Naming Functions for the Vector Space Model

Yannis Tzitzikas and Yannis Theoharis

Computer Science Department, University of Crete, GREECE, and
Institute of Computer Science, FORTH-ICS, GREECE
email: {tzitzik, theohari}@ics.forth.gr

Abstract. The Vector Space Model (VSM) is probably the most widely used model for retrieving information from text collections (and recently from over other kinds of corpi). Assuming this model, we study the problem of finding the best query that "names" (or describes) a given (unordered or ordered) set of objects. We formulate several variations of this problem and we provide methods and algorithms for solving them.

1 Introduction

This paper elaborates on the "naming problem" i.e. the problem of finding a query (or the best query) that describes (names) a given (unordered or ordered) set of objects. The motivation for studying this problem is that several variations of it are useful in a number of tasks (e.g. for relevance feedback, document representation learning). In addition, naming functions can be exploited for providing flexible information access services, for instance, they can be exploited for realizing the context-based interaction scheme described in [9].

More precisely, naming functions can be exploited for supporting an alternative query formulation process. The user selects a number of objects (i.e. a set A') and asks the system to formulate the query that "describes" these objects. The system returns a query q' and subsequently the user can change it in a way that reflects his/her information need. Roughly this resembles the Query By Example (QBE) process in relational databases. It also resembles the relevance feedback mechanisms in IR systems. For example, consider a user who has formulated a query q and has been given an answer A. She can select a subset A' of the current answer A consisting of those elements of A which she finds relevant to her information need. Subsequently, the system has to change appropriately the query. Notice that here the user is not directly driven to the answer of the new query (as it is done in classical relevance feedback mechanisms). Instead, she is given the new query, so she has the opportunity to modify it before submitting it for evaluation. Probably this (intermediate step) could alleviate the problem of unexpected results that frequently occur after a feedback cycle. Notice that if $A' \subseteq A$ then it is like giving explicit negative relevance feedback (to the elements of $A - A'$), and "tacit" positive relevance feedback (to the elements of A'). However, this interaction scheme is more general than the classical feedback mechanisms of IR as it allows giving positive feedback to documents not already in the current answer. In addition, naming functions can help users to be attuned

with the indexing and query language of the system. It is not hard to foresee additional applications of naming functions, in building mediators (metasearchers) and in cross-language retrieval. For these reasons, we believe that the naming problem deserves separate attention and analytical elaboration.

The rest of this paper is organized as follows. Section 2 states the naming problem and Section 3 provides solutions assuming the vector space model. Section 4 reports some experimental results. Finally, Section 5 concludes the paper and identifies issues for further research.

2 The Naming Problem

An IR system S, hereafter source, can be viewed as a function $S : Q \rightarrow \mathcal{A}$ where Q is the set of all queries that S can answer, and $\mathcal{A}$ is the set of all answers to those queries, i.e. $\mathcal{A}$=$\{\ S(q) \mid q \in Q\}$. As the focus is given on retrieval queries, it is assumed that $\mathcal{A}$ is a subset of $\mathcal{P}(O)$, the powerset of O, where O is the set of all objects stored at the source. In order to capture sources that return sets of ranked objects (based on best-match retrieval models), we shall use B's to denote ordered answers (and A's for unordered ones). Given an ordered set B, we use $B(k)$ to denote the first k elements of B and $B\{k\}$ to denote the *set* of elements that appear in $B(k)$. We shall also use the notation $\{B\}$ to denote the *set* of elements that appear in B (in other words $\{B\} = B\{|B|\}$). So if $B = \langle o_1, o_2, o_3\rangle$ and $B' = \langle o_3, o_2, o_1\rangle$, then we can write $\{B\} = \{B'\}$. Furthermore, we use $B_{|A}$ to denote the restriction of B on the set A, i.e. the ordered set obtained if we exclude from B those elements that do not belong to A. Similarly, $\{B\}_{|A}$ denotes the restriction of $\{B\}$ on the set A, i.e. the unordered set obtained if we exclude from $\{B\}$ those elements that do not belong to A.

2.1 The Naming Problem for Unordered Sets

Def. 1 A query q is an *exact name of* A and we write $q = n(A)$, if $S(q)\{|A|\} = A$.

