4 Fusion of Results

Considerable research has been devoted in IF to the problem of “fusion” of results. The term “fusion” has been applied in IF to two different problems: the fusion of results retrieved from multiple collections, and the fusion of results retrieved from the same collection by multiple methods.

4.1 Fusion of Results from Multiple Collections

“The need to search multiple collections in distributed environments is becoming important as the sizes of individual collections grow and network information services proliferate.” [Callan, SIGIR ‘95] A given query may be submitted to multiple collections. A list of documents, typically ranked, will be retrieved from each collection. The fusion problem is then how to merge these ranked collections for presentation to the user. “[T]he goal … is to combine the filtering results from multiple, independent collections into a single result such that the effectiveness of the combination approximates the effectiveness of searching the entire set of documents as a single collection.” [Voorhees, SIGIR ‘95] Given that the user wants to see a total of $N$ documents, the problem is to determine how many of those documents to obtain from each of $C$ collections, $C_1, C_2, \ldots, C_C$.

Note that collections may have different specialties. Hence, the number of documents relevant to a given query $Q_1$ may vary widely from one collection to another. For this reason, it is generally unsatisfactory to merge documents by taking equal numbers $(N/C)$ from each collection. Also note that, in general, each collection may be indexed differently, and may be served by a different information server, using a different IF algorithm or combination of algorithms. Hence, even the same set of documents may be relevance-ranked differently with respect to the same query $Q_1$ by two different information servers. We cannot, in general, assume that the similarities computed by the various information servers are comparable. Hence, in general, we cannot simply take the $N$ documents having the highest similarity scores relative to $Q_1$.

Voorhees et al. propose two schemes for determining how many documents $DR_i$ to retrieve from the i-th collection such that $DR_1 + DR_2 + \ldots + DR_C = N$. The schemes differ in how the $DR_i$ cut offs are to be computed. However, the merging of documents
given that the cut offs have been computed is the same in both cases. The merging scheme is to keep track of how many documents remain to be merged from each collection $C_i$ to satisfy its cutoff requirement $DR_i$. Let $DN_i$ be the number of documents not yet selected from collection $C_i$ to reach its cutoff. (Initially, $DN_i = DR_i$.) At each step, a $C$-faced die ($C =$ the number of collections) is (conceptually) thrown. The die is biased in proportion to the $DN_i$ so that the probability of selecting the next document from $C_i$ is $DN_i/N$. Whichever face of the die “comes up”, the next document in rank order from the corresponding collection is selected and added to the growing merge stream. The biases of the faces of the die are recomputed, the die is thrown again, and another selection made.

For example, suppose that $N = 10$, $C = 3$, and the cutoffs are $DR_1 = 5$, $DR_2 = 3$, $DR_3 = 2$. Then, the merging would proceed as follows (where $D_{ij}$ is the document of rank $j$ retrieved from collection $i$): The first document selected would be either $D_{11}$ (with a probability of 1/2), or $D_{21}$ (with a probability of 3/10), or $D_{31}$ (with a probability of 1/5). Assuming for concreteness that $D_{11}$ was selected, the biases would be recomputed so that the next document selected would be either $D_{12}$ (with a probability of 4/9), $D_{21}$ (with a probability of 1/3), or $D_{31}$ (with a probability of 2/9). This process would continue until ten documents had been selected. Note that the order of documents from a given collection $C_i$ in the merged filtering set reflects the document rankings returned from $C_i$, and the number of documents from each collection in the merged filtering set equals its cutoff value. Since $C_2$ has a cutoff of 3, the merged filtering set will contain the three top ranking documents from $C_2$, i.e., $D_{21}$, $D_{22}$, and $D_{23}$ in that order, but their actual ranks in the merged list will be determined probabilistically by the throws of the die. It should also be noted that this merge method is completely independent of how the cutoff values are computed or estimated; it merely assumes that the user has obtained cutoff values for each collection in the set whose filtering outputs are to be fused. The cutoff values might even be obtained after filtering from each of the individual collections has occurred. That might be necessary, for example, in a dynamic, distributed environment where the collections to be accessed are not known in advance.

Now, how can the cutoff values be computed? Both methods proposed by Voorhees et al. are based on the existence of a static set of collections known in advance, a training set
of queries, and the corresponding training set of retrieved documents from each
collection for each query in the training set. Each document in the training filtering set
for a given training query is assumed to have been judged relevant or non-relevant to the
given query. (That, of course, is what makes it a training set!) As with all methods
dependent on a training set, the assumption is that the results obtained with the training
queries are predictive of the results that will be obtained with new queries, i.e., that there
will be training queries “similar enough” to the new queries with respect to the given
collections. The first method is called the “relevant document distribution” (RDD)
method. Given a proposed query $Q_1$, the $k$ most similar training queries (the “nearest
neighbors” to $Q_1$ in query space) are computed using some similarity measure. (Voorhees
uses cosine similarity.) The value of $k$ is a parameter of the method. The filtering sets for
these $k$ most similar queries are then used to compute a “relevant document distribution”
as follows: For each collection $C_i$, determine the average number of relevant documents
at rank one for the $k$ nearest neighbor training queries. Then, determine the average
number of relevant documents at ranks one plus two for the same $k$ queries, etc. For
example, suppose that there are three collections, $C_1$, $C_2$, and $C_3$. Further assume that $k = 3$, i.e., there are three nearest neighbor training queries to $Q_1$: $QT_1$, $QT_2$, and $QT_3$. Then
our training data includes nine filtering sets, e.g., the filtering set for $QT_1$ applied to
collection $C_2$ is $R_{12}$. The filtering sets $R_{12}$, $R_{22}$, and $R_{32}$ tell us how documents in $C_2$ were
ranked by $QT_1$, $QT_2$, and $QT_3$, respectively. If the rank one document in $R_{12}$ is relevant,
but the rank one documents in $R_{22}$ and $R_{32}$ respectively are non-relevant, then an
“average” of 0.333 relevant documents (one relevant document divided by three training
queries) is retrieved from $C_2$ at rank one by the set of nearest neighbor training queries.
Similarly, if the rank two documents in $R_{12}$ and $R_{32}$ are relevant but the rank two
document in $R_{22}$ is non-relevant, then after two documents have been retrieved from $C_2$
by each of the three training queries, an average of one relevant document has been
retrieved (three relevant documents divided by three training queries). We repeat this
process for ranks three to $N$, generating a cumulative average distribution of relevant
documents by rank (the RDD) for $C_2$. Similar distributions are calculated for $C_1$ and $C_3$. 
By examining these distributions, one can determine the optimum cutoff value for each of
the collections $C_1$, $C_2$, and $C_3$. The essential point is that the optimum set of cutoffs for a
set of collections depends not only on how many relevant documents are retrieved in the top \( N \) documents for each collection. It also depends on how these relevant documents are ranked in each collection. For example, suppose that on the average (across a set of \( k \) training queries for the given query \( Q_1 \)), three relevant documents are retrieved in the top ten for each of two collections \( C_1 \) and \( C_2 \). But suppose that the relevant documents have higher rankings on average in \( C_1 \) than in \( C_2 \), e.g., ranks 1, 2, and 4 in \( C_1 \) vs. ranks 3, 7 and 8 in \( C_2 \). The RDD for \( C_1 \) is \{1, 2, 2, 3, 4, 4, 4, 4, 4, 4\}; the RDD for \( C_2 \) is \{0, 0, 1, 1, 1, 2, 3, 3, 3\}. Then for \( N = 10 \) (the merged stream returned to the user will contain ten documents), the optimum cutoffs are plainly \( DR_1 = 2, DR_2 = 8 \) for which the RDD’s predict that the user will retrieve five \((2 + 3)\) relevant documents.

The 2nd proposed method of computing cutoffs from a set of training queries is computationally cheaper but does not take rankings into account.

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 … The centroid of a query cluster is created by averaging the vectors of the queries contained within the cluster. This centroid is the system’s representation of the topic covered by that query cluster.

For each query cluster in the training set and each collection, a weight is assigned equal to the average number of relevant documents retrieved by queries in the cluster from the given collection among the first \( L \) documents (where \( L \) is a method parameter). Hence given a query cluster, we have a set of weights, one for each collection.

