little relevant or substantial text
 - Duplicate content: content copied from other places. Initial versions had issues to correctly distinguish sources and scrapers replicating content only.
 - Lack of trustworthiness: sources that are not definitive or verified. To avoid impact, web sites should work to become an authority on the topic. Pages full of spelling and grammatical errors.
 - High ad to content ratio: pages with mostly ad and affiliate programs as content
 - Websites blocked by users
 - Panda affects the ranking of an entire site / section rather than an individual page. A penalty remains until the next update of Panda (not part of the core search). If a site has removed the dubious parts, the penalty is removed. Affected almost 12% of queries.

- Caffeine (2010): improved the entire index engine and turned the batch process into a more continuous update process providing up to 50 percent fresher content
 - Historically, Google crawled the web during 30 days, created the index, and then used this index for another 30 days. Soon, the engine was extended by the freshbot which captured news content and of important pages more frequently providing fresh index data to searches.
 - With Caffeine, the entire engine was overhauled. Instead of using several layers with web sites updated at different frequencies, the Caffeine update brought a continuous update process with it. The pages are not more frequently crawled than before, but the updates would become visible more quickly.
 - Internally, the engine was switched from the MapReduce algorithm to BigTable, Google's distributed scale-out database. Caffeine operates on a 100 PB database (!) and adds new information at a rate of a petabyte per day.
- Knowledge Graph (2012): a knowledgebase used to enhance semantic search results. Information is gathered from a wide range of sources
 - Collected from sources like the CIA World Factbook, Wikipedia, and similar sites.
 - Freebase (community managed content) was handed over into Wikidata
 - The newer Knowledge Vault uses artificial intelligence to derive data automatically from web content
 - As of 2016, the knowledge graphs holds 70 billion facts. There is an open Google API to programmatically access the database



3.4 Latent Semantic Analysis

- So far, we used manual methods to correlate terms with each other, e.g. with stemming or synonyms. In the following, we look at an automate way to extract clusters in the vocabulary, so-called topics. Topics help us to search with more flexibility (and fuzziness) without the need to think about alternative ways to ask the question. In addition, the method generate specific topics that are most relevant for the collection at hand. On the other side, we can not use the same topics in different collection; even more, we don't even know what the topics are (but we can guess them)!
- Topic modelling is not the same as text classification (see next section). The difference is that classification uses supervised learning: we have a set classifiers and learn how the features best match to the classifier. Topic modelling is unsupervised and tries to identify clusters/co-occurrences of terms in text documents:



- We will use clustering methods later in image retrieval. But with text retrieval, we face a number of challenges like high dimensionality (vocabulary size), sparciness of the term-document-matrix (almost everything is 0), and strong correlation between terms (run, runs, ran, walks, goes, went, ...). So the usual clustering approaches will not work out of the box.
- The idea of Latent Semantic Analysis (also called Latent Semantic Indexing) is to reduce the dimensionality of term vectors without altering the relations between documents and terms. Given a number of topics, it decomposes the term-document matrix with a singular value decomposition. This defines a mapping from the high-dimensional feature space defined by the vocabulary to a much lower dimensional feature space defined by topics. We use the same transformation for query vectors and compare them with the mapped document vectors similar to vector space retrieval.

3.4.1 Preliminary mathematical background

- For each eigenvalue λ and eigenvector x of a quadratic (n, n) -matrix A , it holds:

$$Ax = \lambda x$$

- Eigenvalues are determined by solving the equation $\det(A - \lambda I) = 0$. This is equivalent to finding the zeroes of a polynomial function of degree n . Note that the zeroes can be real, complex, and may occur several times. The corresponding eigenvectors are orthogonal to each other.
- A symmetric matrix A has real eigenvalues (no complex ones). Let r be the rank of A . We can write matrix A as the following product:

$$A = U \Lambda U^T$$

- Λ denotes an (r, r) -diagonal matrix with the eigenvalues on the diagonal; U is an (r, r) -matrix with columns that are orthonormal, i.e. $U^T U = I$.

- The singular value decomposition generalizes the eigenvalue decomposition for non-quadratic matrices Let \mathbf{A} be an (m, n) -matrix of rank r . There exists an (r, r) -diagonal matrix \mathbf{S} and an (m, r) -matrix \mathbf{U} and an (n, r) -matrix \mathbf{V} both with columns that are orthonormal. It holds:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

- It follows:

$$\mathbf{A}^T\mathbf{A} = (\mathbf{U}\mathbf{S}\mathbf{V}^T)^T(\mathbf{U}\mathbf{S}\mathbf{V}^T) = \mathbf{V}\mathbf{S}\mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{V}\mathbf{S}^2\mathbf{V}^T$$

and

$$\mathbf{A}\mathbf{A}^T = (\mathbf{U}\mathbf{S}\mathbf{V}^T)(\mathbf{U}\mathbf{S}\mathbf{V}^T)^T = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}\mathbf{U}^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T$$

That is: \mathbf{U} holds the eigenvectors of $\mathbf{A}\mathbf{A}^T$ in its columns and \mathbf{V} holds the eigenvectors of $\mathbf{A}^T\mathbf{A}$ in its columns.

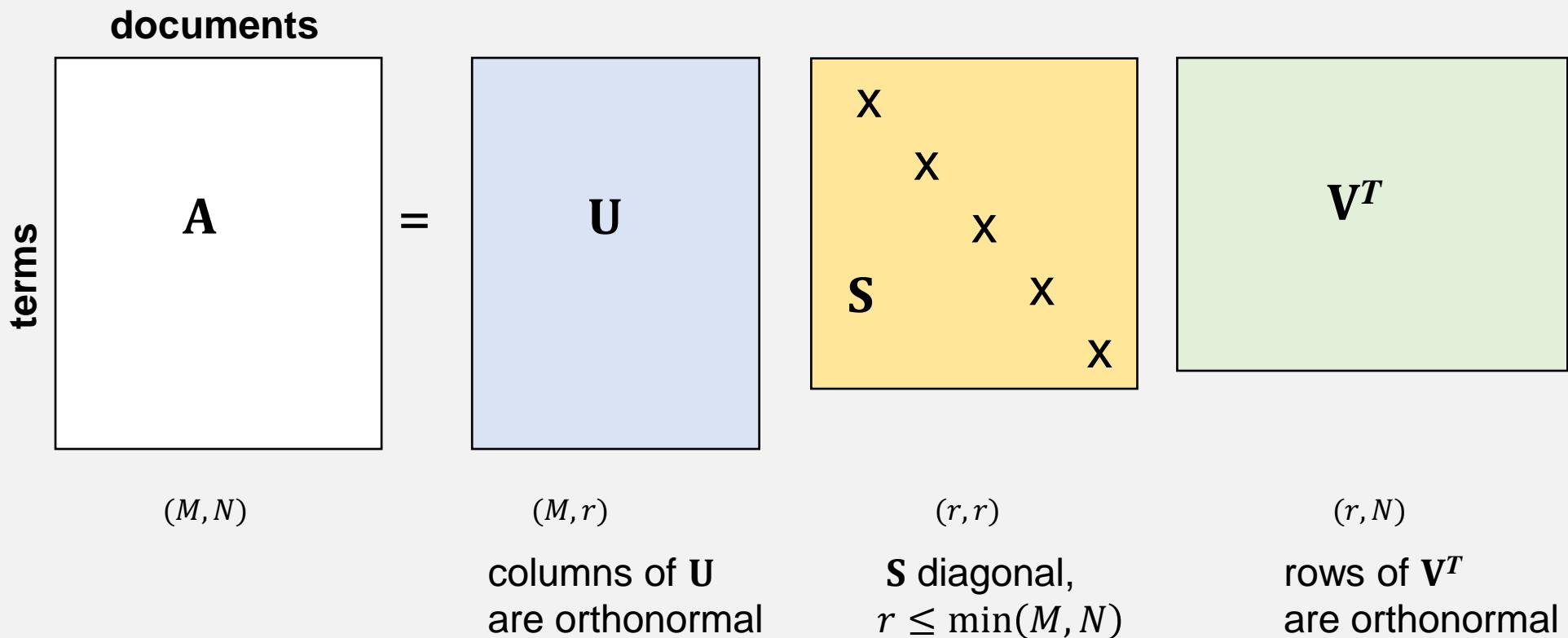
- We can re-write $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ as a sum of vector products (so called dyadic vector products):

