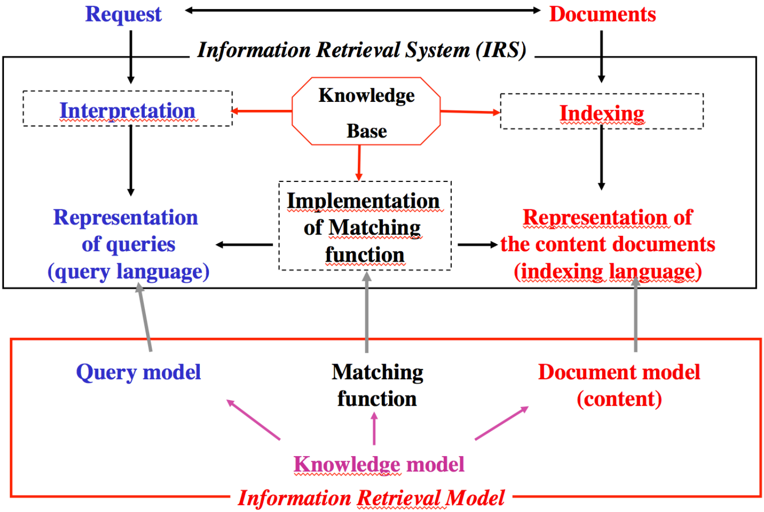
Chap 2: Classical models for information retrieval

Philippe Mulhem & Jean-Pierre Chevallet

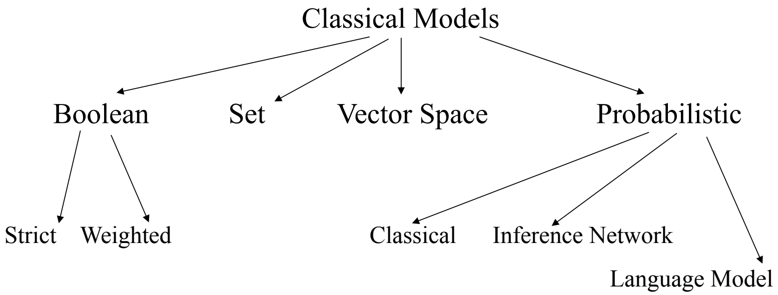
LIG-MRIM

Outline

IR System


Request
Documents
Information Retrieval System (IRS)
Interpretation
Knowledge Base
Indexing
Implementation of Matching function
Representation of queries (query language)
Representation of the content documents (indexing language)
Query model
Matching function
Document model (content)
Knowledge model
Information Retrieval Model

IR Models


Classical Models
Boolean
Set
Vector Space
Probabilistic
Strict
Weighted
Classical
Inference Network
Language Model

Outline

Set Model

Membership of a set

- Queries are single descriptors
- Indexing assignments: a set of descriptors
- In reply to a request, documents are either retrieved or not (no ordering)
- Retrieval rule: if the descriptor in the request is a member of the descriptors assigned to a document, then the document is retrieved (binary retrieval).

This model is the simplest one and describes the retrieval characteristics of a typical library where books are retrieved by looking up a single author, title or subject descriptor in a catalog.

Example

Query

"information retrieval"

Doc1

{"information retrieval", "database" , "salton"}
−− > RETRIEVED < −−

Doc2

{ "database" , "SQL"}
−− > NOT RETRIEVED < −−

Set inclusion

- Query: a set of descriptors
- Indexing assignments : a set of descriptors
- Documents are either retrieved or not (no ordering)
- Retrieval rule : document is retrieved if ALL the descriptors in the request are in the indexing set of the document (binary retrieval).

This model uses the notion of inclusion of the descriptor set of the request in the descriptor set of the document

Set intersection

- Query: a set of descriptors PLUS a cut off value τ
- Indexing assignments : a set of descriptors
- Documents are either retrieved or not (no ordering)
- Retrieval rule : document is retrieved if it shares a number of descriptors with the request that exceeds the cut-off value (binary retrieval).

This model uses the notion of set intersection between the descriptor set of the request with the descriptor set of the document.