Given a query \( Q_1 \), the query cluster method associates \( Q_1 \) with the cluster whose centroid vector is most similar to \( Q_1 \) (presumably using cosine similarity or the like again). A cutoff is assigned to each collection as in the RDD method. As in RDD, the cutoffs must sum to \( N \), but this time they are proportional to the collection weights for the given cluster. Hence, the number of documents \( DR_i \) taken from a given collection \( C_i \) in response to \( Q_1 \) is proportional to the average number of relevant documents retrieved from \( C_i \) by training queries in the cluster that is most similar to \( Q_1 \). It is not at all dependent on how the documents retrieved from \( C_i \) by the training queries were ranked (except that they must have been ranked \( L \) or better, of course).
Initial experiments using a subset of the TREC topics showed that the relevant document distribution method performed better than the query clustering method, but at a cost in greater processing time and larger data structures. Both methods performed less well than the ideal, i.e., the results that would have been obtained if all the documents were in a single collection. But the decrease in quality was small enough that it appears that these fusion methods are feasible given the assumptions on which they are based: static collections, training queries, etc.

Note that regardless of the method used to compute the cutoffs, $DR_i$, it is quite possible that some of the $DR_i$ will be equal to zero, reflecting the fact that some collections may have few if any documents relevant to the given query. This is especially likely if some of the collections are specialized to particular topics.

Also note, as pointed out by Voorhees et al., that these fusion methods make no allowance for the possibility that the same relevant document may be retrieved from two or more collections, a possibility that is particularly likely if two or more collections deal with the same specialized topic.

What if the collections are widely distributed and dynamic? Callan et al. [SIGIR ‘95] have proposed an alternative fusion approach, an extension to the inference network method, a probabilistic approach to IF (see preceding section) that has been applied in the INQUERY system. The inference network approach has previously been applied to filtering of documents in a collection. In the paper discussed here, it has been extended to apply to fusion of outputs from collections by re-interpreting some of the basic statistical measures at the collection level. Hence, two levels of index are created: First, there is the traditional document level where documents are indexed by the terms they contain, their frequencies within each document, and the distributions of these terms over the entire collection. Then, there is a new collection level of index where collections are in effect treated as very large “virtual documents” indexed by terms in ways analogous to the document level indexes. It would appear that the same kind of extension could be applied to any method based on term indexing of documents. The inference network defined by this collection level index is called a Collection Filtering Inference Network, or CORI net for short.
For example, term frequency \( (tf) \) meaning “number of occurrences of a given term within a given document” is replaced by document frequency \( (df) \) meaning “the number of documents within a given collection containing occurrences of a given term.” Similarly, the number of documents in a collection, \( D \), is replaced by the number of collections to be accessed, \( C \). The document frequency, in turn, is replaced by collection frequency \( (cf) \) meaning “the number of collections (out of the total \( C \) to be accessed) containing a given term.” And, inverse document frequency \( (idf) \) defined for a given term in a given collection as \( \log(D/df) \), is replaced by inverse collection frequency \( (icf) \), defined for a given term in a given set of collections as \( \log(C/cf) \). The \( idf \) is a measure of how few documents in a given collection use a given term, i.e., it is zero if the term is used by every document in the given collection and is maximized if only one document in the given collection uses the term (which may indicate that it is a good descriptor of the topic with which the given document deals). Similarly, the \( icf \) is a measure of how few collections (out of the \( C \) to be accessed) use a given term. Note: Actually, Callan et al. use a slightly more complex formula for \( icf \):

\[
icf = \frac{\log \left( \frac{C + 0.5}{cf} \right)}{\log(C + 1.0)}
\]

In the usual application of inference networks to document filtering from a single collection, the presence of a given term in a given document may provide evidence to increase the probability that the given document satisfies the user’s information need. In the extension to multiple collections, a given term may also provide evidence about the probability that a given collection contains documents satisfying the user’s information need.

Hence, filtering of documents for a given query \( Q \) from a distributed collection of documents becomes a two-stage process: First the collection level index is used to rank the collections relative to \( Q \). Then, if the number of collections is large, document filtering is applied only to the higher ranking collections, the ones most likely to contain documents relevant to \( Q \). If the number of collections is small, all the collections (or all with score above some threshold) may be accessed, but greater weight may be given to documents retrieved from higher ranking collections.
One very important limitation of a collection level index should be noted here. In an ordinary document level index, each term (and associated term frequency) is associated with the documents in which it occurs. Hence, if two terms $t_1$ and $t_2$ appear in the index, one can determine whether they co-occur in the same document(s). (However, if they co-occur in document $D_1$, one can’t tell whether they occur in close proximity, unless term position within the document is retained in the index, which would greatly increase the index size, or the actual document itself is retrieved and examined.) If two terms $t_1$ and $t_2$ appear in a collection level index, one can tell whether they co-occur in the same collection, i.e., the same “virtual document,” but we can’t tell whether they co-occur in the same document within that collection, let alone whether they occur in close proximity within a given document. This means that the problem of determining whether two co-occurring terms are semantically related, difficult enough when the terms are known to co-occur in a given document, becomes much more severe when the terms are only known to co-occur in a given collection, perhaps in entirely different documents. Methods of compensating for these limitations are discussed later.

The extension of the method from the document level to the collection level introduces a problem relating to large collections analogous to the problem identified by Lee (discussed earlier) for large documents. Lee pointed out that a large document is apt to deal with multiple topics. Obviously, this is even more likely with respect to a large collection! In the case of a large document, Lee points out that “maximum normalization,” i.e., normalization of term weights for a given document by maximum term frequency in the given document, is liable to drag down the weights of terms relating to a relevant topic; specifically, if the highest frequency term in large document $D_i$ deals with sub-topic $A$, it will drag down the weights of terms in $D_i$ dealing with relevant sub-topic $B$. Callan et al. point out the same difficulty for large collections. At the collection level, maximum term frequency ($t_{f_{max}}$) is replaced by maximum document frequency ($d_{f_{max}}$), the number of documents containing the most frequent term in the given collection. (Here, “most frequent term” means the term that occurs in the most documents of any term in the collection.) Normalizing document frequency ($df$) by $d_{f_{max}}$ for a large collection can produce a similar effect to that noted by Lee for large documents. If the large collection contains a significant subset of documents relevant to an information need $Q_1$, maximum normalization may obscure the presence of this.
subset if other larger subsets deal with other topics, and particularly, if the most frequently occurring term is associated with one of these other topics.

Callan et al. propose to deal with this maximum normalization problem for collections by using a different kind of normalization for \( df \), scaling by \( df + K \) where \( K \) is a large constant, rather than scaling by \( df_{\text{max}} \). They suggest \( K \) should be a function of the number of documents in the collection. (They have tried a similar approach at the document level; in that case, the scaling factor is \( tf + K \) and \( K \) is a smaller constant, a function of document length.)

The CORI net approach allows the system to rank collections by the probability that they will satisfy the user’s information need, just as the document level inference network allows the system to rank documents within a given collection relative to the user’s information need. One virtue of this approach is that the same system can perform the ranking at both levels; indeed, the same algorithm can be applied to both the document level inference network and the collection level CORI net, since the indexes have the same structure and analogous semantics. Another virtue is that the collections, like the documents within a collection, receive scores, not just rankings; the score for a given collection reflects the probability that it will contain documents that satisfy the user’s information need.

Once each collection has been searched, we must address the same problem as with the “relevant document distribution” and “query clustering” methods discussed above: how to merge the ranked (and scored) outputs from each of the searched collections into a single merged, ranked output to present to the user. Because the CORI net approach generates a score for each collection, it is possible to compute a weight for each collection without using or requiring a query training set as in the query clustering method. An example of a formula for calculating collection weights from their CORI net is:

\[
    w_i = 1 + C \cdot \frac{s_i - \bar{s}}{\bar{s}}
\]

where \( w_i \) is the weight assigned to collection \( C_i \), \( C \) is the number of collections searched, \( s_i \) the ORI net score of collection \( C_i \), and \( \bar{s} \) is the mean of all the collection scores. Bear in mind that these collection weights (and the scores from which they are derived) are relative to a given user query or information need. Given a different query, different weights would be computed for the collections searched.
Once a weight has been computed for each collection, how should one use them to merge the retrieved (and ranked and scored) document outputs of these collections into a single stream? One could clearly use the same approach as in the query clustering method: convert the weights into proportional cutoffs summing to some desired total $N$, and then merge the filtering sets using the merge algorithm described above. However, both the query clustering and relevant document distribution methods assume that the filtering sets are ranked, but make no assumption about the documents in each set being scored. The inference net approach produces a score for each document as well as for each collection. If one can assume that the scores assigned to retrieved documents from one collection are strictly comparable to the scores of documents retrieved from another collection, then one can of course merge the documents into a linear order by score alone. However, experience shows that even when the same method, e.g., the inference net method, and the same indexing, is applied to different collections, the scores for a given query may not be comparable due to large variations in the statistical properties of the collections. Hence, the CORI net approach to merging is to compute a global score for each document as the product of the weight of the collection from which it was retrieved, and its “local” score within that collection. Documents are then merged by ranking them according to these global scores. The effect is to favor documents from highly weighted collections (equivalent to favoring documents from collections with high cutoffs in the cutoff-merge method); however, documents with very high scores from lower weighted collections are also favored (equivalent to choosing a very highly ranked document from a collection with a low cutoff in the cutoff-merge method).