This means that the first $|A|$ elements of $S(q)$ are occupied by elements of A.

Def. 2 A query q is the best *upper name of* A and we write $q = n^+(A)$, if there exists $m \geq |A|$, such that $S(q)\{m\}_{|A} = A$, where m is the minimum possible.

This means that the restriction of the set of the first m ($m \geq |A|$) elements of $S(q)$ over A is A.

Def. 3 A query q is the best *lower name of* A and we write $q = n^-(A)$, if there exists $0 < m \leq |A|$, such that $S(q)\{m\} \subseteq A$, where m is the maximum possible.

This means that the first m ($m \leq |A|$) elements of $S(q)$ are occupied by elements of A. However, more often than not, non of these three names exists. For this reason below we define a more relaxed definition, which generalizes both *upper* and *lower* name definitions.

Def. 4 A query q is a *relaxed name of* A and we write $q = \widetilde{n}(A)$, if $|S(q)\{m\} \cap A| = j$, where $m \geq j > 0$.

If $m = j = |A|$ then q is an exact name. If $m > j = |A|$, then q is an upper name (the best one, if m is the least possible). If $m = j < |A|$, then q is a lower name (the best one, if m is the greatest possible).

2.2 The Naming Problem for Ordered Sets

Here we elaborate on the naming problem for ordered sets. Specifically, we consider weakly ordered sets as this captures both sets and linearly ordered sets. We use $\langle o_1, o_2, o_3 \rangle$ to denote a linearly ordered answer where o_1 is the most relevant object, and $\langle o_1, \{o_2, o_3\}, o_4 \rangle$ to denote a weakly ordered answer where o_2 and o_3 are equally relevant (e.g. they have the same degree of relevance).

Def. 5 Let B be a weakly ordered set. A query q is:

- an *exact name* of B (and we write $q = n(B)$), if $S(q)(|\{B\}|) = B$.
- the best *upper name* of B (and we write $q = n^{+}(B)$), if $S(q)(m)_{|\{B\}} = B$, where m is the minimum possible.
- the best *lower name* of B (and we write $q = n^{-}(B)$), if $S(q)(m) = B(m)$, where m is the maximum possible (and $m > 0$).
- a *relaxed name* of B (and we write $q = \widetilde{n}(B)$), if $S(q)(j)_{|\{B(m)\}} = B(m)$, where m is the maximum and j is the minimum possible (and $m > 0$).

3 Naming Functions for the Vector Space Model

The Vector Space Model (VSM) is one of the classical models for information retrieval over text collections and has been recently used for information retrieval over XML documents [2] and ontology-based sources [3]. Let $T = \{t_1, \ldots, t_k\}$ be the set of different words that appear in the documents of the collection O (we ignore stemming, stopwords). According to the VSM, the degree of relevance equals the cosine of the angle between the vectors $\boldsymbol{q}$ and $\boldsymbol{o_j}$.

3.1 Naming Functions for Unordered Sets

As one can imagine, it is very difficult to find the upper name for a given set of objects assuming the classical index structures that IR systems and Web search engines currently use (i.e. inverted files). However, several methods could be used to find an approximate upper name, the most well known example being the best query of the Rocchio method [6]. To be more specific, below we list a number of conditions that a candidate query q could satisfy:
(a) minimizes the maximum distance between q and an element of A,
(b) minimizes the average distance between q and the elements of A,
(c) is the result of subtracting from a query q_1 that satisfies (b) a query q_2 that minimizes the average distance between itself and all those elements not in A.

A query that satisfies (b) is $q = \frac{1}{|A|}\sum_{d\in A} d$. A query that satisfies (c) is $q = \frac{1}{|A|}\sum_{d\in A} d - \frac{1}{|Obj\setminus A|}\sum_{d\notin A} d$, what is called "best query" on which the standard Rocchio method for relevance feedback is based on. A query that satisfies (a) is $q = \frac{1}{2}(a_1 + a_2)$, where a_1, a_2 are the most distant (i.e., less similar) documents of A.