Set intersection plus ranking

- Query Q : a set of descriptors PLUS a cut off value τ
- Indexing assignments for a document D: a set of descriptors
- Retrieved documents are ranked

Retrieval rule : documents showing with the request more than the specified number of descriptors are ranked in order of decreasing overlap. Example (Sørensen-Dice overlap coefficient) : $\frac{2.|Q \cap D|}{|Q|+|D|}$

Conclusion on set models

- Very easy to implement, efficient.
- To poor to express complex user relevance.
- Only used in very basic IR systems, or IR extension to DB, like Mongo-DB

Outline

Logical Models: some origins

Fondamental Hypothesis

Matching : a deduction from the query Q to the document D.

Origin

A position paper of C. J. van Rijsbergen 'A Non-Classical Logic for Information Retrieval' [van Rijsbergen, 1986].

Questioning the mixing of logic for deducing a document from a query using some external knowledge, and computing the probability or belief of this deduction.

Logical Models: moste cited Rijsbergen proposal

Rijsbergen's proposal:

Evaluation using the Ramsey test

"To evaluate a conditional, first hypothetically make the minimal revision of your stock of beliefs required to assume the antecedent. Then evaluate the acceptability of the consequent on the basis of this revised body of beliefs.'

Logical Models: which logic to use ?

Based on a given formalized logic:

- Propositional Logic: boolean model $< --$
- First Order Logic : Conceptual Graph Matching
- Modal Logic
- Description Logic: matching with knowledge
- Concept Analysis
- Fuzzy logic
- ...

Very simple example: Boolean Model

- Query is any boolean combination of descriptors using the operators AND ($\wedge$), OR ($\vee$) and NOT ($\neg$)
- Indexing assignments: a set of descriptors
- Retrieved documents are retrieved or not

Retrieval rules:

- if Query $= t_a \wedge t_b$ then retrieve only documents with both t_a and t_b
- if Query $= t_a \vee t_b$ then retrieve only documents with either t_a or t_b
- if Query $= \neg t_a$ then retrieve only documents without t_a.

Boolean Model

Knowledge Model : $T = \{t_i\}, i \in [1, ..N]$

- Term t_i that index the documents

The document model (content) is a Boolean expression in the proposition logic, with the t_i considered as propositions:

- A document $D_1 = \{t_1, t_3\}$ is represented by the logic formula as a conjunction of all terms direct (in the set) or negated. $t_1 \wedge \neg t_2 \wedge t_3 \wedge \neg t_4 \wedge \ldots \wedge \neg t_{N-1} \wedge \neg t_N$
- A query Q is represented by any logic formula
- The matching function is the logical implication: $D \models Q$

Boolean Model: relevance value

No distinction between relevant documents:

- $Q = t_1 \wedge t_2$ over the vocabulary $\{t_1, t_2, t_3, t_4, t_5, t_6\}$
- $D_1 = \{t_1, t_2\} \equiv t_1 \wedge t_2 \wedge \neg t_3 \wedge \neg t_4 \wedge \neg t_5 \wedge \neg t_6$
- $D_2 = \{t_1, t_2, t_3, t_4, t_5\} \equiv t_1 \wedge t_2 \wedge t_3 \wedge t_4 \wedge t_5 \wedge \neg t_6$

Both documents are relevant because : $D_1 \supset Q$ and $D_2 \supset Q$.
We ‘feel” that D_1 is a better response because “closer” to the query.
Possible solution : $D \supset Q$ and $Q \supset D$.

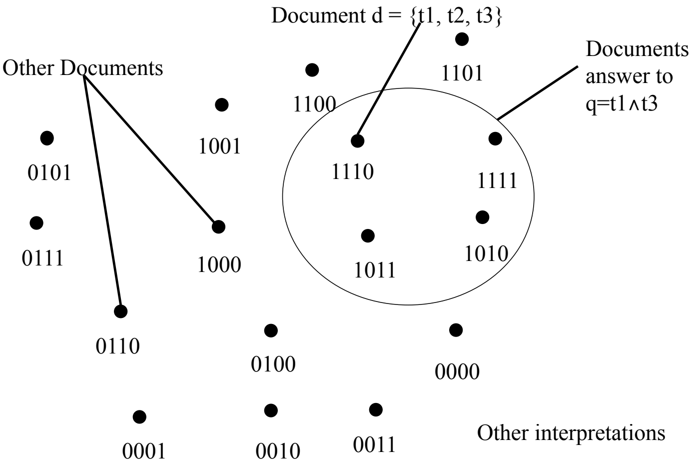
Boolean Model: complexity of queries

- $Q = ((t_1 \wedge t_2) \vee t_3) \wedge (t_4 \vee \neg(\neg t_5 \wedge t_6))$

Meaning of the logical $\vee$ (inclusive) different from the usual "or" (exclusive)

