Digital Libraries: Language Technologies

RAFFAELLA BERNARDI

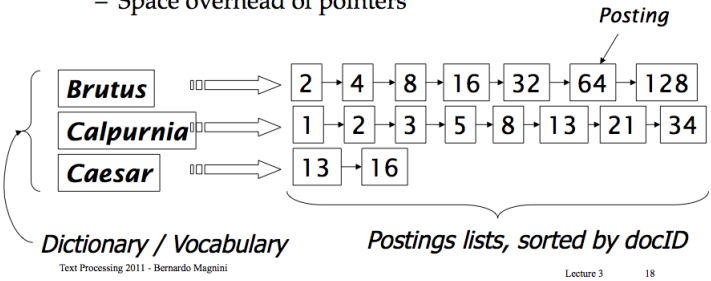
UNIVERSITÀ DEGLI STUDI DI TRENTO

P.ZZA VENEZIA, ROOM: 2.05, E-MAIL: BERNARDI@DISI.UNITN.IT

Contents

1. Recall: Inverted Index

- **Linked lists** generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers

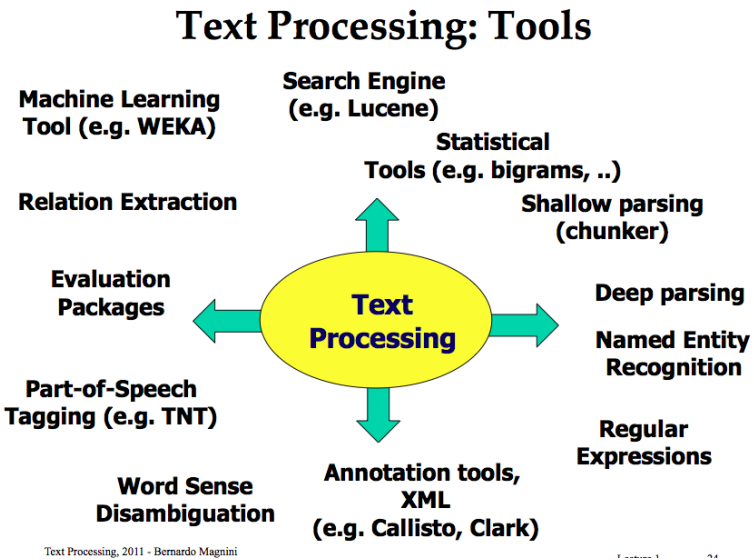


Text Processing 2011 - Bernardo Magnini Lecture 3 18

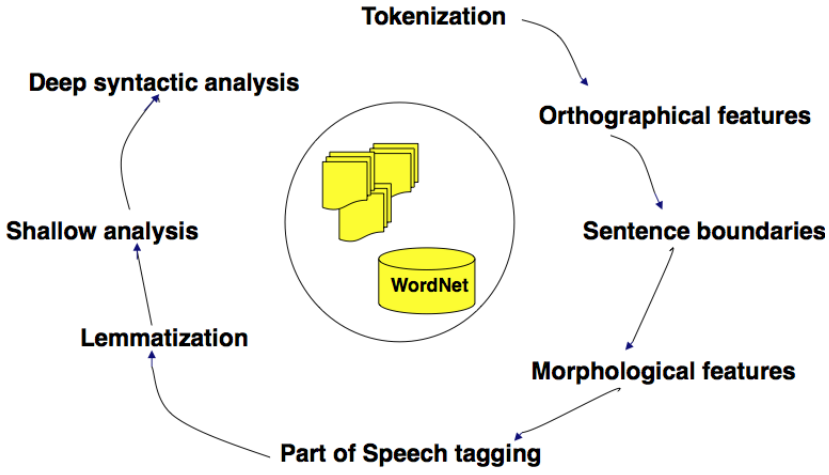
We gave per granted we know:

- what a document is.
- how to “machine-read” each document.

2. Text Processing: tools



3. Text Processing: Pre-processing



4. Definitions: Word, Term, Token, Type

Word A delimited string of characters as it appears in the text.