Why should it be the case that scores from documents retrieved from two different collections are not necessarily comparable even when the documents are represented and described in the same way in both collections, and the same filtering algorithm is used to compute document scores (relative to a given query) for both collections? Consider two collections, one containing legal opinions, the other containing papers relating to computers and computer science. Now consider a query $Q_I$ containing the word “tort.” Many documents in the legal collection will probably contain the word “tort.” The computer science collection may contain a few documents relating to liability of software engineers in which the word “tort” appears. If $idf$ is used in the term weighting scheme,
a document in the computer science collection dealing with liability will receive a much higher score relative to $Q_I$ than a comparable document (even the same document!) in the law collection. Viewing each collection separately, this is quite appropriate; “tort” is a much more significant descriptor in the computer science collection because of its rarity, than it is in the law collection. But the scores cannot be compared directly when $Q_I$ is used to search the two collections, and the resulting filtering sets are to be merged. This problem can be remedied by computing normalized global statistics, e.g., a normalized idf that reflects the number of documents containing the term “tort” in the law and computer collections combined; this normalized idf has to be computed from the statistics of all the collections to be searched, it has to be computed before any of the searches are executed, and it has to be used by all the information servers in place of the local idf’s available to those servers. The effect in the computer-law example is that the scores of documents in the law collection for $Q_I$ that contain “tort” become higher (because of all the documents in the computer collection that don’t contain “tort”) and the scores of the few documents in the computer collection that contain “tort” become lower (because of all the documents in the law collection that do contain “tort”). Now the scores are comparable; in effect, the two collections are being treated as if they were one. However, this is an expensive procedure, in computation and communication, especially if the collections are widely distributed. Experiments indicate that ranking based on the product of collection weight and document score is about as effective, and considerably cheaper.

The CORI net method of fusion (or any similar approach using term-based indexing on the collection level) is obviously better suited than the relevant document distribution or query clustering methods to a dynamic environment where the collections are widely distributed, the number and identity of the collections to be searched changes rapidly from one query to another, and the contents of the individual collections themselves change rapidly as new documents are added. This is because it does not require the existence of a query training set of documents which must necessarily have been prepared by applying some fixed set of training queries to a fixed set of collections. Adding additional collections to be searched does not require the major effort of updating a training set to include these new collections. It only requires that each collection be comparably indexed and that global statistics for the set of collections to be searched, e.g.,
icf, be updated. The latter, of course may involve some significant run-time expense, but a lot less expense than updating a training set. Rapid updating of individual collections requires (at least) periodic updating of their indexes, and correspondingly, updating of global statistics like icf when a set of collections is searched. But again, this is less expensive than regular updating of a training set.

Xu and Callan [SIGIR ‘98] carry the CORInet research further, increasing the number of collections substantially (from 17 to 107), and improving the filtering process. (They also, it appears, abandon the acronym CORInet, which is a good idea since the essence of the reported research does not depend on the use of an INQUERY inference network, and could be carried out with some other effective IF engine.)

The first important (but not surprising) result they report is that the effectiveness of straightforward word-based filtering is considerably poorer for 107 collections than for 17 collections. In particular, average precision for a distributed set of collections becomes considerably worse as the number of collections goes up, relative to the precision achieved when the same data is effectively treated as a single integrated, centralized database, e.g., with population wide statistics such as idf computed and maintained for the entire set of collections rather than for each collection separately. Note that going from 17 to 107 collections did not involve increasing the amount of data. The same TREC data was used in both cases. The number of collections was increased in the latter case by subdividing the same total set of documents more finely, into a much larger number of subsets. Rather, the important difference was that statistics were computed and maintained in a separate index for each collection, as would be the case in a realistic situation, e.g., a large set of collections on the Internet. Yet with 17 collections (17 indexes) and the CORInet approach described above, average precision was almost as good as for a single, centralized collection with a single centralized index. With 107 collections (107 indexes), the average precision declines by 23% to 32.7%, depending on the cutoff (number of documents retrieved).

To compensate for the loss of precision resulting from distributing the index information over 107 separate indexes, they adopt two strategies: use of syntactically determined phrases as well as individual words as index terms, and query expansion using the Local Context Analysis (LCA) method described above in the section on query expansion. The value of the former is obvious: If the common words “high,” “blood,”
and “pressure” co-occur in a collection, they may occur in entirely different documents. However, if the phrase “blood pressure” occurs as an index term to a given collection, the likelihood that one or more documents in the collection actually deal with the subject of “high blood pressure” is obviously much greater.

The value of query expansion by LCA arises from the fact that terms are added to the original query that co-occur in actual documents. Such co-occurrence may be a significant indicator that the collection contains documents relevant to the topic of the original query. Especially important is the addition of “topic” words, i.e., words that by themselves are strong indicators of the topic under discussion in any document in which they appear. For example, given the topic “high blood pressure,” the expansion may generate words like “hypertension” and “cholesterol” whose presence greatly increases the likelihood that a given collection contains documents about the desired topic, and greatly increases the likelihood that documents retrieved from the given collection will actually be relevant to the given topic.

Note that, as explained earlier, LCA works by retrieving the best passages relative to the given query using a conventional IF engine, and then ranking candidate concepts for expansion on the basis of co-occurrence in the retrieved passages with all the query terms. Hence, LCA requires document indexes at the passage level, i.e., the “documents” are passages (fixed length text windows) within the documents. However, these passage-level indexes are maintained separately for each collection. The only global index is the collection-level index, which only contains term statistics by “virtual document,” i.e., by collection. Hence, the global index can be quite small relative to the large set of collections being indexed. Xu and Callan point out that if document boundary information, i.e., which documents a given term is in, were maintained in the collection level index, the index would be about as large as the set of collections being indexed!

In the reported experiments using TREC3 and TREC4 data, LCA proved quite effective. The average precision using the expanded queries was only slightly less (an average drop of only 2.6% for TREC4) than querying the same data as a single centralized database. Use of phrase descriptors also improved performance substantially for distributed collections, but not as much as LCA. The combination of LCA and phrase descriptors was best of all, but LCA alone accounted for most of the improvement.
Viles and French [SIGIR ‘95] deal with a closely related problem: How often must
global statistics be updated as individual collections receive new documents? The
situation they consider is not quite the one we have been considering above: a set of
independent collections at (perhaps) widely distributed sites. Instead, they consider a
single collection (they call it an “archive”) distributed over multiple sites. Each site has a
different subset of the total set of documents in the collection, and its own server which
maintains indexes for its site and cooperates with servers at other sites. How does this
differ from the case of multiple “independent” collections which we have been
considering up to now? The primary difference is that each server maintains copies of
global statistics such as \( \text{idf} \). In other words, all sites possess the same global value of
\( \text{idf} \) for a given term, a value that applies to the total collection, i.e., the total of all the
subsets from all the sites. This global \( \text{idf} \) is essentially the normalized idf discussed above
for the computer collection/ law collection example. A given site must maintain such a
global \( \text{idf} \) for every term used as an index at the given site. Given that a new document is
added to the subset at one of the sites, ideally the update should be disseminated
immediately to all the other sites so that they can update all their \( \text{idf} \)'s. This is quite
expensive if there are many sites, they are widely distributed, and updates are frequent.
Moreover, the addition of a single document at one site is not likely to have a large
effect on all the global \( \text{idf} \)'s (or on any of them). Nor is it essential that sites receive
update information in exactly the same order that the updates occurred. As Viles and
French note, “The goals of an IF system generally do not include serializability of
updates on the \( \text{idf} \).” The question Viles and French pose and attempt to answer is this:
How often must each site notify other sites about updates it has received so that filtering
performance is not significantly degraded? (They call less- than-immediate dissemination
of updates “lazy dissemination.”)