$$\mathbf{A} = s_1(\mathbf{u}_1\mathbf{v}_1^T) + s_2(\mathbf{u}_2\mathbf{v}_2^T) + \dots + s_r(\mathbf{u}_r\mathbf{v}_r^T)$$

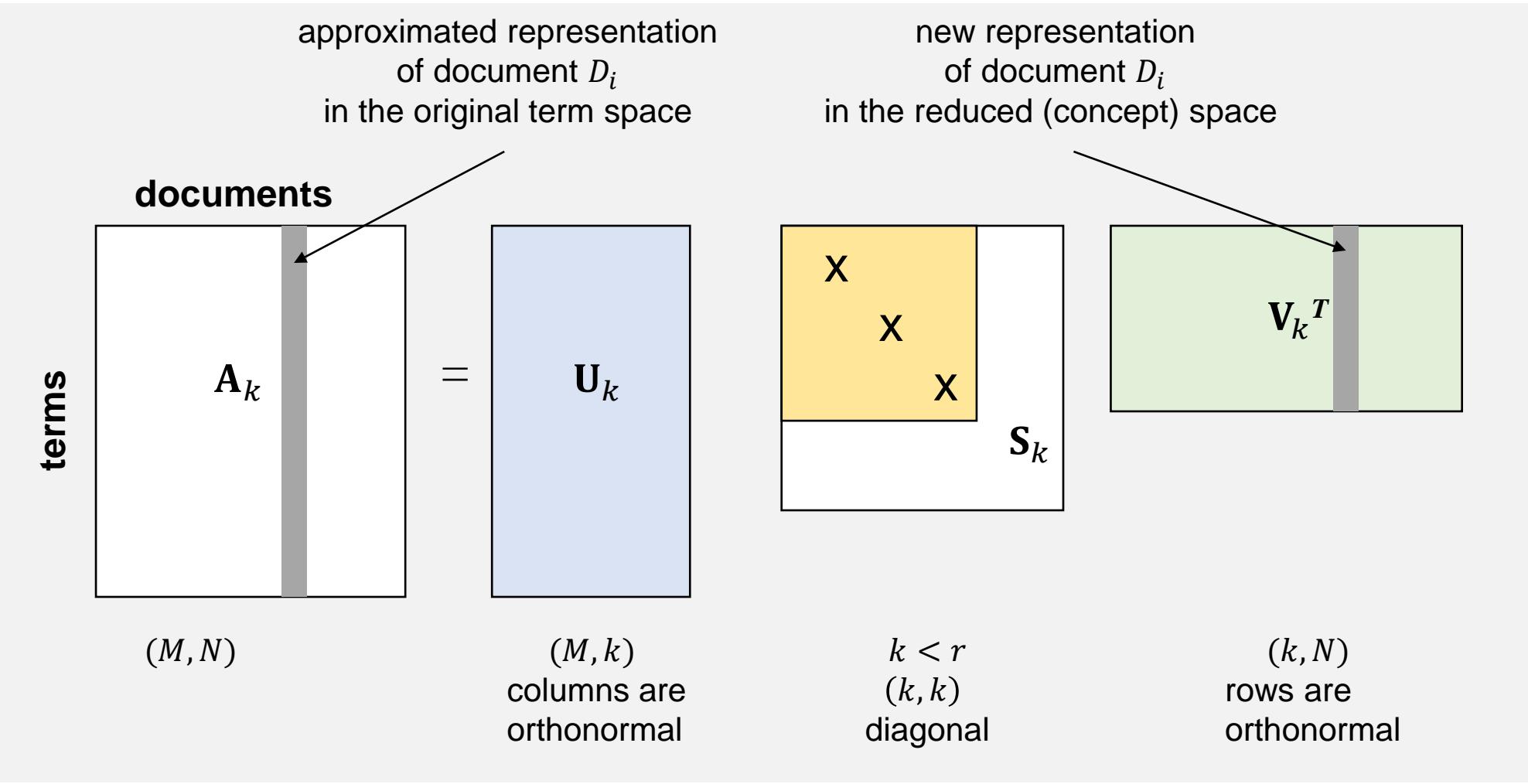
and obtain an approximation $\underline{\mathbf{A}}$ for \mathbf{A} if one or several of the summands are omitted in the above formula. We get the best approximation of rank $k < r$ with the sum over the k largest singular values s_i