Term A "normalized" word (case, morphology, spelling etc); an equivalence class of words.

Token An instance of a word or term occurring in a document.

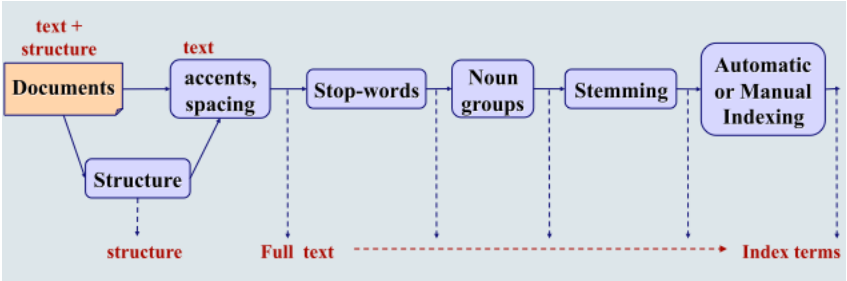
Type The same as a term in most cases: an equivalence class of tokens.

Example: Tokens vs. Words Example: “andarci veramente piano”

- 3 token: “andarci veramente piano” [graphical words]
- 4 words: “andare ci veramente piano” [inflected form]
- 2 lexical unit: “andarci_piano veramente” [unitary meaning]

- Tokens: token in different position are different.
- Types: token with no repetitions (position is not relevant)
- Lemmas: the normalized form of a word (e.g. infinitive for verbs,)

5. Document transformation for indexing



5.1. Normalization

Need to "normalize" terms in indexed text as well as query terms into the same form. E.g.

- U.S.A vs. USA
- Accent: naive vs. naïve, resume vs. résumé.
- Case folding: Fed vs. fed

We most commonly implicitly define equivalence classes of terms (windows, window, Windows, etc.)

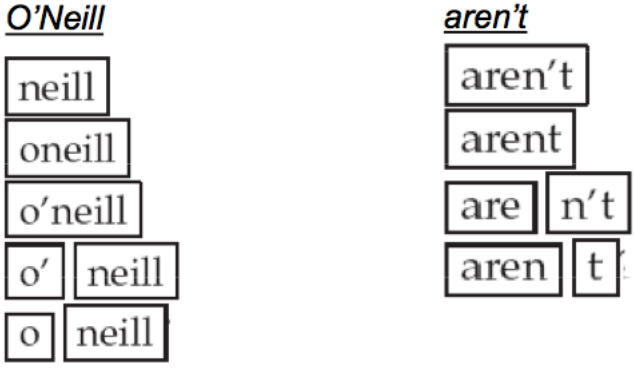
Most important criterion: How are users likely to write their queries for these words?

Even in languages that standardly have accents, users often do not type them.

Most often users use lowercase, hence it's better to lowercase everything.

5.2. Tokenization

Input: Friends, Romans, Countrymen, lend me your ears;
Output: Friends | Romans | Countrymen | lend | me | your | ears



Each token is a candidate for a postings entry.

5.3. Tokenkization difficulties

One word or two (or several)?

- Hewlett-Packard
- State-of-the-art
- co-education
- data base
- San Francisco

Numbers

- 3/20/91 vs. 20/3/91
- Mar 20, 1991
- 100.2.86.144
- 800.234.2333

5.4. Lemmatization

Definition Reduce inflectional/variant forms to base form. E.g.:

- *am, are, is* $\rightarrow$ *be*
- *car, cars, car's, cars'* $\rightarrow$ *car*
- *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*

Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma). Inflectional morphology (*cutting* $\rightarrow$ *cut*) vs. derivational morphology (*destruction* $\rightarrow$ *destroy*)

5.5. Stemming

Definition Crude heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge. It's language dependent. E.g.:

- automate(s), automatic, automation all reduce to automat.
- "for example compressed and compression are both accepted as equivalent to compress" → "for exampl compress and compress ar both accept as equival to compress"

Downloadable, eg. www.tartarus.org/~martin/PorterStemmer Sample rules:

- sses → ss
- ies → i
- ational → ate

5.6. Sentence splitting

Statistics based methods:

- Lexical category of preceding word
- Lexical category of following word
- Case of preceding word
- Case of following word
- ...

