Abstract

Data fusion or Collection fusion is the final phase of a distributed IR system. A distributed IR system involves several collections. On submitting a query, a ranked list of documents is obtained from some or all of these collections. Data fusion involves selecting the most relevant documents from results returned by various collections. In this paper we define the collection fusion problem and analyze ways of solving and implementing it on the Carrot-II (C2) system.

Introduction

The goal of the C2 project is to build a powerful and high-bandwidth distributed IR system. C2 is a multi-agent system. The C2 consists of several types of agents, which interact and co-ordinate to achieve a common goal. A set of documents is called a collection. Collections in C2 are distributed across several nodes. A client submits a query to any one of the several agents in the system. Agents use metadata obtained from its collection to decide whether its collection will yield relevant documents. Based on this inference the query is either forwarded or handled by the agent. The agents that handle the query forward the query to the IR engine, which returns a set of documents, ranked on their relevance to the query. These results when combined represent the final output. There is no simple procedure to combine the results and yet ensure that the most relevant documents are retrieved. The difficulty arises due to the fact that ranking across the various results are not normalized and therefore not comparable. As a result the similarity value of (say) 0.89 computed for two documents in two sub-collections in no way means that the documents are equally relevant to the query string. This problem does not exist in an IR system that involves a single collection, as the rankings are comparable for all documents. The goal of the collection fusion technique is to combine results from multiple, independent sub-collections into a single result such that the effectiveness of combination approximates effectiveness of searching the entire set of documents as a single collection. This paper proposes a version of the Collection Fusion technique that could be used on the Carrot2 system.

Background

The solution to the collection fusion problem can be partitioned into two phases. The first phase involves finding a cut-off level for each sub-collection. The cut-off level is the number of documents that are considered for the next phase, which involves the interleaving of documents. The proposed solution to this problem derives ideas from a paper "The Collection Fusion Problem" by Ellen, Gupta and Johnson-Laird. The first phase can be tackled using the following two approaches. The first approach is to assume that each collection has an equal number of relevant documents and the relevant documents are identically distributed across each sub-collection. Each sub-collection has a specific orientation with respect to the content of its documents. So relevant documents are not distributed equally among the sub-collections. Tests comparing the effectiveness of this strategy with single collection run indicate a performance degradation of around 40%[1]. A second approach is to assume that the rankings of documents are comparable across the various collections. Fusion can be implemented using this technique by simply listing documents from various sub-collections in descending order of their rankings. The ranking of the documents is dependent on the sub-collection to which they belong. Therefore ranking of documents from different sub-collections are not comparable. Dumais noted this effect in her TREC work [2]. In the relevant document distribution fusion technique, training queries are used to explicitly build a model of the distributions of the relevant documents in the retrieved set of each information server I, and these models are used in a maximization procedure to obtain the number of documents to retrieve from each collection. The relevant document distribution of a query, q is modeled by averaging the relevant document distributions of the k most similar training queries (i.e., the k nearest neighbors of q). The vector space model is used to compute
the similarities among the queries. The vector space is built from the set of training queries, and
the nearest neighbors of \( q \) are those training queries that have the highest cosine similarity
with \( q \). The average relevant document distribution over \( k \) queries is computed by taking
the average of the number of relevant documents retrieved by the set of queries after each
document retrieved. Once the average relevant document distribution is computed for the
current query for each collection, the distributions and the total number of documents
to be retrieved are passed to a maximization procedure. This procedure finds the cut-off level,
\( \lambda_i \), for each collection that maximizes the number of relevant documents retrieved. The
computed cut-off levels are the number of documents selected from each collection. Query
clustering is another approach for solving the fusion problem. This second fusion strategy does
not form an explicit model of a collection's relevant document distribution. Instead, the sys-
tem learns a measure of the quality of a search for a particular topic area on the collection. The
number of documents retrieved from a collection for a new query is proportional to the value of
the quality measure computed for that query. As in the relevant document distribution approach,
the query clustering fusion strategy uses query vectors in the vector space formed from the set
of training queries. Topic areas are represented as centroids of query clusters. For each
collection, the set of training queries is clustered using the number of documents retrieved in
common between two queries as a similarity measure. The assumption is that if two queries
retrieve many documents in common they are about the same topic. Averaging the vectors of
the queries contained within the cluster creates the centroid of a query cluster. This centroid is
the system's representation of the topic covered by that query cluster. The training phase also
assigns to a cluster a weight that reflects how effective queries in the cluster are on that
collection. The weight is computed as the average number of relevant documents retrieved
by queries in the cluster, where a document is “retrieved” if it was among the first \( L \) (a
parameter of the method) documents. After training, queries are processed as follows. The
cluster whose centroid vector is most similar to the query vector is selected for the query and the
associated weight is returned. The set of weights returned by all the collections are used to
apportion the retrieved set such that when \( N \)
documents are to be returned and \( w_i \) is the weight
returned by collection \( i \),
\[
\frac{w_i}{\sum_{i=1}^{C} w_i} \times N
\]
(rounded appropriately) documents are retrieved from collection \( i \). The above techniques work
effectively on static scored corpora. In other
words these techniques assume that the
orientation of the sub-collections remain the
same. This is not true for C2 where documents
can be assigned to different collections. A
modified version of query clustering is proposed
for the C2 system. The second phase involves
interleaving the documents from the various sub-
collections into a single result and thereby obtain
a total ordering. This can be done either by
using a round-robin technique or
probabilistically. Both these approaches do not
violate the partial order of the documents in the
result-set returned by each sub-collection. And
the latter strategy gives preferences to documents
that are believed to have more relevant
documents. An alternative is merging based on
weighted scores [3]. Weights can be based upon
a document's score and/or the collection ranking
information. This approach offers the
computational simplicity of simple interleaving
while overcoming some of its disadvantages.
The weight \( w \), below, is an example of how one
might weight results from different collections.
Here the collection's score is used, instead of its
rank, because it is assumed that similar
collections should have similar weights.

\[
w = 1 + |C| \cdot \frac{s - \bar{s}}{\bar{s}}
\]

where:
\( |C| \) = the number of collections searched,
\( s \) = the collection's score, and
\( \bar{s} \) = the mean of the collection scores.