Note that none of these methods guarantees that the resulting query is the best upper name (as defined in Def. 2). In particular, (a) can be better than (b), as depicted in Figure 1 (left). In this case, A comprises 3 elements, i.e., a_1, a_2 and a_3. y is an element that does not belong to A. o is the query vector that method (b) yields, while u is the one that method (a) yields. Obviously, o will rank documents in the rank $\langle a_2, a_1, y, a_3\rangle$, while u will rank them as follows, $\langle a_2, \{a_1, a_3\}, y\rangle$. Also, (a) can be better than (c), as depicted in Figure 1 (right). In that case y_1, y_2 are documents not in A, o is the query vector that method (c) yields, while u is the one that method (a) yields. Note that the vector $q_2 = y_1 + y_2$ (i.e., $\sum_{d\notin A} d$) has the same direction as $q_1 = \frac{1}{3}(a_1 + a_2 + a_3)$ (i.e., $\sum_{d\in A} d$) and thus the resulting query o will rank the documents in the rank $\langle a_2, a_1, \{y_1, y_2\}, a_3\rangle$, while u will rank them as follows, $\langle y_2, a_2, \{a_1, a_3\}, y_1\rangle$. The latter is a better upper name than the former according to Def 2, because it has m=4, while the former has m=5.

We should stress that the query yielded from method (a) can be evaluated more efficiently than the queries yielded by the other two methods, since it only comprises the words of the two most distant documents.

Methods $(b), (c)$ yield a relaxed name. Specifically, and with respect to Def. 4, we can say that method (b) attempts to maximize j, irrespective of the value of m. On the other hand, method (c) attempts to maximize j and minimize $m - j$ simultaneously. Hence, method (c) yields a better approximation of the best lower name than (b).

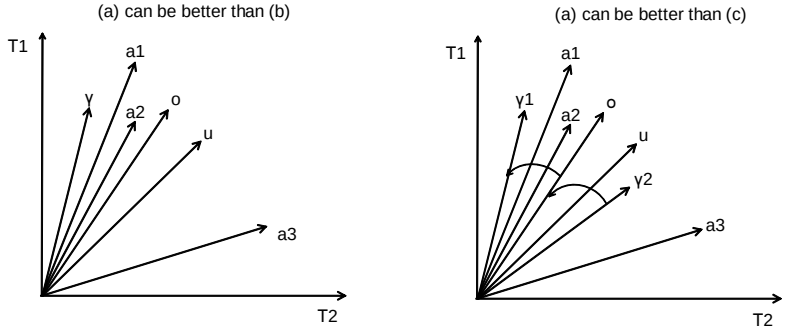

Fig. 1. Different methods to approximate the upper name

Bellow, we elaborate on the method (a) to approximate the upper name. Let a_1 and a_2 be the most distant (less similar) elements of A, i.e. $\cos(\boldsymbol{a_1}, \boldsymbol{a_2}) = \min(\{cos(\boldsymbol{a}, \boldsymbol{b}) \mid a, b \in A\})$. Since the measure of similarity equals the cosine between the corresponding vectors and the maximum angle between two document vectors equals $\frac{\pi}{2}$ (because the vector coordinates are positive) and cosine

is monotonically decreasing (with maximum value at 0 and minimum at $\frac{\pi}{2}$), it follows that the most distant documents will be those that form the bigger angle. If there exist more than one such pairs a_1, a_2, then we choose arbitrarily one of them. The two less similar vectors of A define the $area(\boldsymbol{a_1}, \boldsymbol{a_2})$, which is the geometric space of all vectors of $[0,1]^{|T|}$ such that $\forall b \in area(\boldsymbol{a_1}, \boldsymbol{a_2})$, $|(\widehat{\boldsymbol{b}, \boldsymbol{a_1}})| < |(\widehat{\boldsymbol{a_1}, \boldsymbol{a_2}})|$ and $|(\widehat{\boldsymbol{b}, \boldsymbol{a_2}})| < |(\widehat{\boldsymbol{a_1}, \boldsymbol{a_2}})|$. As one can easily see, every query $q \in area(\boldsymbol{a_1}, \boldsymbol{a_2})$ ranks each document of $area(\boldsymbol{a_1}, \boldsymbol{a_2})$ higher than either a_1 or a_2. The query that minimizes the maximum distance between q and an element of A is the vector $\boldsymbol{q_A} = \frac{1}{2}(\boldsymbol{a_1} + \boldsymbol{a_2})$ whose direction is that of the vector that corresponds to the composed force of $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$. So we can decide whether there is an exact name in two steps:

(1) We find the vectors $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$, i.e.:

$$(\boldsymbol{a_1}, \boldsymbol{a_2}) = \arg_{\boldsymbol{a}, \boldsymbol{a'}} min(\{cos(\boldsymbol{a}, \boldsymbol{a'}) \mid a, a' \in A\})$$

(2) we check if there are documents in $area(\boldsymbol{a_1}, \boldsymbol{a_2})$ that do not belong to A and how many they are. Let denote with n the number of these documents. If $n = 0$, then the exact name exists and it equals to q_A, i.e. $n(A) = q_A$. Otherwise, i.e. if $n > 0$, then there is not an exact name, because, since $q_A \in area(\boldsymbol{a_1}, \boldsymbol{a_2})$, it ranks each of these n documents of $O \setminus A$ higher than either a_1 or a_2.

Clearly, the cost of Step (1) is $\mathcal{O}(|A|^2)$, while the second step of the algorithm costs $\mathcal{O}(|O|)$ time, since we need to iterate over the whole collection in the worst case. However, in most cases (and with the availability of an index) we can just evaluate the query q_A and see if the first $|A|$ elements of the answer is A.

Let's now consider the case where there is not an exact name. In that case we would like to find an upper name and a lower name (ideally the best ones). Note that q_A (as defined earlier) will not rank first the elements of A, however it will rank the elements of A quite high. Specifically, it will be $S(q_A)\{|A| + n\}_{|A} = A$, i.e. q_A is an upper name of A. One can easily see that q_A is definitely the best upper name because for any other query q, if $S(q)\{j\}_{|A} = A$ then certainly $j \geq |A| + n$.

Finding the best lower name means finding a query q that ranks first the biggest number of elements of A, i.e. finding the maximum m ($m \leq |A|$) such that $S(q)\{m\} \subseteq A$. One expensive solution that would certainly yield the lower name would be to find the biggest subset of A that has an exact name (using the method presented earlier). The time complexity of this method is exponential with respect to $|A|$.

Another method is presented next. For each pair $b, b' \in (O \setminus A) \bigcap area(\boldsymbol{a_1}, \boldsymbol{a_2})$ we first investigate the $area(\boldsymbol{b}, \boldsymbol{b'})$: we want only elements of A to be there and if this is the case we compute their number. At the end we select the pair that gave the bigger number and consider as lower name the "composed force" of that pair[1]. The time complexity of this method is $\mathcal{O}(|O|^3)$, since all possible pairs of

[1] If there were an element not in A, then the result of this query would contain it.

(b, b') are $|O|^2$ and for such pair the method iterates over the whole collection in the worst case. Again a multidimensional index would greatly reduce the cost of computing $(O \setminus A) \bigcap area(\boldsymbol{a_1}, \boldsymbol{a_2})$ as well as the cost for computing $area(\boldsymbol{b}, \boldsymbol{b'})$.

3.2 Naming Functions for Ordered Sets

A common subproblem implied by all variations of the naming problem is to investigate how many documents of B are topologically located in the "right" order. For instance, if $B = \langle a_1, a_2, a_3 \rangle$ and a_3 rests in $area(\boldsymbol{a_1}, \boldsymbol{a_2})$, then there doesn't exist an exact name for B. Let a_i be the i-th in order document of B (a_1 stands for the most relevant and $a_{|\{B\}|}$ for the less relevant document of B), i.e. $B = \langle a_1, ..., a_{|B|} \rangle$. It is evident that a relaxed name exists, if $\exists i$ s.t. $2 \leq i \leq |B|$ and $sim(\boldsymbol{a_1}, \boldsymbol{a_2}) \geq ... \geq sim(\boldsymbol{a_1}, \boldsymbol{a_i})$. In that case a_1 is the relaxed name of B. If $i = |B|$ then a_1 is an upper or an exact name of B. Specifically, it is an exact name of B, if $S(a_1)(|B|) = B$, otherwise it is an upper name of it. Finally, if $i < |B|$ and $S(a_1)(i) = B(i)$, then a_1 is a lower name of B. The algorithm *LinearlySorted*, shown below, returns the maximum i.