5.7. Stop word list

Definition stop words are extremely common words which would appear to be of little value in helping select documents matching a user need. E.g. *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*

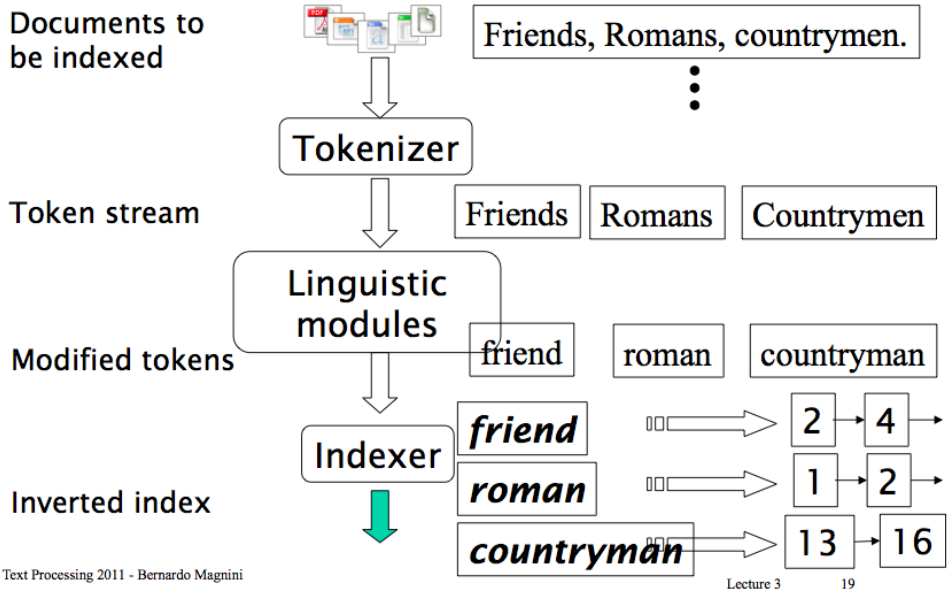
250-300 most common words in English account for 50% or more of a given text

E.g. “the” and “of” represent 10% of tokens in a corpus. “and” “to”, “a” and “in” another 10%. etc.

Moby Dick Ch1: 859 unique words (types), 2256 word occurrences (tokens)

However, if good compression techniques are used stop words can be considered too and in same cases may turn out to be useful. E.g. “let it be”. Hence, nowadays, most web search engines index stop words.

5.8. LT and Inverted Index construction



6. Problem with Boolean search

- Boolean queries often result in either too few (=0) or too many (1000s) results. In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.
- We wish to be able to rank documents, and rank higher the relevant ones.
- Hence, we need to assign a score to each query-document pair, say in $[0, 1]$. This score measures how well document and query "match".

7. Query-document matching scores I

How do we compute the score of a query-document pair?

Let's start with a one-term query.

- If the query term does not occur in the document: score should be 0.
- The more frequent the query term in the document, the higher the score

We will look at a number of alternatives for doing this.

7.1. Take 1: Jaccard coefficient

A commonly used measure of overlap of two sets. Let A and B be two sets, such that $A \neq \emptyset$ or $B \neq \emptyset$, Jaccard coefficient:

$$\text{JACCARD}(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

$\text{JACCARD}(A,A) = 1$, $\text{JACCARD}(A,B) = 0$ if $A \cap B = 0$ A and B don't have to be the same size. It always assigns a number between 0 and 1.

Jaccard coefficient: Example What is the query-document match score that the Jaccard coefficient computes for:

- Query: "ides of March"
- Document "Caesar died in March"
- $\text{JACCARD}(q,d) = 1/6$

7.2. What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.

8. Term weights

We need a way to compute the weight of a term in the term-document matrix.