3.4.2 Singular value decomposition of the term-document-matrix

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

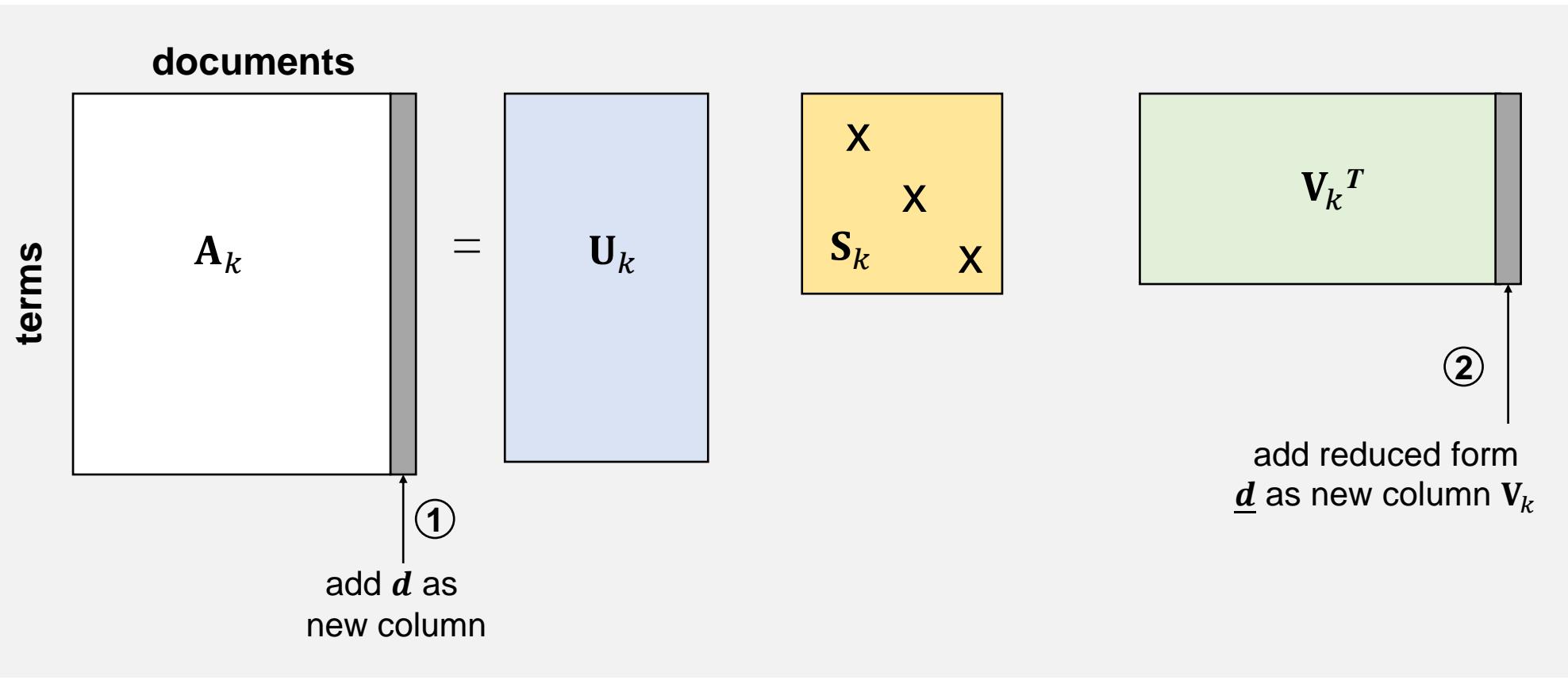


- **Reduction of dimensions:** Sorting elements by decreasing singular values and elimination of the k smallest singular values leads to a new representation with a smaller (k, k) -diagonal matrix \mathbf{S}_k



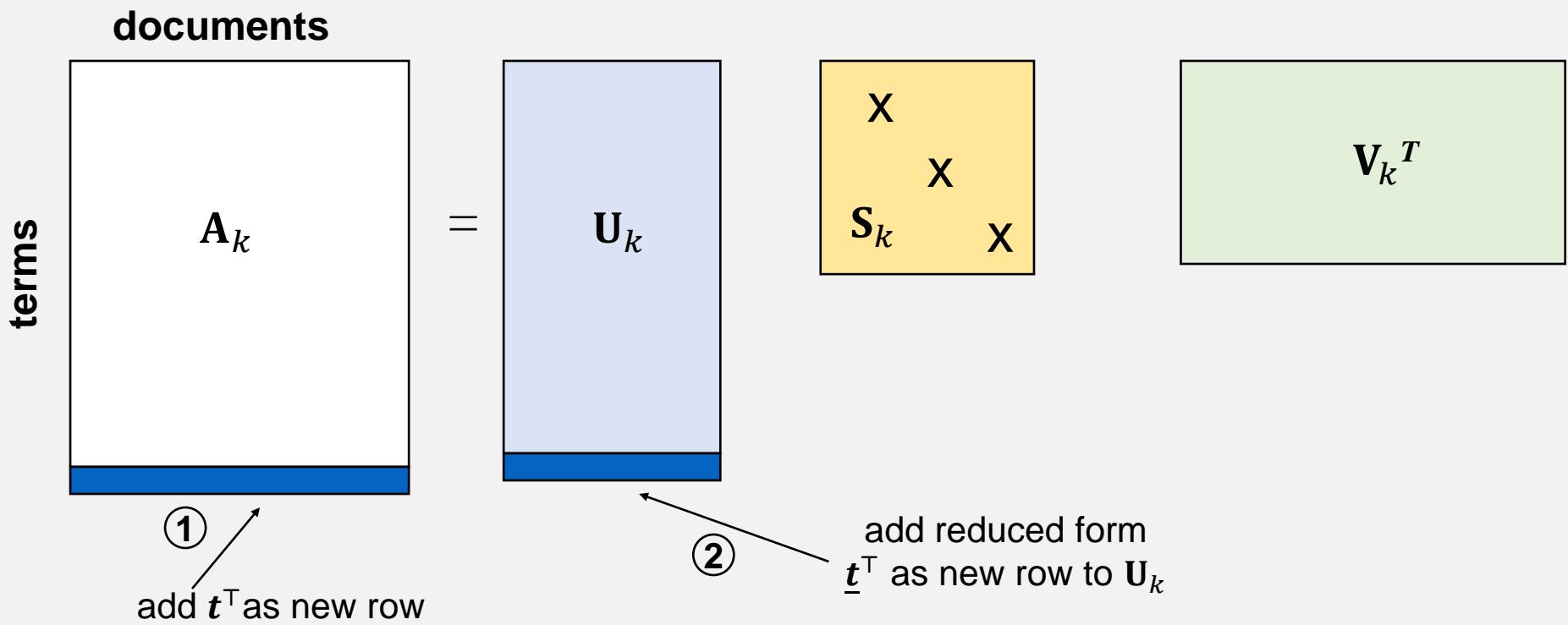
- **Inserting new documents (approximation):** A new document D is added without recalculation of the SVD (approximation for efficiency). We add document vector \underline{d} as a new column to A_k . The reduced form \underline{d} is determined as follows:

$$\underline{d}^\top = \underline{d}^\top \mathbf{U}_k \mathbf{S}_k^{-1}$$



- **Inserting new terms:** We can approximately add a new term T without recomputing the SVD. We add term vector t as a new row to A_k . The reduced form \underline{t} is obtained as follows:

$$\underline{t}^T = t^T \mathbf{V}_k \mathbf{S}_k^{-1}$$



3.4.3 Query evaluation with LSI

- To answer query, a query Q first needs to be mapped to the reduced space of the collection. This is achieved with the following transformation

$$\underline{\mathbf{q}}^\top = \mathbf{q}^\top \mathbf{U}_k \mathbf{S}_k^{-1}$$

Next, we have to compare $\underline{\mathbf{q}}$ with all the reduced representations of documents (\mathbf{V}_k) to identify the most similar ones. The following functions provide two examples of how to compare vectors:

$$sim(Q, D_i) = \mathbf{q} \cdot \mathbf{d}_i = \sum_{j=1}^M q_j \cdot d_{i,j}$$

$$sim(Q, D_i) = \frac{\mathbf{q} \cdot \mathbf{d}_i}{\|\mathbf{q}\| \cdot \|\mathbf{d}_i\|} = \frac{\sum_{j=1}^M q_j \cdot d_{i,j}}{\sqrt{\sum_{j=1}^M q_j^2} \cdot \sqrt{\sum_{j=1}^M d_{i,j}^2}}$$

Similar to vector space retrieval, documents are sorted by decreasing similarity values.