Outline

Interpretations



Beyond boolean logic: matching interpretation

For a logic L, different possible interpretations of matching:

Deduction theory

- $D \vdash_L Q$
- $\vdash_L D \supset Q$

Model theory

- $D \models_L Q$
- $\models_L D \supset Q$

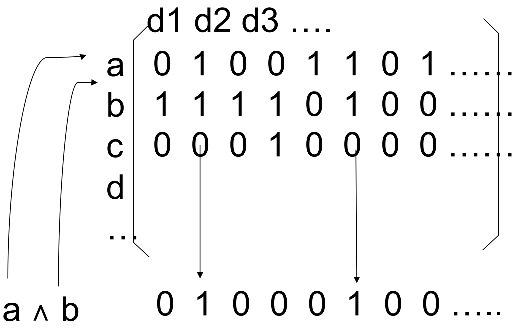
Beyond boolean logic: choice of a logic

A lot of possible choices to go beyond the boolean model. The choice of the underlying logic L to models the IR matching process:

- Propositional Logic
- First order logic
- Modal Logic
- Fuzzy Logic
- Description logic
- ....

Interpretation and Inverted File

Matrix document interpretation can be implemented



Implementing the boolean model

Indexing: same posting list presented in Chap 1.

- List of *unique* doc id associates with an indexing term

Queyring : set (list) fusion algorithms:

- AND = Intersection, OR = union, NOT = complement

List algorithms:

- If no order complexity is $O(n * m)$ (n et m are posting list size)
- Posting list sorted by doc Id. Algorithms depend on this order.
- if order, complexity is $O(n + m)$! So maintaining posting list ordered is important.

Implementing the boolean model

Left to exercise : produces the 3 algorithms: union, intersection and complement, with the following constraints:

- The two lists are read only
- The produced list is a new one and is write only
- Using sorting algorithms is not allowed

Deduce from the algorithms the complexity.

Conclusion for boolean model

- Model used explicitly till the 1990s.
- Very simple to implement
- Implicite usage for almost all search engines : the basic document filtering, then some specific reordering.

Outline

Weighted Boolean Model

Extension of Boolean Model with weights.
Weights denote the representativity of a term for a document.
Knowledge Model : $T = \{t_i\}, i \in [1, ..N]$
Terms t_i that index the documents.
Queries are similar to Boolean model.

A document D is represented by

- A logical formula D (similar to Boolean Model)
- A function $W_D : T \rightarrow [0, 1]$, which gives, for each term in T the weight of the term in D. The weight is 0 for a term not present in the document.

Weighted Boolean Model: matching

Non binary matching function based on Fuzzy logic

- $RSV(D, a \vee b) = Max[W_D(a), W_D(b)]$
- $RSV(D, a \wedge b) = Min[W_D(a), W_D(b)]$
- $RSV(D, \neg a) = 1 - W_D(a)$
- Limitation : this matching using Min and Max does not take all the query terms into account.

Weighted Boolean Model: matching

Non binary matching function based on a similarity function which take more into account all the query terms.