8.1. Term Frequency (tf)

Definition The term frequency $\text{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.

We want to use tf when computing query-document match scores. But raw term frequency is not what we want because:

Normalized tf tf count is usually normalized to prevent a bias towards longer documents (which may have a higher term count regardless of the actual importance of that term in the document) to give a measure of the importance of the term ti within the particular document d_j.

$$\text{tf}_{\text{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where $n_{i,j}$ is the number of occurrences of the considered term (t_i) in document d_j, and the denominator is the sum of number of occurrences of all terms in document d_j, that is, the size of the document $|d_j|$.

- A document with tf = 10 occurrences of the term is more relevant than a document

with tf $= 1$ occurrence of the term.

- But not 10 times more relevant.
- Relevance does *not increase proportionally* with term frequency.

8.2. Log frequency weighting

The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\mathrm{tf}_{t,d} \rightarrow w_{t,d}$:
  $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d:
  $\text{tf-matching-score}(q,d) = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Recall: The logarithm of a number y with respect to base b ($\log_b y$) is the exponent to which b has to be raised in order to yield y. In other words, the logarithm of y to base b is the solution x of the equation

$b^x = y$

8.3. Exercise: Jaccard and tf measure

Compute the Jaccard matching score and the tf matching score for the following query-document pairs.

- q: [information on cars] d: "all you've ever wanted to know about cars"
- q: [information on cars] d: "information on trucks, information on planes, information on trains"
- q: [red cars and red trucks] d: "cops stop red cars more often"

8.4. Problem: Rare vs. Frequent terms

In addition, to term frequency (the frequency of the term in the document), we also want to use the frequency of the term in the *collection* for weighting and ranking.

- Desired weight for *rare terms*
  - Rare terms are more informative than frequent terms.
  - Consider a term in the query that is rare in the collection (e.g., *arachnocentric*).
  - A document containing this term is very likely to be relevant.
  - $\rightarrow$ We want high weights for rare terms like *arachnocentric*.
- Desired weight for *frequent terms*
  - Frequent terms are less informative than rare terms.
  - Consider a term in the query that is frequent in the collection (e.g., *increase*).
  - A document containing these terms is more likely to be relevant than a document that doesn't but frequent terms are not sure indicators of relevance.
  - $\rightarrow$ For frequent terms we want positive weights but lower weights than for rare terms.

8.5. Document Frequency

- We want high weights for rare terms like *arachnocentric*.
- We want low (positive) weights for frequent words like *good*, *increase*, and *line*.
- We will use *document frequency* to factor this into computing the matching score.

Document frequency is the number of documents in the collection that the term occurs in.

8.6. Inverse Document Frequency (idf)

It estimates the rarity of a term in the whole document collection. (If a term occurs in all the documents of the collection, its IDF is zero.)

- idf affects the ranking of documents for queries with at least two terms.
- For example, in the query “arachnocentric line”, idf weighting increases the relative weight of *arachnocentric* and decreases the relative weight of *line*.
- idf has little effect on ranking for one-term queries.

$$\text{idf}_\text{i} = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

with $|D|$: cardinality of D, or the total number of documents in the corpus $|\{j : t_i \in d_j\}|$: number of documents where the term t_i appears (viz. the document frequency) (that is $n_{i,j} \neq 0$). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to use $1 + |\{j : t_i \in d_j\}|$

Example $|D| = 1,000,000$ $\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

8.7. Collection Frequency vs. Document Frequency

word	collection frequency	document frequency
insurance	10440	3997
try	10422	8760

- *Collection frequency* of t: number of tokens of t in the collection
- *Document frequency* of t: number of documents t occurs in
- Which word is a better search term?
- Which frequency is more relevant?

This example suggests that *df* (and idf) is better for weighting than cf: we want the few documents containing "insurance" to get higher boost for a query on "insurance", than the many documents containing "try" should get from a query on "try".