The complexity of Alg. *LinearlySorted* is $\mathcal{O}(|B|)$. To check whether a_1 is a lower or upper or exact name, a query evaluation step is needed. If B is not a linear ordered but a weakly ordered set, e.g. $\langle \{a_1, a_2\}, a_3 \rangle$, then we treat the class of the most highly ranked elements in B (in our example the set $\{a_1, a_2\}$) as we did in Section 3.1, i.e. we try to find an exact name of it. This results in a query $q_{\{a_1,a_2\}}$ that is either an exact or an upper name. Then, we replace in B the set $\{a_1, a_2\}$ with $q_{\{a_1,a_2\}}$ and use the algorithm *LinearlySorted* with some slight modifications. Specifically, we view B as a linear order of sets, i.e. $B = \langle c_1, ..., c_j \rangle$, where $c_i \subseteq O$ and let $|B| = j$. The variation of the algorithm *LinearlySorted* for this case is the algorithm $WeaklySorted$ shown below.

Alg. *LinearlySorted*	Alg. $WeaklySorted$
Input: B	Input: $B = \langle c_1, ..., c_j \rangle$
(1) $M := 1$	(1) $M := 1$
(2) for $i = 2$ to $\|\{B\}\|$ do	(2) for $i = 2$ to $\|B\|$ do
(3) if $sim(\boldsymbol{a_1}, \boldsymbol{a_i}) > M$	(3) $X := max(\{sim(n(c_1), a_k) \mid a_k \in c_i\})$
(4) then return i-1	(4) if $X > M$ then return i-1
(5) else $M := sim(\boldsymbol{a_1}, \boldsymbol{a_i})$	(5) else $M := X$
(6) return i	(6) return i

4 Experimental Evaluation

We have implemented naming functions on top of the experimental Web Search Engine, *GRoogle* (http://groogle.csd.uoc.gr:8080/groogle/), in order to investigate whether our approach can be applied on large collections. Experiments were carried out on a "normal" PC (Pentium IV 3.2GHz, 512MB RAM, Suse Linux v9.1). We used a collection crawled from *www.csd.uoc.gr* and *www.ics.forth.gr*. We considered two subsets of it, one consisting of 1,000 documents and 40,000 terms and another consisting of 5,000 documents and 250,000 terms. In order to

select the documents that play the role of the answer set A (or B for ordered sets), we formulated a random query and defined A and B using the first 10/100 documents. We repeated our experiment 10 times and computed the average times. Recall that the computation of naming functions comprises two steps: a) find the name query, b) find what kind of name (upper/exact) this query is. The second step requires evaluating the name query, and clearly this depends on the underlying IR system. Since the name computed at the first step is certainly an upper name for the case of unordered sets (and relaxed name for the case of ordered sets), it is reasonable to measure separately the first step. Also, note that for ordered sets we always consider the first document of B as name and thus, the cost of the first step is negligible. Table 1 reports the execution times (t_a denotes the time to find the query, t_b the time to evaluate it and t_c the time to identify what kind of name the query is). Recall that T denotes the vocabulary of the collection. The main conclusion drawn is that the most costly task is to evaluate the name query in order to decide whether it is an upper/exact name. For this reason and in order to show the effect of the length of the computed name, we evaluated it once by considering only the 5 better (i.e., more weighted) terms and once more with the 10 better terms. Issues for reducing the query evaluation time are beyond the scope of this paper. From the results of the evaluation we can say that our method is efficient. Specifically, the cost of the first step depends only on the size of A and not on the collection. Concerning the effect of the vocabulary size, the cost of the first step is independent of it, more precisely it depends on the vocabulary of the documents in A (for the computation of the inner product of two vectors we only consider the non-zero coordinates of the vectors, which are usually much less than the vocabulary size).