3.4.4 A Simple Example

- The following documents are given:

| | |
|----|--|
| c1 | Human machine interface for Lab ABC computer applications |
| c2 | A survey of user opinion of computer system response time |
| c3 | The EPS user interface management system |
| c4 | System and human system engineering testing of EPS |
| c5 | Relation of user-perceived response time to error measurement |
| m1 | The generation of random, binary, unordered trees |
| m2 | The intersection graph of paths in trees |
| m3 | Graph minors IV: Widths of trees and well-quasi-ordering |
| m4 | Graph minors: A survey |

- Let the query be: "**human computer interaction**"
 - Boolean search with "AND" returns no documents
 - Boolean search with "OR" or vector space retrieval returns the documents c1, c2, c4
 - Also compare with results from exercise with probabilistic retrieval

- Example: the document-term matrix

$A =$
 $(M=12, N=9)$

| | c1 | c2 | c3 | c4 | c5 | m1 | m2 | m3 | m4 |
|-----------|----|----|----|----|----|----|----|----|----|
| human | 1 | | | 1 | | | | | |
| interface | 1 | | 1 | | | | | | |
| computer | 1 | 1 | | | | | | | |
| user | | 1 | 1 | | 1 | | | | |
| system | | 1 | 1 | 2 | | | | | |
| response | | 1 | | | 1 | | | | |
| time | | 1 | | | 1 | | | | |
| EPS | | | 1 | 1 | | | | | |
| survey | | 1 | | | | | | | 1 |
| trees | | | | | | 1 | 1 | 1 | |
| graph | | | | | | | 1 | 1 | 1 |
| minors | | | | | | | | 1 | 1 |

terms which appear in only one document and stop words are omitted

- Example: singular value decomposition

$\mathbf{U} =$

| | | | | | | | | |
|--------|---------|---------|---------|---------|---------|---------|---------|---------|
| 0.2214 | -0.1132 | 0.2890 | -0.4148 | -0.1063 | -0.3410 | 0.5227 | -0.0605 | -0.4067 |
| 0.1976 | -0.0721 | 0.1350 | -0.5522 | 0.2818 | 0.4959 | -0.0704 | -0.0099 | -0.1089 |
| 0.2405 | 0.0432 | -0.1644 | -0.5950 | -0.1068 | -0.2550 | -0.3022 | 0.0623 | 0.4924 |
| 0.4036 | 0.0571 | -0.3378 | 0.0991 | 0.3317 | 0.3848 | 0.0029 | -0.0004 | 0.0123 |
| 0.6445 | -0.1673 | 0.3611 | 0.3335 | -0.1590 | -0.2065 | -0.1658 | 0.0343 | 0.2707 |
| 0.2650 | 0.1072 | -0.4260 | 0.0738 | 0.0803 | -0.1697 | 0.2829 | -0.0161 | -0.0539 |
| 0.2650 | 0.1072 | -0.4260 | 0.0738 | 0.0803 | -0.1697 | 0.2829 | -0.0161 | -0.0539 |
| 0.3008 | -0.1413 | 0.3303 | 0.1881 | 0.1148 | 0.2722 | 0.0330 | -0.0190 | -0.1653 |
| 0.2059 | 0.2736 | -0.1776 | -0.0324 | -0.5372 | 0.0809 | -0.4669 | -0.0363 | -0.5794 |
| 0.0127 | 0.4902 | 0.2311 | 0.0248 | 0.5942 | -0.3921 | -0.2883 | 0.2546 | -0.2254 |
| 0.0361 | 0.6228 | 0.2231 | 0.0007 | -0.0683 | 0.1149 | 0.1596 | -0.6811 | 0.2320 |
| 0.0318 | 0.4505 | 0.1411 | -0.0087 | -0.3005 | 0.2773 | 0.3395 | 0.6784 | 0.1825 |

$\mathbf{S} =$

| | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 3.3409 | | | | | | | | |
| | 2.5417 | | | | | | | |
| | | 2.3539 | | | | | | |
| | | | 1.6445 | | | | | |
| | | | | 1.5048 | | | | |
| | | | | | 1.3064 | | | |
| | | | | | | 0.8459 | | |
| | | | | | | | 0.5601 | |
| | | | | | | | | 0.3637 |

$\mathbf{V}^T =$

| | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|--------|---------|
| 0.1974 | 0.6060 | 0.4629 | 0.5421 | 0.2795 | 0.0038 | 0.0146 | 0.0241 | 0.0820 |
| -0.0559 | 0.1656 | -0.1273 | -0.2318 | 0.1068 | 0.1928 | 0.4379 | 0.6151 | 0.5299 |
| 0.1103 | -0.4973 | 0.2076 | 0.5699 | -0.5054 | 0.0982 | 0.1930 | 0.2529 | 0.0793 |
| -0.9498 | -0.0286 | 0.0416 | 0.2677 | 0.1500 | 0.0151 | 0.0155 | 0.0102 | -0.0246 |
| 0.0457 | -0.2063 | 0.3783 | -0.2056 | 0.3272 | 0.3948 | 0.3495 | 0.1498 | -0.6020 |
| -0.0766 | -0.2565 | 0.7244 | -0.3689 | 0.0348 | -0.3002 | -0.2122 | 0.0001 | 0.3622 |
| 0.1773 | -0.4330 | -0.2369 | 0.2648 | 0.6723 | -0.3408 | -0.1522 | 0.2491 | 0.0380 |
| -0.0144 | 0.0493 | 0.0088 | -0.0195 | -0.0583 | 0.4545 | -0.7615 | 0.4496 | -0.0696 |
| -0.0637 | 0.2428 | 0.0241 | -0.0842 | -0.2624 | -0.6198 | 0.0180 | 0.5199 | -0.4535 |

- Example: reduction of dimensions
- with $k=2$, we obtain

| \mathbf{U}_k | \mathbf{S}_k | \mathbf{V}_k^T |
|----------------|----------------|---|
| 0.2214 -0.1132 | 3.3409 | 0.1974 0.6060 0.4629 0.5421 0.2795 0.0038 0.0146 0.0241 0.0820 |
| 0.1976 -0.0721 | 2.5417 | -0.0559 0.1656 -0.1273 -0.2318 0.1068 0.1928 0.4379 0.6151 0.5299 |
| 0.2405 0.0432 | | |
| 0.4036 0.0571 | | |
| 0.6445 -0.1673 | | |
| 0.2650 0.1072 | | |
| 0.2650 0.1072 | | |
| 0.3008 -0.1413 | | |
| 0.2059 0.2736 | | |
| 0.0127 0.4902 | | |
| 0.0361 0.6228 | | |
| 0.0318 0.4505 | | |