8.8. Tf-idf weighting

The tf-idf weight of a term is the *product* of its tf weight and its idf weight.

It's the best weighting scheme in IR.

- It increases with the number of occurrences within a document. [term frequency]
- It increases with the rarity of the term in the collection. [inverse document frequency]

8.9. Summing up: Weights

Term Frequency As first approximation we can consider the *frequency* of the term in the document.

Inverse Document Frequency Estimate the *rarity* of a term in the whole document collection. (If a term occurs in all the documents of the collection, its IDF is zero.)

TF-IDF A *combination* of the term frequency (TF) and the inverse document frequency (IDF).

Others: http://www.lans.ece.utexas.edu/~strehl/diss/note52.html

8.10. Summing up: Binary, Count, Weight Matrix

	d1	d2	d3	d4	d5	d6	...
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	

	d1	d2	d3	d4	d5	d6	...
Brutus	4	157	0	2	0	0	
Caesar	232	227	0	2	1	0	
Calpurnia	0	10	0	0	0	0	

	d1	d2	d3	d4	d5	d6	...
Brutus	1.21	6.10	0.0	1.0	0.0	0.0	
Caesar	8.59	2.54	0.0	1.51	0.25	0.0	
Calpurnia	0.0	1.54	0.0	0.0	0.0	0.0	

9. Background: Vector Space

A vector space is a mathematical structure formed by a collection of vectors: objects that may be added together and multiplied ("scaled") by numbers, called scalars in this context.

Vector an n-dimensional vector is represented by a column:

$$\begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix}$$

or for short as $\vec{v} = (v_1, \dots v_n)$.

9.1. Operations on vectors

Vector addition:

$$\vec{v} + \vec{w} = (v_1 + w_1, \ldots v_n + w_n)$$

similarly for the $-$.

Vectors are visualized by arrows. They correspond to points (the point where the arrow ends.)

9.2. Dot product or inner product

$$\vec{v} \cdot \vec{w} = (v_1 w_1 + \ldots + v_n w_n) = \sum_{i=1}^{n} v_i w_i$$

Example We have three goods to buy and sell, their prices are (p_1, p_2, p_3) (price vector $\vec{p}$). The quantities we are buy or sell are (q_1, q_2, q_3) (quantity vector $\vec{q}$, their values are positive when we sell and negative when we buy.) Selling the quantity q_1 at price p_1 brings in $q_1 p_1$. The total income is the dot product

$$\vec{q} \cdot \vec{p} = (q_1, q_2, q_3) \cdot (p_1, p_2, p_3) = q_1 p_1 + q_2 p_2 + q_3 p_3$$

9.3. Length

Dimension of a vector the number of the *components* of a vector. We take all vectors to have the same dimension N (N is the terms of the vocabulary of the collection)

Length $||\vec{v}|| = \sqrt{\vec{v} \cdot \vec{v}} = \sqrt{\sum_{i=1}^{n} v_i^2}$

For instance, given the vector:

$$\vec{d} = (4,4,4,4)$$

its length is computed as:

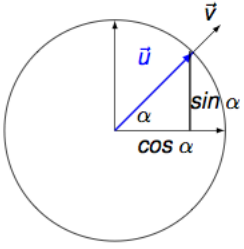
$|\vec{d}| = \sqrt{\sum_{i=1}^{n} d_i^2} = \sqrt{16+16+16+16} = \sqrt{64} = 8$

9.4. Unit vector

is a vector whose length equals one.

$$\vec{u} = \frac{\vec{v}}{||\vec{v}||}$$

is a unit vector in the same direction as $\vec{v}$. (normalized vector)