To discuss this problem, Viles and French define a dissemination parameter, \( d \), and an
affinity probability, \( a \). The \( i \)-th site, \( s_i \), knows about all its own documents, i.e., the
documents physically stored at \( s_i \). The site also “knows about” the first \( d \)-th fraction of the
documents stored at any other site, \( s_j \); here, “knows about” means that global statistics
such as \( \text{idf} \) have been updated at \( s_j \) to reflect that fraction of the documents at any other
site, \( s_j \). Hence, \( d \) varies continuously between 0 and 1. When \( d = 0 \), no dissemination
occurs. When \( d = 1 \), each site has “complete” (statistical) knowledge about the
documents at every other site. When \( 0 > d > 1 \), \( s_i \)'s global knowledge is derived partly from its own physical holdings, and partly from disseminated knowledge of documents held elsewhere. Hence, \( d \) is a parameter for experimenting with the percentage of a site’s holdings for which knowledge is disseminated to other sites. The affinity probability, \( a \), is a tool for experimenting with different strategies for allocating new documents among the sites. When \( a = 0 \), documents are randomly allocated among the sites. “When \( a = 1 \), documents relevant to the same query are co-located, mapping to the case where content has a large influence on document location.” (The assumption that documents relevant to the same query are relevant to each other is a simple scheme employed by Viles and French to experiment with content-based allocation of documents; it is not intended “as a recommendation for document clustering.”)

Viles and French experimented using SMART v11.0 software and four well-known document collections. Perhaps the most important result was that for all values of \( a \) the greatest increase in IF effectiveness occurred as \( d \) was increased from 0.0 (no dissemination) to 0.2. For \( a = 0.0 \), (random rather than content-based allocation of documents to sites), boosting \( d \) from 0.0 to 0.2 was sufficient to achieve precision levels for all levels of recall that were “essentially indistinguishable from [a] central archive”. At high levels of affinity (\( a = 1.0 \)) corresponding to content-based allocation of documents, higher levels of dissemination were required to achieve precision equivalent to that of a centralized archive. The required dissemination varied from 0.4 to 0.8 depending on the document collection used for the experiment. However, even in these cases, the largest part of the improvement occurred in going from \( d = 0.0 \) to \( d = 0.2 \). “Successive jumps in dissemination past the \( d = 0.2 \) mark yield relatively lower effectiveness gains.” Hence, it appears that effective IF can occur in a distributed archive, even when each site has considerably less than complete knowledge of the other sites. However, “[t]here appears to be some minimal sample of documents that a site needs to know about to achieve search effectiveness comparable to a central archive. It remains to be seen whether this sample is a fraction of the whole, or if some minimal number of documents is needed.”

4.2 Fusion of Results Obtained by Multiple Methods

A number of researchers have observed that different filtering methods applied to the same collection to satisfy the same information need can result in retrieving quite
different document sets, i.e., there was surprisingly little document overlap across
sets, either in relevant documents retrieved, or in non-relevant documents retrieved.
[Belkin et al, SIGIR ‘93] [McGill et al., Syr U, 1979] Moreover, the performance of
these different methods tended to be comparable, i.e., the proportion of relevant
documents retrieved did not vary as much as one would have expected from one method
to another given the small amount of overlap in their respective filtering sets. [Katzer
et al., 1982] These plausible findings would lead one to expect that combining results of
multiple methods would lead to improved filtering, because more relevant documents
would be retrieved (or would receive high rankings) from the combination of methods
than from any one method alone. Such a result would be plausible, because one would
expect that different methods would have different strong points and weak points. This
has been called the “skimming effect” [Vogt et al., SIGIR ‘98] because the user is
“skimming” the best documents retrieved by each method.

In contrast, Lee [SIGIR ‘97] and Vogt et al. [SIGIR ‘98] find (in more recent research)
that different filtering methods tend to retrieve the same relevant documents, but different
non-relevant documents. This has been called the “chorus effect,” [Vogt et al., SIGIR ‘98]
because it means that the more methods retrieve a document (in other words, the louder
the “chorus” acclaiming the document), the more likely it is to be relevant.

The phrase “different filtering methods” can mean quite different things:

1. Different users using the same query formalism, e.g., all the users formulate boolean
   queries in response to the same information requirement, but each user formulates
   her query independently of the others. [Belkin et al, SIGIR ‘93] [Saracevic et al.,
   JASIS, 1988] [McGill et al., Syr U, 1979]

2. The same or different users employing different query formalisms, e.g., natural
   language vs. Boolean vs. Probabilistic, to satisfy the same information need.
   [Turtle et al., ACM Trans IS, 1991] [Belkin et al, SIGIR ‘93]

3. The same or different users employing different vocabularies, e.g., controlled vs.
   free-text, to satisfy the same information need.[McGill et al., Syr U, 1979]

4. Different document representations, e.g., title vs. abstract, or automatically
   generated index terms vs. manually assigned terms, or LSI vs. keywords. [Katzer
   et al., IT, R&D, 1982] [Turtle et al., ACM Trans IS, 1991] [Foltz et al., CACM, 1992]
5. Different weights applied to the query terms and document terms within a single query representation and a single document representation. [Lee, SIGIR ‘95]

6. Different filtering (document classification, filtering, query-document similarity) strategies, e.g., vector space cosine similarity with query expansion vs. logistic regression, vs. neural networks vs. linear discriminant analysis (LDA), or linear and logistic regression vs. neural networks vs. pattern recognition techniques. [Schutze et al., SIGIR ‘95] [Chen, CIKM ‘98]

As noted above, some of this research has also indicated that combining filtering or classification results from two or more methods (fusion of results) can produce better filtering performance than any one of the methods by itself. Typically, results are combined by applying each method separately to a given document, and taking the sum (or equivalently, the mean) of the scores. [Fox, et al., TREC-2] [Lee, SIGIR ‘95] [Hull et al., SIGIR ‘96] Note that there are two cases here: In the ad hoc query case, a query is executed against a given collection by each of several methods, e.g., several IF engines each employing a different method of computing document-query similarity. In that case, each individual run returns a ranked list of documents, with each retrieved document in the ranked list assigned a similarity or probability score. Then, for each document retrieved by at least one run, the scores assigned to the given document by each run are summed (or averaged); of course, if a given document is not retrieved at all by a given run, its score for that run is usually considered zero. On the other hand, in the routing (or filtering) case, a single stream of documents is classified. Each document is classified by each of the classification methods. Each classifier assigns a score to the given document with respect to each query. The scores assigned to the given document for a given query are then summed (or averaged) as in the ad hoc case.

Variations on this simple combination scheme are possible. A weighted sum may be employed if there is reason to believe that one method is more reliable than another. If the scores are probabilities, it “may make more sense” to average logodds ratios. [Hull et al., SIGIR ‘96]. If it is desired to give particular favor to documents retrieved by multiple methods, even more favor than does a simple sum of the scores, then one can employ Fox and Shaw’s [TREC-2] CombMNZ function, defined as the sum of the similarities times the number of non-zero similarities. Lee’s [SIGIR ‘95] findings with this function are discussed below.
In some of the cases listed above, the “multiple methods” are multiple query formulations, rather than distinct IF engines or classifiers. In case one, multiple users employ the same formalism. Hence, multiple individual runs correspond to multiple users, each formulating a query using the same formalism, in response to the same information need, and executing their queries against the same collection via the same IF engine. Hence, the individual runs can be combined exactly as above, but instead of trying to balance the strengths and weaknesses of different filtering or classification methods, we are trying to balance the strengths and weaknesses and backgrounds of different users. The second “multiple method” case also involves multiple queries applied to the same information need, but in this case each query (whether generated by the same or a different user) employs a different formalism. So, once again, individual runs can be combined using the above methods, but now it is the strengths and weaknesses of different formalisms (assuming the users have comparable experience and ability, of course) that are being balanced. It should also be noted that, using a system like INQUERY, it is possible to combine separately formulated queries into one “super” query which is then executed normally by the INQUERY filtering engine; in that situation (not discussed here), it is the queries (not the results) that are fused together, and only one output filtering set is produced. [Belkin et al, SIGIR ’93]

To complicate matters further, two distinct query formalisms may indicate two distinct filtering methods, e.g., a term query may be treated as a term vector and evaluated by cosine similarity, while a boolean query is presumably being executed by boolean logic evaluation. However, a term query and a natural language query may both be evaluated as term vectors, i.e., in some cases, key terms are extracted from a natural language query exactly as from a document to form a term vector.