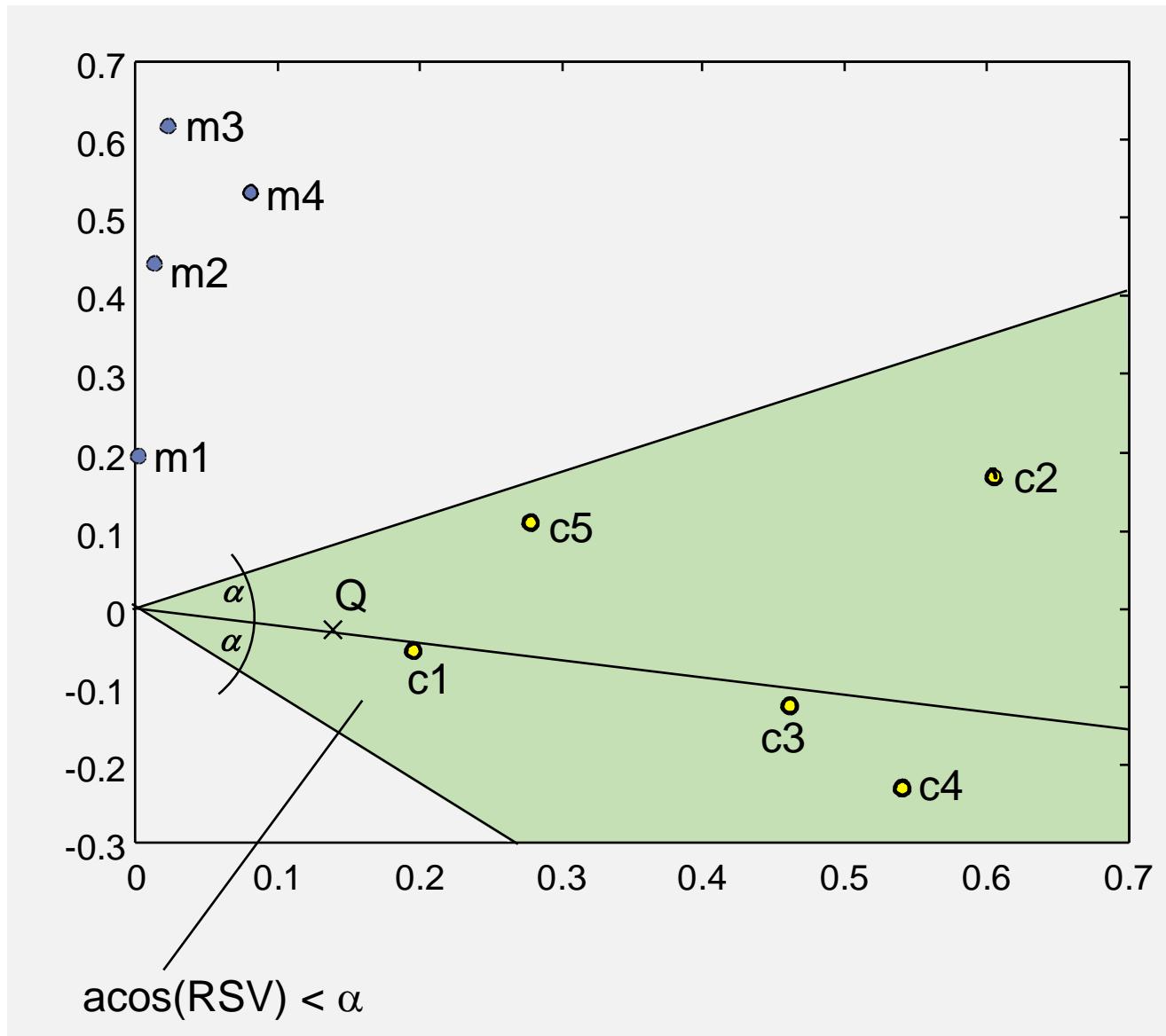
new representation
of document c1

- It follows for: $\mathbf{A}_k = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^T$ and $\underline{\mathbf{q}}^T = \underline{\mathbf{q}}^T \mathbf{U}_k \mathbf{S}_k^{-1}$

| $\mathbf{A}_k =$ | \mathbf{q} | $\underline{\mathbf{q}}$ |
|--|---|--|
| $\mathbf{A}_k = \begin{bmatrix} 0.1621 & 0.4005 & 0.3790 & 0.4676 & 0.1760 & -0.0527 & -0.1151 & -0.1591 & -0.0918 \\ 0.1406 & 0.3698 & 0.3290 & 0.4004 & 0.1650 & -0.0328 & -0.0706 & -0.0968 & -0.0430 \\ 0.1524 & 0.5050 & 0.3579 & 0.4101 & 0.2362 & 0.0242 & 0.0598 & 0.0869 & 0.1240 \\ 0.2580 & 0.8411 & 0.6057 & 0.6974 & 0.3923 & 0.0331 & 0.0832 & 0.1218 & 0.1874 \\ 0.4488 & 1.2344 & 1.0509 & 1.2658 & 0.5563 & -0.0738 & -0.1547 & -0.2096 & -0.0489 \\ 0.1596 & 0.5817 & 0.3752 & 0.4169 & 0.2765 & 0.0559 & 0.1322 & 0.1889 & 0.2169 \\ 0.1596 & 0.5817 & 0.3752 & 0.4169 & 0.2765 & 0.0559 & 0.1322 & 0.1889 & 0.2169 \\ 0.2185 & 0.5496 & 0.5110 & 0.6281 & 0.2425 & -0.0654 & -0.1425 & -0.1966 & -0.1079 \\ 0.0969 & 0.5321 & 0.2299 & 0.2118 & 0.2665 & 0.1368 & 0.3146 & 0.4444 & 0.4250 \\ -0.0613 & 0.2321 & -0.1389 & -0.2656 & 0.1449 & 0.2404 & 0.5461 & 0.7674 & 0.6637 \\ -0.0647 & 0.3353 & -0.1456 & -0.3014 & 0.2028 & 0.3057 & 0.6949 & 0.9766 & 0.8487 \\ -0.0431 & 0.2539 & -0.0967 & -0.2079 & 0.1519 & 0.2212 & 0.5029 & 0.7069 & 0.6155 \end{bmatrix}$ | $\mathbf{q} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ | $\underline{\mathbf{q}} = \begin{bmatrix} 0.1382 \\ -0.0276 \end{bmatrix}$ |

- Visualization in reduced feature space

- Let the cosine measure be the similarity function: all documents which lie inside the green area are relevant
- Documents c_1, \dots, c_5 are similar to the query. Notice that c_3 does not contain any query term but still is among the best matching documents
- In this (constructed) example, we identify **two main topics** ($k=2$): the c -documents have larger values on the horizontal axis, the m -documents have larger values on the vertical axis



- **Advantages:**
 - Synonyms are automatically detected
 - Simplifies features extraction; no dictionary and ontology required; different languages and cross-language retrieval are for free; stemming not necessary
 - Good retrieval quality
- **Disadvantages:**
 - Extremely expensive computation of SVD; fast algorithms for parallel computations necessary (but not available)
 - Expensive evaluation of queries; there exists no optimal index structure to support LSI queries; inverted lists are no longer feasible due to dense vectors obtained after transformation
 - Retrieval quality not much better than with other methods; ratio of quality over performance is very poor

3.5 Naive Bayes

- **Bayesian classifiers go back to 1950** It has been applied in many areas, and still is competitive in text classification and medical diagnosis. Especially, Naïve Bayes scales very well to large feature dimensions where other methods, like decision trees, struggle from the curse of dimensionality.
- In the following, we use Naïve Bayes for a simple sentiment analysis of text. We first introduce the generic method and then apply it to a bag-of-word model of text documents. “Sentiment” is just one example of extracting a higher-level classifier. Other examples include “spam filter”, “email routing”, “language detection”, “genre classification”, and “author detection”.
- In a nutshell, Naïve Bayes classification uses terms and their frequency to automatically detect correlation between terms and classes with the help of a (labeled) test data set. With these correlations, we can automatically classify new documents and process them accordingly. Naïve Bayes uses a simple and very efficient approach and classifies with high quality.

