The Query-Vector Document Model

Diego Puppin, Fabrizio Silvestri
ISTI-CNR, HPC Laboratory
Via Moruzzi 1, Pisa, Italy
{diego.puppin, fabrizio.silvestri}@isti.cnr.it

Categories & Subject Descriptors: H.3.1 Content Analysis and Indexing
General Terms: Algorithms
Keywords: Document Partitioning, Collection Selection, Document Model

Introduction. Modern Web IR systems have to manage collections of billions of documents. The indexes used to represent them are very large data structures, the form of which can have a big impact on the quality and the speed of IR algorithms. Traditionally, two main ways are used to model the documents available: the *bag-of-words* model, and the *vector-space* model.

In the *query-vector* document model, documents are modeled with the list of queries they match, along with the *rank* they get for each. The query-vector representation of a document is built out of a query-log. A reference search engine is used in the building phase: for every query in the training set, the system stores the first 100 results along with their rank. This creates a matrix, with documents on columns and queries on rows, where each entry is the rank of a document for a given query.

To be more precise, let Q be a query log containing queries $q_1, q_2, \ldots, q_m$. Let $d_{i1}, d_{i2}, \ldots, d_{in_i}$ be the list of documents returned as results to query q_i. Furthermore, let r_{ij} be the rank that document d_j gets as result of query q_i (0 if the document is not a match).

A document d_j is represented as an m-dimensional vector $\overline{d_j} = [\overline{r_{ij}}]^T$, where $\overline{r_{ij}} \in [0, 1]$ is the normalizated value of r_{ij}: $\overline{r_{ij}} = r_{ij} / \sum_{ij} r_{ij}$.

The $\overline{r_{ij}}$ values form a contingency matrix R (proof in [4]), where we can perform the co-clustering algorithm by Dhillon et al. [3]. This approach creates, simultaneously, clusters of rows (queries) and columns (documents) out of an initial matrix, with the goal of minimizing the loss of information. The result of co-clustering is a matrix $\widehat{P}$ defined as:

$$\widehat{P}(qc_a, dc_b) = \sum_{i \in qc_b} \sum_{j \in dc_a} \overline{r_{ij}}$$

In other words, each entry $\widehat{P}(qc_a, dc_b)$ sums the contributions of $\overline{r_{ij}}$ for the queries in the query cluster a and the documents in document cluster b. We call this matrix simply PCAP. The values of PCAP are important because they measure the relevance of a document cluster to a given query cluster.

Document Clustering and Collection Selection. We use our model to perform document clustering. This is used as a central part of a distributed, document-partitioned information retrieval system. Instead of partitioning the documents randomly and broadcasting queries to all sub-collections, we do a guided partioning, and we use a collection selection strategy to query only a few sub-collections.

We compared different approaches to partitioning:
(1) **random**: a random allocation;
(2) **shingles**: documents' signature were computed using shingling[1], on which we used the standard k-means;
(3) **URL-sorting**: it is a very simple heuristics, which assign documents block-wise, after sorting them by their URL;
(4) **k-means**: k-means on the query-vector representation;
(5) **co-clustering**: we used the algorithm from [3] to compute documents and query clusters.

We used the CORI [2] technique to perform collection selection in all cases. In the last case we could also use a novel technique based on the results of co-clustering, as follows.

The queries belonging to each query cluster are joined together into *query dictionary* files. Each dictionary file stores the text of each query belonging to a cluster, as a single text file. When a new query q is submitted to our IR system, we use the TF.IDF metric to find which clusters are the best matches: each dictionary file is considered as a document, which is indexed with the usual TF.IDF technique. This way, each query cluster qc_i receives a score relative to the query q, say $r_q(qc_i)$.

This is used to weight the contribution of PCAP $\widehat{P}(i, j)$ for the document cluster dc_j, as follows:

$$r_q(dc_j) = \sum_i r_q(qc_i) \times \widehat{P}(i, j)$$

Experimental Results. We performed all our test using the WBR99 collection. WBR99 consists of 5,939,061 documents documents. We could use also a query log for the period January through October 2003.

Following previous example in literature, we measure the

Copyright is held by the author/owner(s).
CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.
ACM 1-59593-433-2/06/0011.

coverage of the top results from a reference search engine[1] when using only a subset of the available collection: when we measure the *coverage at 5*, we see how many of the first top 5 results are present in the first chosen sub-collection, the first 2 collections and so on.

The query-vector representation is built using, as a training set, the first three weeks of our query log, which comprise about 190,000 unique queries. This means that, at the logical level, documents are represented by vectors in $\mathbb{R}^{190,000}$.

Techniques no. 4 and 5 divided the available documents into 16 clusters. The 17th cluster is the *overflow* cluster, composed of the documents that are never recalled by queries in the training set, represented by empty query-vectors. The overflow cluster is always selected as the last one. In the other cases, the documents are distributed evenly over 17 clusters.