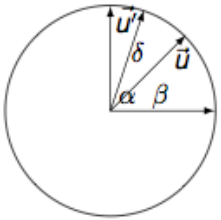


$\vec{u} = \frac{\vec{v}}{||\vec{v}||} = (\cos\alpha, \sin\alpha)$

9.5. Cosine formula

Given δ the angle formed by the two unit vectors $\vec{u}$ and $\vec{u'}$, s.t.
$\vec{u} = (cos\ \beta, sin\ \beta)$ and $\vec{u'} = (cos\ \alpha, sin\ \alpha)$

$$\vec{u} \cdot \vec{u'} = (cos\ \beta)(cos\ \alpha) + (sin\ \beta)(sin\ \alpha) = cos(\beta - \alpha) = cos\ \delta$$



Given two arbitrary vectors *v* and *w*:

$$cos\ \delta = \frac{\vec{v}}{||\vec{v}||} \cdot \frac{\vec{w}}{||\vec{w}||}$$

The bigger the angle δ, the smaller is $cos\ \delta$; $cos\ \delta$ is never bigger than 1 (since we used unit vectors) and never less than -1. It's 0 when the angle is 90^o

10. Vector Space Model: Document as vectors

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

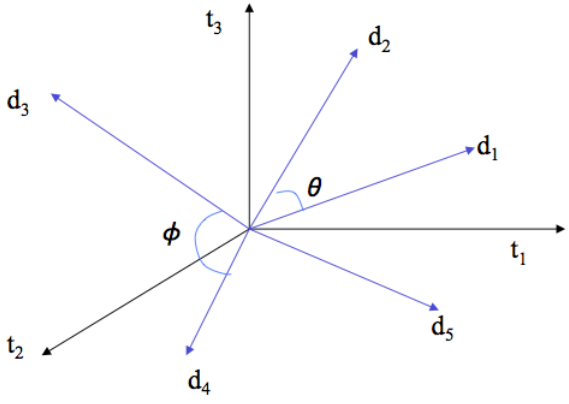
- So we have a $|V|$-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

$$\vec{d}_J = (t1_J, \ldots, tn_j)$$

$\vec{d}_j$ stands for a certain document j, and ti_J stands for a term that occurs in j at position i of the vector.

10.1. Assumption

Documents that are “close together” in the vector space talk about the same thing.

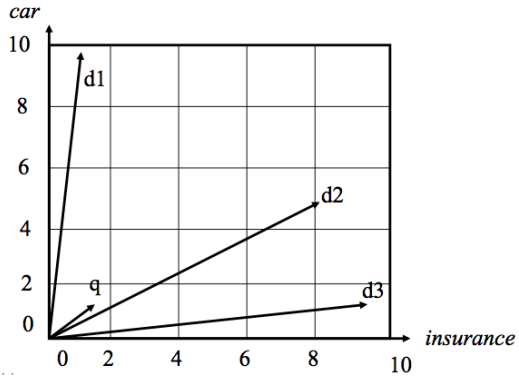


11. Query-document matching scores II

- Key idea 1: Do the same for queries: represent them as vectors in the high-dimensional space, $\vec{q}_k = (t1_k, \ldots, tn_k)$
- Key idea 2: Rank documents according to their proximity to the query

	car	insurance
q =	(1,	1)
d1=	(9,	1)
d2=	(5,	8)
d3=	(1,	9)

Which is the document more similar to *q*?



11.1. Proximity: Angle and Cosine

- Rank documents according to *angle* with query
- Thought experiment: take a document d and append it to itself. Call this document d'. d' is twice as long as d.
- "Semantically" d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity ...
- ...even though the Euclidean distance (which is based on their length) between the two documents can be quite large.

From angles to cosines The following two notions are equivalent.