- Naïve Bayes uses a conditional probability model based on Bayes theorem:

$$P(C_k|x) = \frac{P(x|C_k) \cdot P(C_k)}{P(x)}$$

$$posterior = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

where x is a feature vector and C_k the class (=target). $P(C_k)$ is the so-called “prior”, i.e., the knowledge (here a probability) about the distribution of classes C_k . $P(x|C_k)$ is the **likelihood** to observe the feature x for a given class C_k , and $P(x)$ is the **evidence** to observe x (for any class). $P(C_k|x)$ is then the so-called “posterior”, i.e., the knowledge we gain (or better: predict) given the observation of feature x to infer that it belongs to class C_k .

- Let x be a high-dimensional vector, for instance, from a huge term space for documents. Due to the high-dimensionality and the limited set of training data, it is difficult to accurately describe the probability distribution function in such a sparse space. To simplify matters, **naïve Bayes assumes conditional independence of features**. This immediately leads to the following simplification:

$$P(C_k|x) = P(C_k|x_1, \dots, x_M) = \frac{1}{P(x)} \cdot P(C_k) \cdot \prod_{j=1}^M P(x_j|C_k)$$

Note that $P(x)$ is a constant over classes c_k and scales the probabilities. For our purposes, we do not need to know it.

- Given the probability model, we pick the hypothesis (here: class C_{k^*}) which is most probable. This selection rule is also known as the **maximum a posteriori (MAP)**:

$$k^* = \operatorname{argmax}_k P(C_k|x) = \operatorname{argmax}_k P(C_k) \cdot \prod_{j=1}^M P(x_j|C_k)$$

That is it! The equation describes the decision rule of Naïve Bayes. The only thing left are the estimates for the probabilities on the right hand side

- To obtain the prior and the likelihood, we need to estimate the probability distributions based on the training set. And we need to address a number of practical issues such as numerical underflow due to the multiplication of many (small) probabilities, smoothing to address missing features, and feature selection. At the end, we apply the method to text classification
- **Learning process**
 - Estimating $P(C_k)$ is the easy part: let N_k be the number of training items with label C_k and let N be the total number of training items. Then:

$$P(C_k) = \frac{N_k}{N}$$

If the exact numbers are not clear (for instance, spam classifier: what is the ratio between spam and normal email?), the probabilities can be approximated with $P(C_k) = 1/K$ with K denoting the number of classes, i.e., equiprobable classes. This is not accurate but works well.

- To find the probability distribution $P(x_j | C_k)$ we first need to model the underlying distribution of values for x_j , and then learn the model parameters from the training set. The typical approach to learn estimators from training data is the **maximum likelihood estimation (MLE)**, i.e., choosing model parameters that maximize the likelihood of making the observations given the parameters.
- Let x_j be discrete with values from \mathbb{V}_j . Let $N_k(x_j = v)$ with $v \in \mathbb{V}_j$ be the number of training items with label C_k that have $x_j = v$. In other words, it denotes how often $x_j = v$ is observed in the training set for items belonging to the class C_k . Naturally, we obtain

$$P(x_j = v | C_k) = \frac{N_k(x_j = v)}{N_k}$$

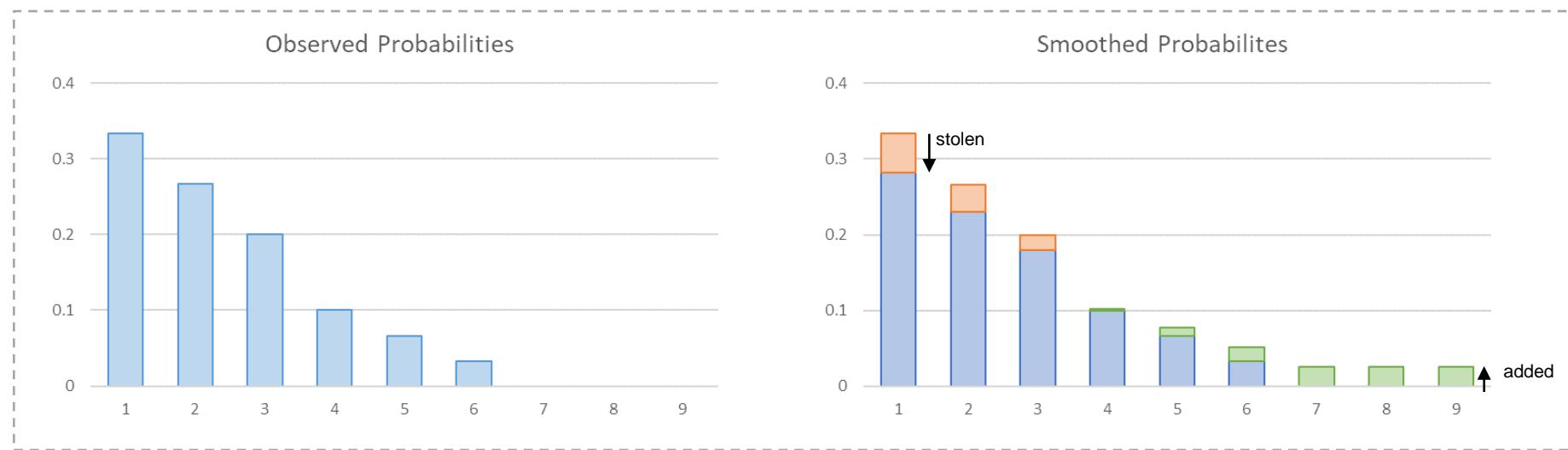
- What if a value v is never seen for x_j over a class C_k . Obviously, $P(x_j = v | C_k) = 0$ and with that:

$$P(C_k | \mathbf{x}) = P(C_k | x_1, \dots, x_j = v, \dots, x_M) = 0$$

In other words, if v was never observed for a class C_k , its presence in a new data item **eliminates C_k as a prediction** regardless how well the other features support C_k . To prevent 0-probabilities, we need to smooth the probability distribution, commonly using **Laplace smoothing (add-1)**. The idea is that we “steal” probability mass and distribute it to the values with 0-probabilities:

$$P(x_j = v | C_k) = \frac{N_k(x_j = v) + 1}{N_k + |\mathbb{V}_j|}$$

Note: the sum of $P(x_j = v | C_k)$ over all values $v \in \mathbb{V}_j$ is still 1. But we got rid of 0-probabilities.