In all cases, we assigned each document cluster to a server of our distributed IR system, and then we used the standard CORI [2] technique to perform collection selection. CORI collects some statistics about the distribution of terms in the collections, and then weights the collections accordingly. With co-clustering, we also used PCAP. With PCAP, the overflow cluster is always queried as the last one. We used the fourth week from the query-log as our test set: the queries from the fourth week were submitted to the distributed search engine, and we measured the coverage we get by returning only documents from 1, 2, 4, 8, 16 and all clusters.

The quality of coverage increases dramatically when we shift from random allocation, k-means on shingles and URL-sorting, to techniques which use the query-vector representation. The simple k-means on the query-vectors is able to reach about 30% coverage (1.47 out of 5) when using only one sub-collection out of 17. Co-clustering improves this figure to 1.57. When we use our collection selection strategy based on PCAP, we are able to cover 34% of the top five documents using only one sub-collection.

These figures are confirmed when we measure the coverage of top-10 and top-20 results, and we shift to later queries (not shown). In general, our proposed approach is able to dramatically reduce the number of collections we need to query to reach a chosen coverage.

Footprint of the Representation. Every collection selection strategy needs a representation of the collections, which is used to perform the selection. Let's call dc the number of document collections, t the number of terms, qc the number of query clusters and t' the number of terms in the queries.

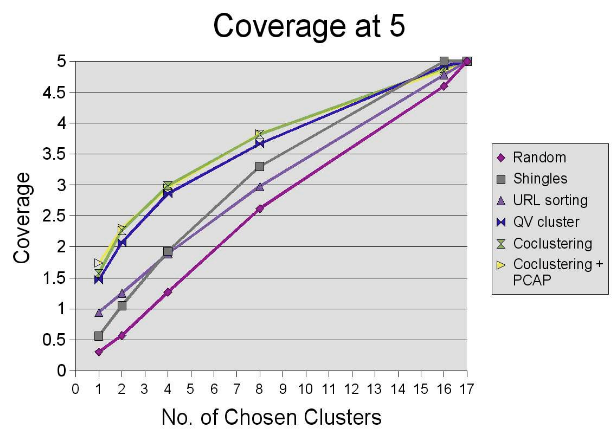
The CORI representation is dominated by a datum called $df_{i,k}$, which is the number of documents in collection i containing term k. Its size it is $O(dc \times t)$. Overall, the CORI representation is composed of about 48.6 million entries before compression.

On the other side, the PCAP representation is composed of the PCAP matrix, with the computed $\widehat{p}$, $O(dc \times qc)$, and the index for the query clusters, which can be seen as $n_{i,k}$, the number of occurences of term k in the query cluster i, for each term occurring in the queries, $O(qc \times t')$. These two terms sum up to about 9.4 million entries, significantly smaller (more than 5x) than the CORI representation.

Empty Query-vectors. At the end of the training period, we observed that a large number of documents, around 52% of them, are not returned among the first 100 top-ranked results of any query. We also verified that they contribute only to 2%-3% coverage of the results in the test query set.

These documents can be pruned and stored in another server, used only when the user browses for low-relevance documents, without a significant loss in coverage.

[1]Zettair, available at http://www.seg.rmit.edu.au/zettair/.



random allocation (CORI)	0.3
clustering with k-means on shingles (CORI)	0.56
URL sorting (CORI)	0.94
clustering with k-means on query-vectors (CORI)	1.47
co-clustering (CORI)	1.57
co-clustering (PCAP)	1.74

Figure 1: Summary of results. Coverage at 5 on all clusters (top) and first cluster only (bottom).

Conclusions. In this contribution, we discussed the query-vector document representation for IR systems. The query-vector model is a compact representation for documents, based on the queries performed by users. It allows for a very effective document clustering, which boost the performance of standard collection selection techniques (CORI).

Moreover, the co-clustering algorithm we use on query-vectors induces a novel selection strategy that outperforms CORI by about 10% and has a smaller footprint.

We could also perform a very effective document pruning, by removing about 52% documents with a limited loss of 3% coverage.

Our results were confirmed at different coverage level (5, 10, 20) and also when we used later queries from the query-log: the system appears to be robust to topic-shift.

References.

[1] Andrei Z. Broder, Steven C. Glassman et al. Syntactic clustering of the web. In *WWW 1997*, pages 1157–1166.

[2] J.P. Callan, Z. Lu, and W.B. Croft. Searching Distributed Collections with Inference Networks. In *SIGIR 1995*, pages 21–28.

[3] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD-2003*, pages 89–98.

[4] D. Puppin, F. Silvestri, and D. Laforenza. Query-driven document partitioning and collection selection. In *Infoscale 2006*.