- Rank documents according to the *angle* between query and document in *decreasing* order
- Rank documents according to *cosine(query,document)* in *increasing order*

11.2. Vectors inner product

Recall: The inner product of two vectors is:

$$\vec{x} \cdot \vec{y} = \sum_i x_i \times y_i$$

		anna,	mario,	apple,	home,	fabio,	paola,	school,	lecture
D1	=	(3,	3,	3,	3,	0,	0,	0,	0)
D2	=	(1,	1,	1,	1,	0,	0,	0,	0)

$\overrightarrow{D1} \cdot \overrightarrow{D2} = \sum_i D1_i \times D2_i = (3 \times 1) + (3 \times 1) + (3 \times 1) + (3 \times 1) = 12$

11.3. Cosine Similarity

$$cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}||\vec{y}|} = \frac{\sum_{i=1}^{n} x_i \times y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}}$$

- x_i is the tf-idf weight of term i in x.
- y_i is the tf-idf weight of term i in y
- $||\vec{x}||$ and $||\vec{y}||$ are the lengths of $\vec{x}$ and $\vec{y}$.
- This is the cosine similarity of $\vec{x}$ and $\vec{y}$ or, equivalently, the cosine of the angle between $\vec{x}$ and $\vec{y}$.

In short, the similarity between two vectors is computed by the cosine of the angle between them.

11.4. Example: Length

D1: Anna and Mario eat the apple at home. Anna and Mario have not an apple at home. Anna and Mario go out of home to buy an apple.

D2: Anna and Mario eat the apple at home.

D3: Fabio and Paola went to school and took a lecture.

		anna,	mario,	apple,	home,	fabio,	paola,	school,	lecture
D1	=	(3,	3,	3,	3,	0,	0,	0,	0)
D2	=	(1,	1,	1,	1,	0,	0,	0,	0)
D3	=	(0,	0,	0,	0,	1,	1,	1,	1)

Recall definition: $||\vec{x}|| = \sqrt{\sum_{i=1}^{n} x_i^2}$

$$
\begin{aligned}
||\overrightarrow{D1}|| &= \sqrt{9+9+9+9} &&= \sqrt{36} &&= 6 \\
||\overrightarrow{D2}|| &= \sqrt{1+1+1+1} &&= \sqrt{4} &&= 2 \\
||\overrightarrow{D3}|| &= \sqrt{1+1+1+1} &&= \sqrt{4} &&= 2
\end{aligned}
$$

11.5. Example: Cosine Similarity

$$cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{||\vec{x}||||\vec{y}||} = \frac{\sum_{i=1}^{n} x_i \times y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}}$$

		anna,	mario,	apple,	home,	fabio,	paola,	school,	lecture
D1	=	(3,	3,	3,	3,	0,	0,	0,	0)
D2	=	(1,	1,	1,	1,	0,	0,	0,	0)
D3	=	(0,	0,	0,	0,	1,	1,	1,	1)

$$cos(\vec{D1},\vec{D2}) = \frac{\vec{D1}\cdot\vec{D2}}{||\vec{D1}|| \times ||\vec{D2}||} = \frac{(3\times 1)+(3\times 1)+(3\times 1)+(3\times 1)}{6\times 2} = \frac{12}{12} = 1$$

$$cos(\vec{D1},\vec{D3}) = \frac{\vec{D1}\cdot\vec{D3}}{||\vec{D1}|| \times ||\vec{D3}||} = \frac{0}{6\times 2} = \frac{0}{12} = 0$$

11.6. Example: Query-Document Matching

Query: “best car insurance”. Document: “car insurance auto insurance”.

word	query					document				product
	tf-raw	tf-wght	df	idf	**weight**	tf-raw	tf-wght	weight	**n’lized**	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n’lized: document weights after cosine normalization, product: the product of final query weight and final document weight

Final similarity score between query and document: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

12. Summary: Ranked retrieval in the vector space model

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top K (e.g., $K = 10$) to the user

13. LT: much more

The processing of the text can be much more elaborate:

- Pos Tagging
- Parsing
- Named Entity Recognition
- From unstructured to structured data
- Ontology Learning
- ...

Some of these tools are useful to build Language Resources encoding the knowledge need to better capture the user information need.

14. Your presentation

Possible topics:

- http://disi.unitn.it/~bernardi/Courses/DL/clir.pdf Cross-Language Information Retrieval
- http://disi.unitn.it/~bernardi/Courses/DL/faceted_search.pdf
- http://disi.unitn.it/~bernardi/Courses/DL/automated_metadata.pdf