Red indicates “stolen” probability mass and green denotes added probability mass.

- A special case is a discrete Boolean value $x_j \in \{0,1\}$ denoting the presence ($x_j = 1$) or absence ($x_j = 0$) of a feature in the training data. In this case, the distribution follows a Bernoulli event model (or a multivariate **Bernoulli event model** if several values are Boolean). As the probabilities sum up to 1, only one parameter is required:

$$P(x_j | C_k) = (p_{k,j})^{x_j} \cdot (1 - p_{k,j})^{(1-x_j)}$$

with $p_{k,j}$ representing the probability that the feature is present, i.e., how often $x_j = 1$ is observed in the training set for objects with label C_k . Hence:

$$p_{k,j} = \frac{N_k(x_j = 1)}{N_k} \quad \text{or smoothed:} \quad p_{k,j} = \frac{\min(N_k - 1, \max(1, N_k(x_j = 1)))}{N_k}$$

Note that smoothing is done with stealing 1 only in the extreme case that all observations are the same (either all $x_j = 1$ or all $x_j = 0$).

- A final case for discrete values is the **multinomial event model** which is given by a feature vector $\mathbf{x} = (x_1, \dots, x_M)$ representing a histogram with x_j counting the number of times a feature or event j was observed in the training set. We will see an example later on with x_j denoting the number of occurrences of a term t_j in a document. The probability distribution is given by:

$$P(\mathbf{x} | C_k) = \frac{(\sum_j x_j)!}{\prod_j x_j!} \cdot \prod_j (p_{k,j})^{x_j}$$

Note that the factor to the left of the product symbol is a constant when looking for the best class C_k and hence drops in the argmax equation

Let $n_{k,j}$ be the total number of occurrences of feature j in all training items with label C_k . Then:

$$p_{k,j} = \frac{n_{k,j}}{\sum_l n_{k,l}} \quad \text{or smoothed:} \quad p_{k,j} = \frac{n_{k,j} + 1}{\sum_l n_{k,l} + M}$$

- If feature values x_i are continuous, we need to choose a model for the probability distribution $p(x_i|C_k)$ and then learn the parameters of the model using the training set. A common approach is assuming a **Gaussian distribution** with the two parameters $\mu_{k,i}$ denoting the mean value, and $\sigma_{k,i}^2$ being the variance. The probability distribution is defined as:

$$p(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} \cdot e^{-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}}$$

To estimate the two parameters, we need to use the **unbiased estimators** based on the observations from the training set. Let $N_k = |C_k|$ be the number of training items with label C_k :

$$\mu_{k,i} = \frac{1}{N_k} \sum_{x \in C_k} x_i \quad \sigma_{k,i} = \frac{1}{N_k - 1} \sum_{x \in C_k} (x_i - \mu_{k,i})^2$$

When estimating variance from samples, we must account for the error in the estimated mean value, that is, we underestimate the variance because differences between values and the estimated mean are too small.

- Using a **Gaussian mixture model**, we can adopt to arbitrarily shaped distribution function. We overlay L normal distributions $\mathcal{N}(\mu_{k,i,l}, \sigma_{k,i,l}^2)$ with weights w_l :

$$p(x_i|C_k) = \sum_{l=1}^L w_l \cdot \mathcal{N}(\mu_{k,i,l}, \sigma_{k,i,l}^2)$$

To learn the parameters of the normal distributions, we can use the **Expectation Maximization** approach (we will see this later for clustering methods). In addition, we should use a validation set to adjust the hyper-parameter L , i.e., if L is large, we may fit the probability distribution for the training set very well, but cannot generalize well to the validation set due to overfitting. Using least mean squared errors over the validation set provides an instrument to control L .

- **Prediction**

- To predict the class C_{k^*} to which a new data item with features x belongs to, we apply the maximum a posteriori (MAP) selection:

$$k^* = \operatorname{argmax}_k P(C_k|x) = \operatorname{argmax}_k P(C_k) \cdot \prod_{j=1}^M P(x_j|C_k)$$

With moderate to large numbers for M , we run into practical issues due to the multiplications of small probabilities (numerical underflow). To provide a stable calculation of the probabilities, naïve Bayes algorithms compute log-probabilities as the logarithm does not impact the ordering:

$$k^* = \operatorname{argmax}_k \log(P(C_k|x)) = \operatorname{argmax}_k \left(\log P(C_k) + \sum_{j=1}^M \log P(x_j|C_k) \right)$$

- To reduce the noise of a large number of features, we can focus on a few features only that are sufficient to classify data items. In general terms, we want to identify features whose presence or absence is correlated with the data item having or not having a label. This leads to 4 tests for each of the combinations of {"feature present", "feature not present"} and {"item in class", "item not in class"}. If there is a strong correlation for any combination of events, then the feature is discriminative for classification. Literature provides several approaches with **Chi-square** and **mutual information** being the most prominent ones. A much simpler approach is to select the most discriminative features, much like we have seen in classical text retrieval.

- **Example: Text Classification** – Naïve Bayes is quite popular due to its simplicity, its speed, and accuracy. Common applications include spam detection, author identification, age/gender identification, language identification, and sentiment analysis. With sentiment analysis, for example, we want to distinguish positive from negative movie reviews.
 - There are two models for text classification: 1) set of words, and 2) bag of words. With the former, we consider only the presence of terms and apply a multivariate Bernoulli model. With the latter, we count term occurrences and use the multinomial model. Both approaches assume that the position of terms in the text does not matter and that terms are conditionally independent.
 - **Set of words and multivariate Bernoulli:** like with Boolean text retrieval models, a **binary feature vector x** denotes the presence of terms, taken from a defined vocabulary, in the given documents. The training documents have labels for **classes C_k** , and we use the training set to estimate the probabilities. Let N_k be the number of training items with label C_k , then

$$P(C_k) = \frac{N_k}{N}$$

or if N_k is not known: $P(C_k) = \frac{1}{K}$

Let $x_j = 1$ denote that term t_j is present in the document represented by x . Then:

$$p_{k,j} = \frac{N_k(x_j = 1)}{N_k}$$

or smoothed: $p_{k,j} = \frac{\min(N_k - 1, \max(1, N_k(x_j = 1)))}{N_k}$

Prediction means finding the class that maximizes $P(C_k|x)$ for a document with representation x :

$$k^* = \operatorname{argmax}_k P(C_k|x) = \operatorname{argmax}_k \left(\log P(C_k) + \sum_{j=1}^M (x_j \log p_{k,j} + (1 - x_j) \log(1 - p_{k,j})) \right)$$

Instead of using all terms of the vocabulary, we can reduce the features (see feature selection) or only take the terms present in the document (i.e., we only consider $x_j = 1$).