Why should fusion of results produce better performance, e.g., better precision for a given level of recall? Belkin et al. [SIGIR ’93] suggest two general reasons: First (and most obvious), if there is relatively little overlap between the document sets retrieved by two methods, and the methods (taken separately) exhibit comparable performance, the implication is that each method only retrieves a different fraction of the relevant documents. Hence, merging the best documents retrieved by each method should result in a set containing a higher percentage of relevant documents than any single method alone. Presumably, each query formulation or term weighting method or filtering strategy has its own strengths and weaknesses.
Second, if there is some non-zero probability $p_i$ that a method $m_i$ will retrieve a relevant document $D_{rel}$, then the probability that at least one of several such distinct methods, $m_j$, $m_k$, and $m_l$, will retrieve $D_{rel}$ is surely greater. Hence, a document retrieved by several different methods is more likely to be relevant than a document retrieved by one method alone. If a document is retrieved (or classified) as relevant to a given query by more than one method, i.e., more than one method assigns it a score above a specified threshold, the probability that it will “make the cut” when the outputs are combined is greater than if only one method retrieves the document. This probability is greater still if the rank of a document is raised in proportion to the number of methods that retrieve it. Saracevic and Kantor [JASIS, 1988] didn’t actually return the intersection, but they found that “the odds of a document being judged relevant increased monotonically according to the number of retrieved sets that it appeared in.” [Belkin et al., SIGIR ‘93]

It should be noted that all of the fusion results discussed in this section represent averages over a set of queries. If one looks closely at the results for individual queries, one finds that some methods work very well on some queries, other methods work very well on other queries. Hence, if one knew in advance which method was best for each query, applying that method would produce better results than fusion. But since, in the current state of the art, one generally does not know which method is best for a given query, fusion of the results of multiple methods represents the best compromise.

Let’s consider several of the above examples briefly. Lee [SIGIR ‘95] combines the results of pairs of filtering methods. All of the filtering methods use the term-based vector space method (with the SMART system as a testbed). Each method differs from the others in the weighting scheme used. A weighting scheme (see section on “Classification of Term Vector Weighting Schemes”) is characterized by two three-character codes, one three character code specifying the weighting scheme applied to the documents in the target collection, the other the weighting scheme applied to the query. Lee argues theoretically and demonstrated experimentally that “different classes of weighting schemes may retrieve different types of documents - different sets of document (both relevant and nonrelevant).” Specifically, weighting schemes that employ cosine normalization of documents (he calls this class $C$) are better at retrieving single topic documents of widely varying length. On the other hand, weighting schemes (called class $M$) that employ maximum normalization of documents, i.e., normalization
of term frequency by maximum term frequency within a given document, but that do not employ cosine normalization, are better at retrieving those multi-topic documents in which only one of the topics is relevant to the given query. Hence (as one would expect), combining the results of a class \( C \) run with the results of a class \( M \) run produced significant improvement over the results of either run alone. However, it is evident that the extent of improvement (if any) is dependent on the characteristics of the collection and the query.

However, Lee defines a third class of weighting schemes, \( N \), consisting of schemes that use neither cosine nor maximum normalization. Such schemes tend to favor long documents over short documents. Surprisingly, combining a \( C \) run with an \( N \) run also produced improvement. As Lee summarizes, “we can get significant improvements by combining two runs in which one performs cosine normalization and the other does not if the two runs provide similar levels of filtering effectiveness.” Lee notes that “the combinations between class \( C \) and the other classes have less common documents than those between classes \( M \) and \( N \), which means that cosine normalization is a more important factor than maximum normalization in retrieving different sets of documents.”

It should be noted that, as discussed above in the section on normalization of term vectors, the pivoted unique normalization (\( L_{nu} \)) scheme developed by Singhal et al. appears to achieve the same kind of improvement with a single filtering run that Lee achieves by fusion of output from multiple runs, each run using a different, older weighting scheme. In other words, \( L_{nu} \) appears to achieve (and perhaps improve on) the combined benefits of several older weighting schemes.

Finally, Lee combined an extended boolean (\( p \)-norm) run with a vector run, and achieved improvement both by combining \( p \)-norm with a \( C \) class and by combining \( p \)-norm with an \( M \) class vector weighting scheme.

In all of the above cases, each individual run produces a list of documents ranked by similarity. In each combined run, the results of the participating individual runs are combined so that each retrieved document receives a combined similarity score, and the documents are ranked by these combined scores. In all cases, Lee chooses the top-ranking \( N \) documents (\( N = 200 \)), i.e., the documents with the top \( N \) similarity scores. The problem is how to compute the combined similarity for a given document. The vector space runs generate document rankings based on cosine similarity values (similarity of
each document to the given query). The extended boolean runs generate a similarity score based on the \( p \)-norm model. The range of possible similarity values for cosine similarity or \( p \)-norm is always zero to one. However, the range of actual similarity values for the same query applied to the same document collection will be different for each model and weighting scheme. Hence, the similarity values must be normalized to make them comparable. The formula:

\[
\text{Normalized Similarity} = \frac{\text{old\_sim} - \text{minimum\_sim}}{\text{maximum\_sim} - \text{minimum\_sim}}
\]

converts each similarity, \( \text{old\_sim} \), calculated for a given query in a given individual filtering run, to a value in the common range zero to one, i.e., the largest similarity value, \( \text{maximum\_sim} \), will be mapped into one, the smallest similarity value, \( \text{minimum\_sim} \), will be mapped into zero, and all intermediate similarity values will be mapped into values between zero and one. (Lee also notes that if one knew in advance which filtering runs were likely to perform better, it would make sense to weight the similarity values of those runs more heavily; however, in general, for ad-hoc queries and arbitrary collections, one doesn’t have that kind of information.) Once all the similarity values have been calculated for the runs to be combined, the filtering sets could be merged in straightforward numeric order by (normalized) similarity value, and the \( N \) documents with the highest similarity returned.

However, straightforward numeric merging by normalized similarity value has the drawback that it does not take into account the number of filtering sets in which a given document occurs. As noted above, the more filtering sets in which document \( D_i \) occurs, the more likely it is to be relevant to the given query. Hence, Lee, following Fox & Shaw [TREC-2, 1994], computes a combined similarity value for each document equal to the sum of its similarity values in each filtering set in which it occurs. (Naturally, the similarity of a given document is zero in a filtering set in which it does not occur.) Documents are then ranked by these combined similarity values, and the top \( N \) selected as before. Note that ranking each document by the sum of its similarity values is equivalent to ranking the given document by the mean of its similarity values.

Fox and Shaw [TREC-2, 1994] combined the results of extended boolean (\( p \)-norm) query runs with the results of “natural language vector query” runs, i.e., vector queries obtained by extracting and stemming terms in the usual way from natural language.
topic statements. As noted above, they merged the results of multiple filtering runs for a given query by computing a combined similarity value for each document retrieved in at least one run. In addition to the sum (or equivalently mean) of the similarity values (which they call “CombSUM”), they also tried two other methods of combining similarities: In their “CombANZ” method, they divide the CombSUM value by the number of filtering runs in which the document received a non-zero similarity; the effect is to compute a mean that ignores filtering runs in which the given document is not retrieved. In their “CombMNZ” method, they multiply the CombSUM value by the number of non-zero similarities the given document received; the effect is to enhance the importance of filtering of a given document by multiple runs.

Fox and Shaw ran five individual filtering runs. Three of these runs used $p$-norm extended boolean queries, each with a different value of $p$ (1.0, 1.5, and 2.0). The other two runs used vector queries. The vectors were generated from TREC-2 natural language topic descriptions. The “short vector” run took its query terms from the Title, Description, Concepts, and Definitions sections of the standard TREC-2 topic format. The “long vector” run took its query terms from all of those sections plus the Narrative section as well. In contrast to Lee, who only combined pairs of filtering runs, Fox and Shaw combined all five of their individual filtering runs as well as combining two or three individual runs. This led them to the following interesting observation: “While combining all five runs produced an overall improvement in filtering effectiveness over each of the [individual] runs, the same does not always hold true when combining only two or three runs.” Thus, the effectiveness of combining runs can depend not only on the query and the collection, but also on how many runs are combined and which ones.