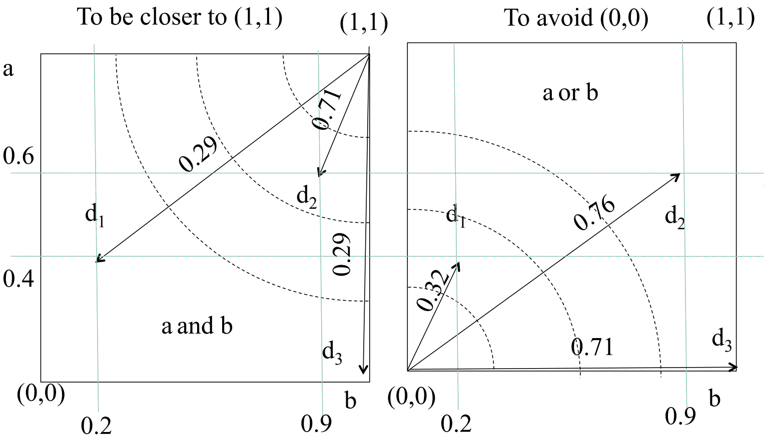
- $RSV(D, a \vee b) = \sqrt{\frac{W_D(a)^2 + W_D(b)^2}{2}}$
- $RSV(D, a \wedge b) = 1 - \sqrt{\frac{(1 - W_D(a))^2 + (1 - W_D(b))^2}{2}}$
- $RSV(D, \neg a) = 1 - W_D(a)$
- Limitation : query expression for complex needs

Weighted Boolean Model: matching example

Document \ Query	a	b	Boolean $a \vee b$	Boolean $a \wedge b$	Weighted Boolean $a \vee b$	Weighted Boolean $a \wedge b$
D_1	1	1	1	1	1	1
D_2	1	0	1	0	$1/\sqrt{2}=0.71$	$1-1/\sqrt{2}=0.29$
D_3	0	1	1	0	$1/\sqrt{2}$	$1-1/\sqrt{2}$
D_4	0	0	0	0	0	0

Weighted Boolean Model: matching example

d1=(0.2, 0.4) , d2=(0.6, 0.9), d3=(0,1)



Outline

Vector Space Model

- Query: a set of descriptors each of which has a positive number associated with it
- Indexing assignments : a set of descriptors each of which has a positive number associated with it.
- Retrieved documents are ranked

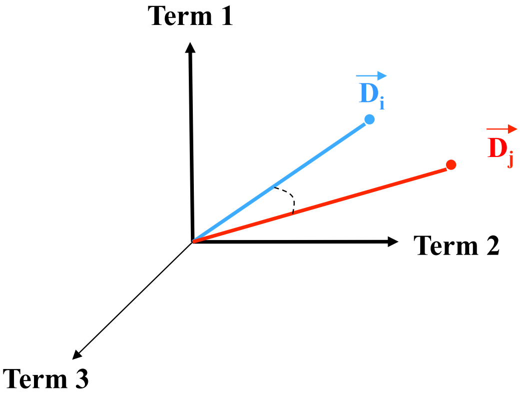
Retrieval rule : the weights of the descriptors common to the query and to indexing records are treated as vectors. The value of a retrieved document is the cosine of the angle between the document vector and the query vector.

Vector Space Model

Knowledge model: $T = \{t_i\}, i \in [1, .., n]$
All documents are described using this vocabulary.
A document D_i is represented by a vector d_i described in the R^n vector space defined on T
$d_i = (w_{i,1}, w_{i,2}, ..., w_{i,j}, ..., w_{i,n})$, with $w_{k,l}$ the weight of a term t_l for a document.
A query Q is represented by a vector q described in the same vector space $q = (w_{Q,1}, w_{Q,2}, ..., w_{Q,j}, ..., w_{Q,n})$

Vector Space Model

The more two vectors that represent documents are ?near?, the more the documents are similar:



Vector Space Model: matching

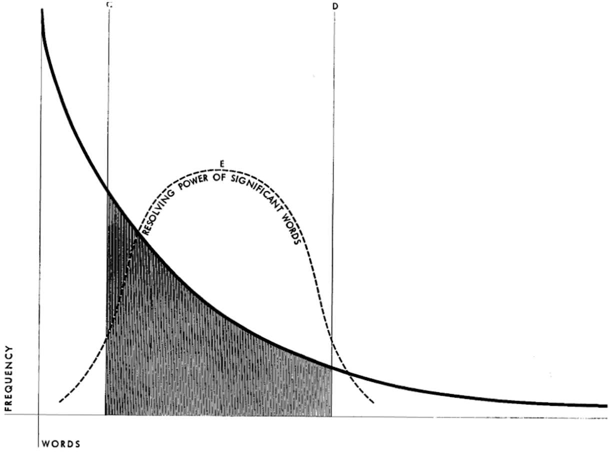
Relevance: is related to a vector similarity.
$RSV(D, Q) =_{def} SIM(\vec{D}, \vec{Q})$