Table 1. Execution times for computing names of unordered and ordered sets

Collection			Naming Functions (in sec)					
			Unordered			Ordered		
$\|O\|$	$\|T\|$	$\|\{A\}\|$	t_a	$t_b(query\ terms)$	t_c	t_a	$t_b(query\ terms)$	t_c
1K	40K	10	0.015	1.566 (5) - 3.174 (10)	0.001	0	1.878 (5) - 3.575 (10)	0.008
1K	40K	100	0.328	1.56 (5) - 3.176 (10)	0.005	0	1.624 (5) - 3.192 (10)	0.004
5K	255K	10	0.015	112.1 (5) - 262.5 (10)	0.001	0	131.2 (5) - 264.7 (10)	0.048
5K	255K	100	0.328	116.0 (5) - 251.1 (10)	0.006	0	153.4 (5) - 271.1 (10)	0.152

5 Concluding Remarks

Naming functions for taxonomy-based sources were originally proposed in [8] and were subsequently exploited for defining a flexible interaction scheme for information sources [9]. However, they were defined only for sources that return sets of objects. In this paper we formulated the naming problem for sources that return ordered sets of objects. Subsequently, we provided algorithms for computing naming functions appropriate for systems that are based on the vector space model. Somehow related work includes [4, 7, 1].

Although we provided polynomial algorithms for the naming function problem, they are by no means optimal and a lot of work has to be devoted for finding more efficient algorithms and appropriate indexing structures that would allow the application of this interaction scheme on large corpi of documents. For instance, indexes like [5] could be exploited for speeding up the computation of naming functions.

Another issue for further research and experimental investigation is how we could shorten the usually long queries that are computed by the naming functions. For instance, a user might prefer to receive only short queries, e.g. queries that contain at most k terms. One method to address this requirement would be to reformulate the naming problem by introducing one additional parameter (i.e. the maximum number of terms desired). An alternative approach (followed in the Section 4) would be to present to the user only the best k terms of the queries that the current naming functions compute, i.e. the k terms having the highest weights. The relative evaluation of these two approaches is an open issue.

Acknowledgement: This work was partially supported by the EU project CASPAR (FP6-2005-IST-033572).

References

1. R. S. Bot and Y. B. Wu. "Improving Document Representations Using Relevance Feedback: The RFA Algorithm". In *Procs of the 13th ACM intern. Conf. on Information and Knowledge Management*, Washington, USA, 2004.
2. Vinay Kakade and Prabhakar Raghavan. "Encoding XML in Vector Spaces". In *Proceedings of the 27th European Conference on Information Retrieval*, Santiago de Compostela, Spain, 2005.
3. Latifur Khan, Dennis McLeod, and Eduard Hovy. "Retrieval effectiveness of an ontology-based model for information selection". *The International Journal on Very Large Data Bases*, 13(1):71–85, January 2004.
4. Massimo Melucci. "Context Modeling and Discovery Using Vector Space Bases". In *Proceedings of the 14th ACM international Conference on Information and Knowledge Management*, Bremen, Germany, 2005.
5. G. Qian, S. Sural, Y. Gu, and S. Pramanik. Similarity between Euclidean and cosine angle distance for nearest neighbor queries. *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1232–1237, 2004.
6. J.J. Rocchio. "Relevance Feedback in Information Retrieval". In G. Salton, editor, *The SMART Retrieval System*. Prentice Hall, Englewood Cliffs, NJ, 1971.
7. Xuehua Shen, Bin Tan, and ChengXiang Zhai. "Implicit User Modeling for Personalized Search". In *Proceedings of the 14th ACM international Conference on Information and Knowledge Management*, Bremen, Germany, 2005.
8. Y. Tzitzikas and C. Meghini. "Ostensive Automatic Schema Mapping for Taxonomy-based Peer-to-Peer Systems". In *7th Intern. Workshop on Cooperative Information Agents, CIA-2003*, pages 78–92, Helsinki, Finland, August 2003.
9. Y. Tzitzikas, C. Meghini, and N. Spyratos. "A Unified Interaction Scheme for Information Sources". *J. of Intelligent Information Systems*, 26(1):75–93, Jan. 2006.