Each document is ranked based upon the product
of its score and the weight \( w \) for its collection.
This algorithm favors documents from
collections with high scores, but also enables a
good document from a poor collection to be
ranked highly.
Carrot2 System.

Carrot-2 (C2) is an agent-based architecture for distributed information retrieval and document collection management. In C2, agents index and provide search services over local document collections, such as might be hosted on a single web server. A C2 system can consist of an arbitrary number of agents, distributed across a variety of platforms and locations. C2 agents advertise content-derived metadata that describes their local document store; this metadata is sent to other C2 agents which agree to act as brokers for that collection - every agent in the system has the ability to serve as such a broker. The C2 system consists of the Java-based software agents. The Jackal communication infrastructure is used in C2. C2 agents use KQML for communication. The C2 system makes use of IR engines to index documents as well as the metadata. The C2 agent interfaces with the IR engine through a wrapper. There is no restriction on the type of IR engine. The current implementation makes use of MG and Telltale as the IR engines. The wrapper provides functionality to create an inverted index for both the documents and metadata. The metadata object is a vector containing a N-gram and an associated weight. The weight is the number of documents that contain the term. On start-up the collection manager allots each C2 agent a set of documents (sub-collection). Each agent has the metadata object formed from the sub-collection allotted to it.

Fig 1: Query Clustering fusion strategy.

1. Build data structures from training queries.

set of query clusters per collection

2. Form ranked retrieved set for new query

find closest centroid in each collection and return corresponding weight

apportion retrieved set according to weights

Query Routing in Carrot2

The Collection Manager provides a list of agent names that it may query. The client randomly picks an agent and submits it the query. On receiving the query the agent decides whether the query should be answered locally, forwarded to another agent or a combination of the two. This decision is taken after the agent queries its metadata pool. Metadata pool of an agent consists of metadata objects belonging to some or all other agents including its own. The similarity of the query to all the metadata objects is calculated and the top n agents are picked. The query is forwarded to each of these n agents. All the agents who receive the query do the same. The process ends when there are no more agents in the system that have not already received this query or the number of times the query has been forwarded has reached a threshold value.

Implementation

The fusion techniques explained earlier work effectively in the TREC retrieval environment, which is a static scored corpus. The C2 system on the other hand is dynamic. The same document can be assigned to different agents. So the orientation of each sub-collection is not fixed. Hence training queries cannot be used. But some modification to the query clustering technique could lead to a fusion technique that can be implemented on Carrot2. This technique can be implemented on the Carrot2 system without making any major changes to the existing architecture. In-fact a brief examination indicates that no change to the query routing mechanism is required. Access to metadata
objects of all agents is required to determine the cut-off value for each of the sub-collections. If metadata distribution is done through flooding, then information required for determining the cut-off values can be calculated by accessing the metadata pool of any agent. Under the flooding scheme, any agent receiving the query would have complete knowledge of the system. In other words it has metadata objects belonging to all agents in its metadata pool. Once the system reaches a certain size, this scheme ceases to be practical. The primary reason being that the size of the metadata pool will become unmanageable once the number of agents becomes very large. An alternative would be to use a new agent whose sole purpose is to access the metadata object of every agent to calculate the cosine similarity of the submitted query to the (metadata) sub-collection. This agent can then compile a list of agents and the associated similarity of the query to the metadata. A simpler approach would be to append the similarity values for the query to the result being returned. As the agents start returning their results the similarity values can be simultaneously gathered. These meta-data and the cosine similarity are analogous to query clusters and their weights. Now the similarity is used to apportion the retrieved set such that if N documents are to be retrieved then the cut-off value is determined by the formula.

\[
\frac{w_i}{C} \times N
\]

\[
\sum_{i=1}^{C} w_i
\]

\( w_i \) is the cosine similarity of the query to the metadata
\( C \) is the total number of sub-collections.
\( N \) is the number of documents that need to be retrieved.

Once the cut-off level is determined, the interleaving of documents can be done. A probabilistic approach is preferred to a round-robin scheme. The probabilistic approach prefers sub-collections, which are believed to be more relevant to the query.

\[
\text{Interleaving leads to the ordering shown below}
\]

1: //agent1
   A1
   D3
   R2
   R23
   R1
   D2

2: //agent2
   Q1
   Q2

3: //agent3
   A2
   R1

This structure consists of a agent name (1,2 and 3) and document-ids (A1, D3…..).
Here \( \lambda_{\text{agent1}} = 5 \), \( \lambda_{\text{agent2}} = 2 \) and \( \lambda_{\text{agent3}} = 2 \)

Note that this algorithm maintains the partial order of each of the sub-collections.
Future

The proposed technique at this stage is purely theoretical. Its effectiveness needs to be tested on the C2 system. This technique makes use of existing entities of the C2 system and therefore could be implemented on C2 without making drastic changes to the existing architecture.

Conclusion

Analysis of the various fusion techniques reveals that the combination of a query clustering like approach coupled with interleaving using probability is a suitable option to implement on the C2 system. The technique’s use of existing information makes this option all the more attractive. And finally the decentralized nature of this technique makes it less prone to scalability issues.

References

[3] James P Callan, Zhihong Lu and Bruce Croft Searching Distributed Collections With Inference Networks.

Acknowledgements

Prof. Scott Cost
Srikanth Kallurkar
Hemali Majithia