In later research, Lee [SIGIR ’97] builds on the work of Fox and Shaw. He studies combinations of up to six individual filtering runs, using results derived from TREC-3. In his own previous work [SIGIR ’95], described above, he showed theoretically, and demonstrated experimentally, that two different term weighting schemes, appropriately chosen, could result in retrieving different relevant documents from the same collections, even when the weights are applied to term vectors in both cases, and cosine similarity is the method used to compute the individual query-document similarity in all cases. Improvement resulted in his experiments provided the two methods contrasted appropriately, and were equally effective. Even in that work, he also allowed for the
effect of multiple runs retrieving the same document by using one of Fox and Shaw’s combination functions, CombSUM, to compute the combined similarity of a document that was retrieved by both runs. In the SIGIR’97 work, he finds that the improvement that results from combining multiple TREC runs derives primarily from the fact that the runs tend to retrieve the same relevant documents (which pushes up the combined similarity of each document retrieved by more than one run), but different non-relevant documents (which pushes down their combined similarity scores). He went beyond Fox and Shaw by (1) normalizing the similarities, using his SIGIR ‘95 formula given above, so that the similarity scores combined would be more comparable, (2) by showing that CombMNZ (defined above), which emphasizes the importance of being retrieved by multiple runs, gives even better results than CombSUM, and (3) by computing the actual amount of overlap across individual runs, for relevant and non-relevant documents. Lee computes the overlap on a scale running from zero (no overlap) to one (total overlap), using the functions: \( R_{\text{overlap}} = \frac{R_{\text{common}}}{2} \), \( R_1 + R_2 \) and \( N_{\text{overlap}} = \frac{N_{\text{common}}}{2} \), \( N_1 + N_2 \). He finds values of \( R_{\text{overlap}} \) in the range 0.74 to 0.82, values of \( N_{\text{overlap}} \) in the range of 0.30 to 0.40. Plainly, the proportion of overlap among retrieved relevant documents is much higher than the proportion of retrieved non-relevant documents.

Turtle and Croft [ACM Trans IS, 1991] used the inference network approach (see section 7.3.2) to combine Boolean and term-based (they call the latter “probabilistic”) queries for the same information need. Both query formulations were based on an initial natural language statement of the problem. The queries were combined using a weighted sum. They found that the combination produced better results (better precision at most recall levels) than either type of query formulation by itself. However, they found that the improvement was due to the fact that the boolean query formulators used the boolean structure to capture information present in the natural language statement of the information need that was lacking in the term-based query. Hence, the boolean query retrieved a subset of the documents retrieved by the term-based query. Adding the boolean query to the term-based query produced not additional documents but a better ranking of the documents retrieved by the term-based query, resulting in better precision at a given recall level. They conjecture that if the boolean formulators had been asked to produce high-recall boolean queries, they would have added additional terms, and retrieved additional documents not retrieved by the term-based queries. It should
be stressed that Turtle and Croft were combining two query formulations to produce a single query, and then running this single combined query against their filtering engine. They were not combining the filtering sets returned by the two queries run separately.

Belkin et al. [SIGIR ‘93] generated extended boolean queries which were executed using the INQUERY system’s extended boolean operators which are similar to, but not identical to, the operators of the P-norm model. They “recruited experienced on-line searchers to generate search statements for the same search topics.” The recruits were told to generate boolean queries using AND, OR, NOT, any degree of nesting desired, and operators (at the word level only) for adjacency, i.e., two terms next to each other, proximity, i.e., terms within a given distance of each other, and order, i.e., terms in proximity to each other and occurring in a specified order. They were not told the system (INQUERY) to which the queries were to be submitted or that the queries were to be executed as extended booleans. The queries were executed separately, and then in combination. However, in contrast to Lee, and Fox and Shaw, as described above, Belkin et al. did not combine the results (filtering sets) of the individual query runs; instead, they used INQUERY’s ability to combine the queries themselves. The queries were combined using the INQUERY “unweighted sum” operator. (Actually, the experiment was more complex. They started with five query “groups”, each group consisting of a query for each of ten TREC-2 topics.) The queries were combined cumulatively, e.g., first a single query group, then two of the query groups (for the same set of search topics) combined, then three, then four, and then all five query groups for the given search topic combined. Results were reported for group 1, then group 1 plus group 2, etc. Unweighted sums were used to combine queries across groups. Combination within a group is not discussed. Then the combination of all five boolean queries was further combined with a natural language query based on the corresponding TREC-2 natural language topic description. This natural language query provides a powerful baseline because it takes advantage of a “version of INQUERY [that] performs a sophisticated analysis of the TREC topics … including recognition of country names and automatic syntactic phrase generation …” Combination of Boolean queries (translated into INQUERY) and the corresponding natural language baseline INQUERY queries (designated “INQC” by Belkin et al.) was done with various weighted sums.
The results obtained by Belkin et al. indicated that combining Boolean queries (actually query groups - see above) improved performance. An interesting point they note is that in some cases adding a group that performed poorly on its own to a group that performed well on its own resulted in better performance than the good group by itself. Hence, one cannot always judge the performance of a combination of methods solely by evaluating the methods separately. However, combining the Boolean queries with the INQC queries reduced performance when the boolean and the INQC queries were given equal weight. Significantly improved performance was obtained only when the INQC query was given a weight four times that of the combined boolean query. By contrast, Fox and Shaw obtained significant improvement (see above) when natural language queries were combined with extended boolean queries. However, as Fox and Shaw note, the methods of combination are not strictly comparable because Belkin et al. combine queries while Fox and Shaw combine output filtering sets.

Foltz and Dumais [CACM, 1992] combine two vector space methods: key-word, i.e., terms are words, and LSI, i.e., terms are LSI factors. Their application is a routing/filtering rather than an IF application. The task they address is to assign abstracts of incoming technical reports to users based on user profiles, and document profiles. A user profile is a list of words and phrases provided by the user to characterize her technical interests. The document profile for a given user is the set of abstracts that the given user has previously rated as highly relevant to her interests. Hence, the query or information need for a given user is the user’s profile or her document profile. The document “collection” is the stream of incoming technical reports. Hence, there are four IF methods:

1. Vector space filtering by calculating the similarity of an incoming abstract and a given user profile,

2. Vector space filtering by calculating the similarity of an incoming abstract to abstracts previously rated highly relevant by a given user,

3. Same as (1) but with similarity calculated in the reduced dimension LSI space.

4. Same as (2) but with similarity calculated in the reduced dimension LSI space.

Methods were combined by sending each user monthly the top seven abstracts selected by each of the four methods. This meant that each user could receive up to 28 abstracts per
month. But since a given abstract could be selected by more than one method, the users actually received an average of 17 abstracts per month. Foltz and Dumais found that as the number of methods that retrieved a given abstract increased, the “mean relevance rating” increased too. (Each user was asked to rate the abstracts she received each month for relevance on a scale from one for non-relevant to seven for very relevant.) The rating went up from about three for an abstract selected by one or more methods to five for an abstract selected by all four methods. But of course, the number of abstracts selected by all four methods was much less (about 5%) than the number of abstracts returned to all users. However, ratings also improved if one was more selective for a given method, e.g., only selecting abstracts above a given similarity threshold. So the mean rating for abstracts selected by all four methods was compared to the mean rating for the top 5% of abstracts selected by each method separately. The mean rating for documents selected by all four methods together still came out on top, though not by as much, e.g., a mean of 4.54 for abstracts selected by a single method vs. 5.04 for abstracts selected by all four methods.

All of the above cases involve combination of IF algorithms or query formalisms, e.g., boolean and vector space, vector space with two different weightings, etc. However, if a training set of documents with relevance judgments is available (as is often the case in routing and filtering applications), one can make use of general methods of machine learning, methods not specific to IF. Each method can be trained to classify documents using the training set. The result is a set of predictive models, one for each learning technique. These predictive models can be combined just as traditional IF methods are.

For example, Hull et al. [SIGIR ‘96] study the combination of four methods: Only the first, Rocchio query expansion based on relevance feedback, derives from the IF field. The other three are general purpose “learning” methods, employed to generate a predictive model for document classification. They are: “Nearest Neighbors,” Linear Discriminant Analysis (LDA),” and a “Neural Network” fitting a logistic model. Hull et al. study a filtering application. Hence, each predictive model must classify each incoming document as either relevant (accept the document), or non-relevant (discard the document. Each of the four resulting models, given a document to classify, generates a probability-of-relevance score. Hull et al. try several approaches to combining the scores for a given document. (1) Most simply, they compute a straight average of the scores. (2) Next, “given that they are working in a probabilistic domain,” they average the
logodds ratios, and then reconver this average back to a probability. (Given a score interpreted as a probability, $p$, the logodds ratio is defined as $\log(p/(1-p))$. See the section on the Probabilistic Approach.) They point out that probabilities derived by straight averaging will tend to have much less variability than probabilities derived from averaging of logodds ratios. In particular, if one of the classifiers is very certain of the relevance (or non-relevance) of a given document, the probability derived from logodds ratio averaging will be very close to one (or zero). In general, this is not the case with straight probability averaging. Hence, logodds ratio averaging will reflect the certainty of an individual classifier more clearly and directly than straight probability averaging. Both straight probability averaging and logodds ratio averaging were found to outperform the individual classifiers for ranking documents, but for a filtering application, where the important criterion is accurate calculation of relevance probability (or other similarity score) relative to a filtering threshold, the neural network classifier outperformed both classifiers. (3) Hence, to improve calculation of average probabilities, Hull et al. “renormalized the probability estimates via logistic regression using the relevance judgments from the training set.” They found that “after normalization, the probability estimates [were] much more accurate, scoring significantly better than the neural network” except at very low thresholds.