- Symmetry: $SIM(\vec{D}, \vec{Q}) = SIM(\vec{Q}, \vec{D})$
- Normalization: $SIM : V \rightarrow [min, max]$
- Reflectivity : $SIM(\vec{X}, \vec{X}) = max$

Outline

Vector Space Model: weighting

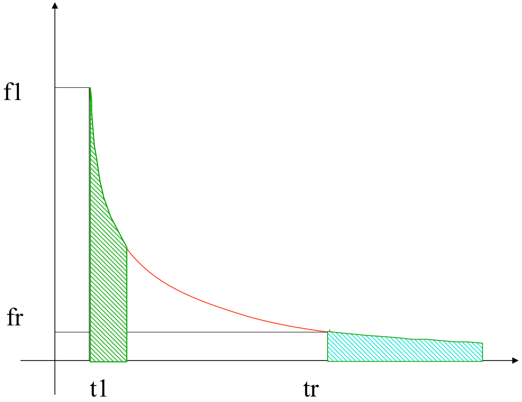
Based on counting the more frequent words, and also the more significant ones.



Zipf Law

Rank r and frequency f:

$$r \times f = const$$

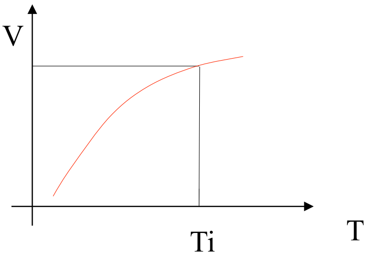


Heap's Law

Estimation of the vocabulary size $|V|$ according to the size $|C|$ (number of words) of the corpus:

$$|V| = K \times |C|^{\beta}$$

for English : $K \in [10, 100]$ and $\beta \in [0.4, 0.6]$ (e.g., $|V| \approx 39\ 000$ for $|C|$=600 000, K=50, β=0.5)

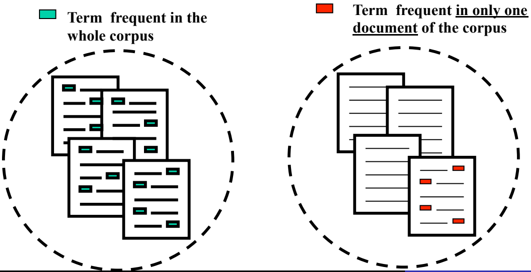


Term Frequency

Term Frequency

The frequency $tf_{i,j}$ of the term t_j in the document D_i equals to the number of occurrences of t_j in D_i.

Considering a whole corpus (document database) into account, a term that occurs a lot does not discriminate documents:



Document Frequency: tf.idf

Document Frequency

The document frequency df_j of a term t_j is the number of documents in which t_j occurs.

The larger the df, the worse the term for an IR point of view... so, we use very often the inverse document frequency idf_j:

Inverse Document Frequency

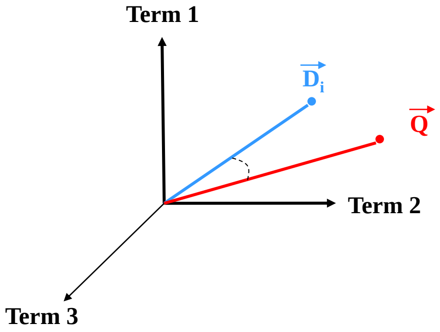
$idf_j = \frac{1}{df_j}$

$idf_j = \log(\frac{|C|}{df_j})$ with $|C|$ is the size of the corpus, i.e. the number of documents.

The classical combination : $w_{i,j} = tf_{i,j} \times idf_j$.

matching

Matching function: based on the angle between the query vector $\vec{Q}$ and the document vector $\vec{D_i}$.
The smaller the angle the more the document matches the query.



matching: cosine

One solution is to use the cosine the angle between the query vector and the document vector.