- **Bag of words and multinomial**: like with vector space retrieval models, a feature vector \mathbf{x} denotes the number of occurrences of terms, taken from a defined vocabulary, in the given documents. The training documents have labels for **classes** C_k , and we use the training set to estimate the probabilities. Let N_k be the number of training items with label C_k , then

$$P(C_k) = \frac{N_k}{N} \quad \text{or if } N_k \text{ is not known: } P(C_k) = \frac{1}{K}$$

Let $n_{k,j}$ be the total number of occurrences of term t_j in all training documents with label C_k :

$$p_{k,j} = \frac{n_{k,j}}{\sum_l n_{k,l}} \quad \text{or smoothed: } p_{k,j} = \frac{n_{k,j} + 1}{\sum_l n_{k,l} + M}$$

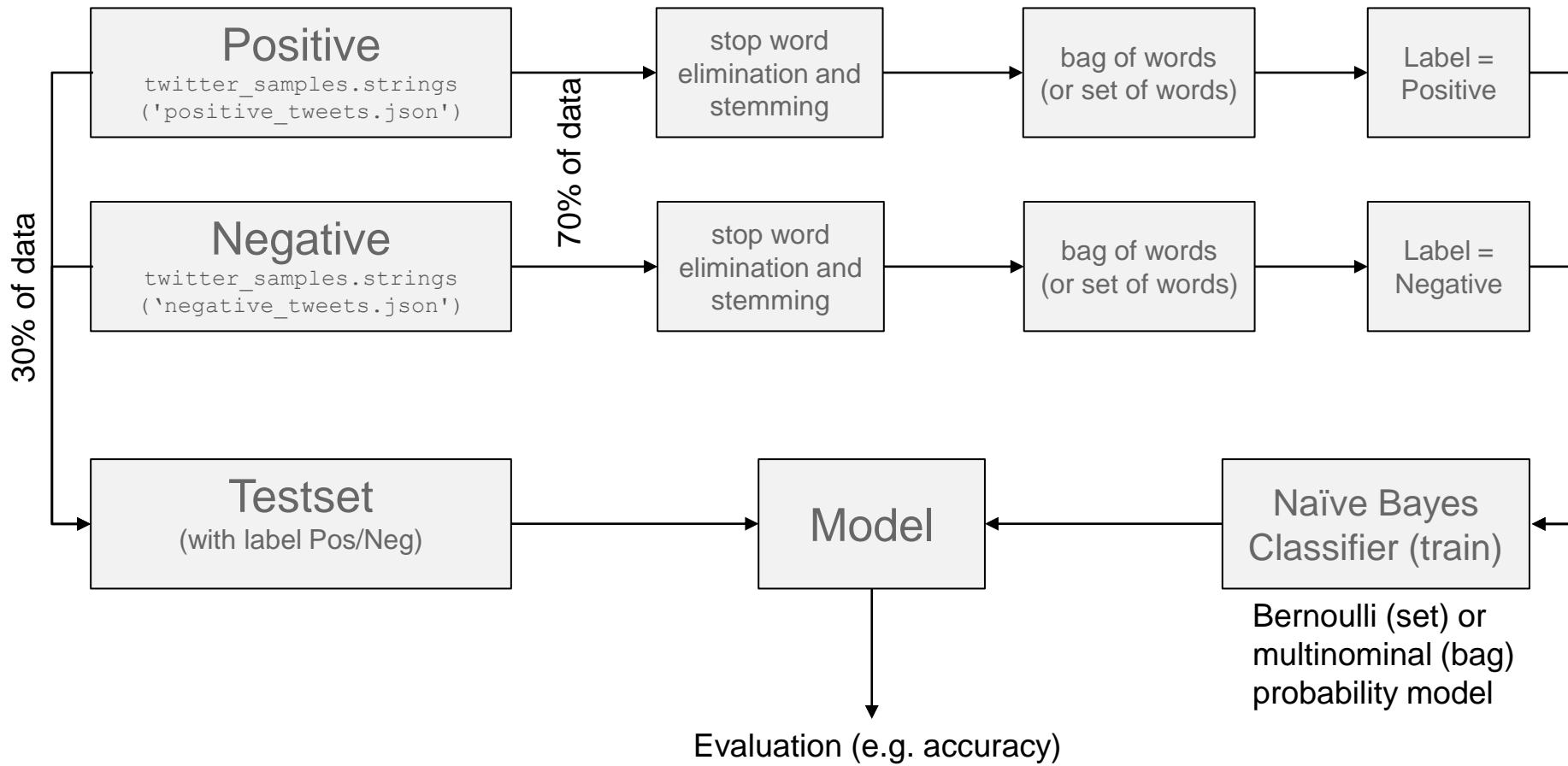
Prediction means finding the class that maximizes $P(C_k|\mathbf{x})$ for a document with representation \mathbf{x} :

$$k^* = \operatorname{argmax}_k P(C_k|\mathbf{x}) = \operatorname{argmax}_k \left(\log P(C_k) + \sum_{x_j > 0} x_j \log p_{k,j} \right)$$

That is, we select the best class only with the terms that are present in the document.

- **Summary: Naïve Bayes is not so naïve.** Even though the strong assumption of independence does not always apply in practice, it excels due to high speed, low storage requirements, robustness to noise, and very good performance (accuracy). There are better methods but still naïve Bayes is an excellent baseline for text classification.

- Application: Sentiment Analysis of tweets
 - We use NLTK to learn positive and negative tweets (corpus `twitter_samples`)



3.6 Literature and Links

Publications

- Brin, S.; Page, L. (1998). ["The anatomy of a large-scale hypertextual Web search engine"](#) (PDF). Computer Networks and ISDN Systems. 30: 107–117.
- Page, Lawrence; Brin, Sergey; Motwani, Rajeev; Winograd, Terry (1999). ["The PageRank citation ranking: Bringing order to the Web"](#)., published as a technical report on January 29, 1998 [PDF](#)
- Lempel, R.; Moran S. (April 2001). ["SALSA: The Stochastic Approach for Link-Structure Analysis"](#) (PDF). ACM Transactions on Information Systems. 19 (2): 131–160.
- G. Jeh and J. Widom. SimRank: A Measure of Structural-Context Similarity. In [KDD'02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining](#), pages 538-543. [ACM Press](#), 2002. [PDF](#)
- J. Dean, M. Henzinger, Finding related pages in the World Wide Web, Computer networks 31 (11), 1467-1479. [PDF](#)
- J. Kleinberg. ["Hubs, Authorities, and Communities"](#). ACM Computing Surveys 31(4), 1999.

Books

- Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python. O'Reilly Media, 2009. Free online version: <http://www.nltk.org/book/>

Thesaurus & Ontologies for selected Languages

- EuroWordNet: <http://www.illc.uva.nl/EuroWordNet/>
- GermanNet: <http://www.sfs.uni-tuebingen.de/lst/>
- WordNet: <http://www.cogsci.princeton.edu/~wn/>

Implementations

- Natural Language Toolkit (NLTK), <http://www.nltk.org>
- Apache Lucene, <https://lucene.apache.org>