All of the above examples of fusing different IF methods involve fusion of a small number of manually selected methods. Bartell et al. [SIGIR ‘94] have developed a method for automatically combining “experts,” i.e., modules executing different IF methods. The method involves a heuristic gradient-based search over the space of possible combinations and can be applied to a large number of experts (although the two tests of the method discussed in their paper involve two experts and three experts respectively). The method is independent of how each expert performs its IF task; the only requirement, satisfied by an increasing number of IF systems, is that the experts must all return ranked output, i.e., each system must return a numerical estimate of the degree of relevance of each retrieved document to the given query. A notable feature of this fusion method is that it optimizes the combined output of all the participating experts, rather than evaluating the performance of each expert separately. This is significant in light of the finding of Belkin et al. (noted above) that one cannot always judge whether a given IF method will make a positive contribution to combined performance based solely on evaluating the given method separately.
In the Bartell model, each expert $i$ returns a numeric estimate $E_i(Q, D)$ of the degree of relevance of document $D$ to query $Q$. They combine these estimates into a single overall estimate, $Re(Q, D)$ of the degree of relevance of $D$ to $Q$. In this paper, they use a linear combination of the estimates, e.g., for three experts, they have:

$$Re(Q, D) = \Theta_1 \cdot E_1(Q, D) + \Theta_2 \cdot E_2(Q, D) + \Theta_3 \cdot E_3(Q, D)$$

Their goal is then to find values of the parameters $\Theta_i$ “so that the overall estimates result in the best ranking of documents possible.” Optimization is based on a training set, i.e., a training set of documents, a set of training queries, and a relevance judgment for every document retrieved by a given training query. The relevance judgment is expressed as a preference relation, i.e., user prefers $D_1$ to $D_2$ for any pair of documents retrieved by a query $Q$. (This is the same kind of preference relation used by Rhagavan and Sever [SIGIR ‘95] as discussed above in regard to reuse of optimal queries.) If only the usual two-valued judgment, relevant or non-relevant, is available, then the preference relation reduces to preferring the relevant to the non-relevant document. “The goal of the optimization is to find parameter values such that the [combined] system ranks document $D_1$ higher than document $D_2$ whenever $D_1$ is preferred by the user to $D_2$.” A gradient-based numerical optimization technique is used. (Note that this is very similar to Rhagavan and Sever’s use of the preference relation in a “steepest descent” search for an optimum query in query space.)

Most of the examples above combined two or more document classifiers and studied the performance of the combination relative to that of the individual classifiers. Lee also identified certain cases where combination would be effective, and factors contributing to the success of actual experimental combinations. However, Vogt et al. [SIGIR ‘98] studied more comprehensively the factors contributing to effective combination. They limited their study to linear combinations, and also limited themselves to combinations of two classifiers. The combinations they studied were derived from TREC5 ad hoc query data. Since there were 61 entries in the ad hoc competition, Vogt et al. were able to form $(61*6)/2 = 1830$ pairs for each query. They studied 20 queries for a grand total of $1830*20 = 36,600$ “IF system (method) combinations.” Although they drew their data from the ad hoc query competition, the fact that they combined systems on a per-query basis means that the results are more applicable to the routing application.
Their theoretical approach was to identify a set of method performance features. Some of the features were measures of the performance of an individual system (method), e.g., average precision. Others were pairwise measures. For example, Guttman’s Point Alienation (GPA) is a measure of how similar two document rankings are to each other. Another pairwise measure employed was the intersection, i.e., the number of documents retrieved by both methods. Following Lee, they also computed $R_{overlap}$ and $N_{overlap}$ (see above). The former measures the proportion of relevant documents retrieved by both systems; the latter measures the proportion of non-relevant documents retrieved by both systems. They then performed a multiple linear regression, using the actual TREC5 data as the training set, the method performance features computed from this set (for all system pairs and a given query) as the independent, i.e., predictor, variables of the linear regression equation, and the average precision of the optimal combination (for the given query) as the dependent variable of the equation, the variable to be predicted. The regression then computes coefficients for the predictor variables in this equation. These coefficients can be interpreted as indicating how much each predictor contributes to the overall estimate of the dependent variable.

The results they obtained indicate that the best time to combine two systems (methods) linearly is when (1) at least one system exhibits good performance, (2) both systems return similar sets of relevant documents, and (3) both systems return dissimilar sets of non-relevant documents.

### 4.3 Fusion of Results Obtained by Multiple Versions of the Same Method

In the previous section, we discussed techniques for combining multiple classification methods, where the training set used to set the parameters of each method was the same, but the IF or machine learning algorithm was different for each classifier. In this section, we discuss approaches where the training set, the machine learning (ML) method, and the underlying IF method are the same, yet multiple classifiers are obtained. This is accomplished, e.g., by taking multiple samples from the training set (“resampling”) with replacement and using each sample as a new training set (“bagging”), or by weighting the training documents differently in each training session.
Note that since reweighting the training set is equivalent to changing the number of occurrences of each document in the training set, boosting can also be viewed as a “resampling” method. A number of variants of these approaches are known, mostly derived from the machine learning community. Breiman [TR, 1996] characterizes this entire family as Perturb and Combine (P&C) methods, i.e., perturb the training set a number of times to create a number of new training sets, generate a classifier for each training set created by perturbation, and then combine these classifiers.

In bagging, [Breiman, ML, 1996] one selects $N$ documents at random from the training set “with replacement,” where $N$ is the size of the training set. The phrase “with replacement” means that after each document is taken, it is (in effect) put back, so that all the documents of the training set are available the next time a document is selected. In other words, each document is taken from the full, original training set. Since $N$ documents are selected, the “new” training set will be exactly the same size as the original training set. However, since each of the $N$ documents is chosen at random from the original set, some documents may be chosen more than once, while others may not be chosen at all. Hence, the new training set will be different from the original. This procedure can be repeated as many times as desired, to produce a set of training sets, each of size $N$. Each training set is chosen independently of the others, so that the order in which the training sets are chosen, or used for training, is immaterial. Each training set is then used to train a classifier, using the same IF or ML method. Hence, a set of classifiers (commonly called an “ensemble” in this context) is generated. Each classifier is then executed against any new document, and their results are combined. A common method of combination is “voting,” i.e., if the classifiers have been trained to determine whether a new document $d$ belongs to class $C$ or not, the choice is “yes” if more classifiers choose $C$ than choose “not $C$.” Another common method is to average the classification scores produced by all the classifiers in the ensemble. (Of course, $C$ may be relevance to a given topic $T$. On the other hand, the classifiers may be trained to select among multiple classes, $C_1, C_2,$ etc.)

Breiman [TR, 1996] argues that the main effect of bagging is to reduce classification error due to variance. This is the degree to which the classification estimate varies with the data the classifier is being asked to classify. [Witten et al., DM] [Opitz et al., 1999] [Friedman, DM&KD] In other words, it is a measure of how dependent the classifier
is on the particular training set chosen, which may be unrepresentative of the larger population the classifier may be required to judge. (This overdependence on the training set is called “overfitting.”) Opitz et al. [1999] following Bauer and Kohavi [1999], argue that bagging also reduces bias error, the average difference between the output of the classifier and the output of the “target” function the classifier is trying to learn.