Cosine

$$SIM_{cos}(\vec{D}_i, \vec{Q}) = \frac{\sum_{k=1}^{n} w_{i,k} \times w_{Q,k}}{\sqrt{\sum_{k=1}^{n} (w_{i,k})^2 \times \sum_{k=1}^{n} (w_{Q,k})^2}}$$

Other matching functions

Dice coefficient

$$SIM_{dice}(\vec{D}_i, \vec{Q}) = \frac{2\sum_{k=1}^{N} w_{i,k} \times w_{Q,k}}{\sum_{k=1}^{N} w_{i,k} + w_{Q,k}}$$

Discrete Dice coefficient

$$SIM_{dice}(\vec{D}_i, \vec{Q}) = \frac{2|D_i^{\{\}} \cap Q^{\{\}}|}{|D_i^{\{\}}| + |Q^{\{\}}|}$$

Outline

Matching cosine simplification

Question

Transform the cosine formula so that to simplify the matching formula.

Cosine

$$SIM_{cos}(\vec{D}_i, \vec{Q}) = \frac{\sum_{k=1}^{n} w_{i,k} \times w_{Q,k}}{\sqrt{\sum_{k=1}^{n} (w_{i,k})^2 \times \sum_{k=1}^{n} (w_{Q,k})^2}}$$

Exercice: Similarity and dissimilarity and distance

The Euclidian distance between point x and y is expressed by :

$$L_2(x, y) = \sum_i (x_i - y_i)^2$$

Question

Show the potential link between L_2 and the cosine.

Tips:

- consider points as vectors
- consider document vectors has normalized, i.e. with $\sum_i y_i^2$ as constant.

Exercice: dot product and list intersection

The index is usually very sparse: a (usefull) term appears in less than 1% of document.

A term t has a non null weight in $\vec{D}$ *iff* t appears in document D

Question

Show that the algorithm for computing the dot product is equivalent to list intersections.

Outline

Link between dot product and inverted files

If $RSV(x,y) \propto \sum_i x_i y_i$ after some normalizations of y, then :

$$RSV(x,y) \propto \sum_{i, x_i \neq 0, y_i \neq 0} x_i y_i$$

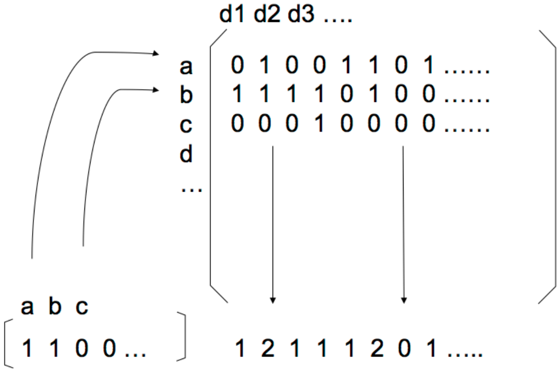
- Query: just to proceed with terms that are in the query, i.e. whose weight are not null.
- Documents: store only non null terms.
- Inverted file: access to non null document weight for for each term id.

The index

- Use a Inverted file like Boolean model
- Store term frequency in the posting list
- *Do not pre-compute* the weight, but keep raw integer values in the index
- Compute the matching value on the fly, i.e. during posting list intersection

Matching: matrix product

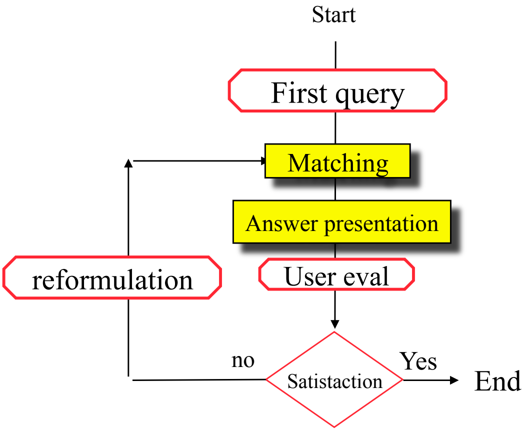
With an inverted file, in practice, matching computation is a matrix product:



Outline

Relevance feedback

To learn system relevance from user relevance:



Rocchio Formula

Rocchio Formula

$$\vec{Q}_{i+1} = \alpha \vec{Q}_i + \beta \vec{Rel}_i - \gamma n\vec{Rel}_i$$

With:

- $\vec{Rel}_i$: the cluster center of relevant documents, i.e., the positive feedback
- $n\vec{Rel}_i$: the cluster center of non relevant documents, i.e., the negative feedback

Note : when using this formula, the generated query vector may contain negative values.