Boosting, in contrast to bagging, generates a series of classifiers, ordered in the sense that each classifier is generated based on the performance of earlier classifiers in the series. [Opitz et al., 1999] In a powerful version of boosting called Ada-Boosting [Schapire et al., 1998], each classifier is trained on the same training set, using the same IF algorithm. However, the documents in the training set are weighted, and the weights assigned to the training set for generating classifier $CL_i$ are based on the performance of the previous classifier $CL_{i-1}$. Specifically, after each classifier, $CL_{i-1}$, is executed, the documents in the training set are reweighted so that the weights of the documents that it misclassified are increased, and the weights of the documents it classified correctly are decreased. This new weight vector and the associated training set are the input for training $CL_i$. Hence, it is hoped, $CL_i$ will be better at classifying the documents that were previously misclassified. This process is repeated for $T$ iterations, resulting in $T$ classifiers, $CL_1$ to $CL_T$. (T is chosen by an ad hoc rule.) A weight $w_i$ is assigned to each classifier $CL_i$. The final “boosted” classifier for classifying new documents is a weighted vote of these $T$ classifiers. That is, for a given new document $d$, each classifier $CL_i$ votes +1 if it classifies $d$ as relevant, -1 if it classifies $d$ as non-relevant. The vote of each classifier $CL_i$ is multiplied by its weight, $w_i$. These $T$ weighted votes are then summed. The classification of $d$ is relevant if the sum is positive, non-relevant if the sum is negative. Note that if the weights are all equal (which they usually are not), this is equivalent to classification by majority vote.

The underlying IF algorithm, which may be any simple algorithm chosen by the developer, is called a “weak learner.” The goal of boosting is to combine a set of weak learners into a single “strong learner.” (The terms “weak learner” and “strong learner” have technical definitions. [Breiman, TR, 1996]) The weak learner that Schapire et al. use is the presence or absence of a term in a given document.
term is present, the document is assumed to be relevant, i.e., belongs to the class for which the classifier is being developed. If the term is absent, the document is assumed to be non-relevant. The algorithm “learns” from the training set at stage \( i \) by choosing the term \( t \) that minimizes the misclassification error, \( \text{err}_i(t) \). The error \( \text{err}_i(t) \) is defined as the sum of the weights of documents in the training set that either contain the term \( t \) but are non-relevant, or do not contain \( t \) but are relevant. Schapire et al. define a “term” to be either a single word, or a bigram, i.e., two consecutive words. Actually, the learner doesn’t always choose the \( t \) that minimizes \( \text{err}_i(t) \). Rather, it chooses the optimum term \( t = t_{opt} \) that minimizes either \( \text{err}_i(t) \) or \( 1 - \text{err}_i(t) \), because a term with a very high misclassification error is distinguishing relevant from non-relevant documents just as well as a term with a very low error; it is just getting its classification decisions reversed, consistently calling relevant documents non-relevant, and non-relevant documents relevant. (By contrast, a term with a misclassification error close to 1/2 is very poor at distinguishing relevant from non-relevant documents.) Hence, classifier \( CL_i \) follows the simple rule that a new document is relevant if it contains \( t_{opt} \) and non-relevant if it does not contain \( t_{opt} \).

The document weights are maintained as a probability distribution over the training set, i.e., the sum of the weights always equals one. (Hence, the misclassification error may be interpreted as a probability of misclassification.) So initially, each document in the training set is given a weight of \( 1/N \) where \( N \) is the number of documents in the training set. Thereafter, each time the documents are reweighted, the weights are also normalized so that their sum remains equal to one. The documents are reweighted at stage \( i \), based on whether they were correctly classified or misclassified by the \( i \)-th classifier, \( CL_i \), which learned using the \( i \)-th set of weights. Each document that was correctly classified by \( CL_i \) has its weight multiplied by \( e^{-a_i} \), and each document that was misclassified by \( CL_i \) is multiplied by \( e^{a_i} \), where \( a_i \) is defined as \((1/2) \ln((1-e_i)/e_i)\). The effect is that if the error \( e_i \) is minimized, the smaller it is the larger \( a_i \) is, and correspondingly, the more drastically document weights are modified, the weights of correctly classified documents going down, and the weights of misclassified documents going up. Hence, the next classifier generated, \( CL_{i+1} \), will be tend to be better at classifying the documents that were misclassified by \( CL_i \). On the other hand, if \( 1 - e_i \) is
minimized, $e_i$ is maximized. If $e_i$ is $>$ 1/2, $a_i$ will be negative. The closer $e_i$ is to one, the more negative $a_i$ will be, and correspondingly, the more drastically document weights will be modified in the opposite direction, weights of correctly classified documents going up, and weights of misclassified documents going down. (In other words, if the term is present in a document, its weight will go down if the document is, in fact, non-relevant.)

A explained above, after $T$ iterations, the resulting $T$ classifiers, $CL_i$ to $CL_T$ are combined by weighted voting. Each classifier $CL_i$ is multiplied by a weight $w_i$. This weight $w_i = a_i$ as defined in the previous paragraph. Since $a_i$ is a very large positive number for very low classification errors, i.e., for very good classifiers, a very large negative number for classification errors close to one, i.e., for classifiers that misclassify consistently, and close to zero for classifiers that don’t do any-thing consistently, the effect is to weight classifiers by their effectiveness.

How many iterations $T$, i.e., how many classifiers, does boosting require? Schapire et al. have no theoretical basis for choosing the value of $T$, so they use a simple empirical rule: Iterate until the training error reaches a minimum (which may be zero). Call this number of iterations, $T_0$. Then run another 10%, i.e., run $(1.1,)T_0$ iterations, generating $(1.1,)T_0$ classifiers. Note that since each classifier distinguishes relevance from non-relevance on the basis of a single term, the effect of generating $(1.1,)T_0$ classifiers is to create a final weighted vote classifier based on $(1.1,)T_0$ terms, $(1.1,)T_0$ document features. In general, the harder the classification “problem,” i.e., the harder it is to learn to recognize the target class, the greater $T_0$ will be, and hence the greater the number of features in the final classifier.

The boosting scheme described above makes use of document weights, and (at the final stage) classifier weights, but it does not make use of term weights, as most traditional IF algorithms do. Schapire et al. say that they are studying ways of incorporating term weights into their boosting algorithm.

One easily remedied problem with boosting as described above is that it gives equal credit to classifying relevant and non-relevant documents correctly. In practice, it is often more important to recognize relevant documents. For example, if there are very few relevant documents, a “dumb” classifier that classifies every document as non-relevant
will exhibit a very low classification error, although it will certainly not be very useful! Hence, it is desirable to tell the boosting algorithm that correct classification of relevant documents is more useful than correct classification of non-relevant documents. Schapire et al. accomplish this very simply by modifying the initial distribution of weights. Specifically, instead of giving each document an initial weight of $1/N$, each relevant document is given a weight of $(u_{rel+} - u_{rel-})/Z_0$, and each non-relevant document is given a weight of $(u_{nrel-} - u_{nrel+})/Z_0$, where $u_{rel+}$ is the utility of classifying a relevant document correctly, and $u_{rel-}$ is the utility (in this case, the harm) of misclassifying a relevant document. Similarly, $u_{nrel-}$ is the utility of correctly classifying a non-relevant document, and $u_{nrel+}$ is the utility (i.e., harm) of misclassifying a non-relevant document. $Z_0$ is a normalization factor, set so that the sum of all the initial weights is one as usual. By adjusting these four initial weights, the relative utility of correctly classifying relevant and non-relevant documents can be set as desired.

Breiman [TR, 1996] characterizes the family of boosting algorithms, including Ada-Boosting, as Adaptive Resampling and Combining or *arcing* algorithms. It has already been observed that both boosting and bagging involve *resampling* (to provide multiple training sets from the original set) and combining of the classifiers generated from these multiple sets, e.g., by averaging or weighted voting. The term “adaptive” refers to the fact that in boosting algorithms, each classifier learns from the performance of previous classifiers. Breiman hypothesized that the effectiveness of arcing came from use of adaptive sampling, and *not* from the particular reweighting function employed. To demonstrate this, he experimented with other reweighting approaches. He found that a reweighting method which he dubbed arc-x4 worked as well as Ada-boosting (which he dubbed arc-fs (in honor of the original developer, Freund and Schapire)).

In arc-x4, the reweighting of a given document $d_j$ at the $i$-th stage (to generate the weights that will be used for training classifier $CL_{i+1}$) depends on the number of times $d_j$ has been misclassified by *all* the $i$ classifiers generated up to that point. Specifically, the weight of document $d_j$ (or equivalently the probability $p_j$ that $d_j$ will be selected for the training set of classifier $CL_{i+1}$) is given by:
where $m_j$ is the number of times $d_j$ has been misclassified by all of the previous $i$ classifiers, and $N$ is the number of documents in the training set. After $T$ iterations, the resulting $T$ classifiers are combined by unweighted voting.