Conclusion

Common step for indexing:

- Define index set
- Automatize index construction from documents
- Select a model for index representation and weighting
- Define a matching process and an associated ranking

This is used for textual processing but also for other media.

References

van Rijsbergen, C. J. (1986).
A non-classical logic for information retrieval.
Comput. J., 29(6):481–485.

Outline

Solution: matching cosine simplification

Because Q is constant, norm of vector Q is a constant that can be removed.

$$K = \frac{1}{\sum_{k=1}^{n} (w_{Q,k})^2}$$

$$SIM_{cos}(\vec{D}_i, \vec{Q}) = K \frac{\sum_{k=1}^{n} w_{i,k} \times w_{Q,k}}{\sqrt{\sum_{k=1}^{n} (w_{i,k})^2}}$$

Solution: matching cosine simplification

The division of the document vector by its norm can be precomputed:

$$w'_{i,k} = \frac{w_{i,k}}{\sqrt{\sum_{k=1}^{n} (w_{i,k})^2}}$$

Solution: matching cosine simplification

Hence, one juste has to compute a vector dot product:

$$SIM_{cos}(\vec{D}_i, \vec{Q}) = K \sum_{k=1}^{n} w'_{i,k} \times w_{Q,k}$$

The constant does not influence the order:

$$RSV_{cos}(\vec{D}_i, \vec{Q}) \propto \sum_{k=1}^{n} w'_{i,k} \times w_{Q,k}$$

Solution: Similarity et dissimilarity and distance

To transform a distance or dissimilarity into a similarity, simply negate the value. Ex. with the Squared Euclidian distance :

$$RSV_{D_2}(x,y) = -D_2(x,y)^2 = -\sum_i (x_i - y_i)^2$$

$$= -\sum_i (x_i^2 + y_i^2 - 2x_i y_i) = -\sum_i x_i^2 - \sum_i y_i^2 + 2\sum_i x_i y_i$$

if x is the query then $\sum_i x_i^2$ is constant for matching with a document set.

$$RSV(x,y)_{D_2} \propto -\sum_i y_i^2 + 2\sum_i x_i y_i$$

Solution: Similarity et dissimilarity and distance

$$RSV(x,y)_{D_2} \propto -\sum_i y_i^2 + 2\sum_i x_i y_i$$

If $\sum_i y_i^2$ is constant over the corpus then :

$$RSV(x,y)_{D_2} \propto 2\sum_i x_i y_i$$

$$RSV(x,y)_{D_2} \propto \sum_i x_i y_i$$

Hence, if we normalize the corpus so each document vector length is a constant, then using the Euclidean distance as a similarity, provides the same results than the cosine of the vector space model !

Solution: dot product and list intersection

In the dot product $\sum_i x_i y_i$, if the term at rank i is not in x or y then the term $x_i y_i$ is null and does not participate to the value. Hence,

If we compute directly $\sum_i x_i y_i$ using a list of terms for each document, then we sum the $x_i y_i$ only for terms that are in the intersection.

Hence, it is equivalent to an intersection list.

Solution: efficiency of list intersection

If X and Y are not sorted, for each x_i one must find the term in Y by sequential search. So complexity is $\mathcal{O}(|X| \times |Y|)$
If X and Y are sorted, the algorithm uses two pointers:

- We start from the begining of the head of both lists
- Lets compare the pointers :
  - If terms are equal : we can compute the merge (or the product), then move the tow pointers
  - It terms are different : because of the order, one are sure the smaller term to net be present in the other list.

Hence one move at least one pointer on each list, the the complexity is : $\mathcal{O}(|X| + |Y|)$