2. Approaches to IF – General

Broadly, there are two major categories of IF technology and research: semantic and statistical. Semantic approaches attempt to implement some degree of syntactic and semantic analysis; in other words, they try to reproduce to some (perhaps modest) degree the understanding of the natural language text that a human user would provide. In statistical approaches, the documents that are retrieved or that are highly ranked are those that match the query most closely in terms of some statistical measure. By far the greatest amount of work to date has been devoted to statistical approaches so these will be discussed first. (Indeed, even semantic approaches almost always use, or are used in conjunction with, statistical methods. This is discussed in detail later.)

Statistical approaches fall into a number of categories: boolean, extended boolean, vector space, and probabilistic. Statistical approaches break documents and queries into terms. These terms are the population that is counted and measured statistically. Most commonly, the terms are words that occur in a given query or collection of documents. The words often undergo pre-processing. They are “stemmed” to extract the “root” of each word. [Porter, Program, 1980] [Porter, Readings, 1997] The objective is to eliminate the variation that arises from the occurrence of different grammatical forms of the same word, e.g., “retrieve,” “retrieved,” “retrieves,” and “retrival” should all be recognized as forms of the same word. Hence, it should not be necessary for the user who formulates a query to specify every possible form of a word that he believes may occur in the documents for which he is searching. Another common form of preprocessing is the elimination of common words that have little power to discriminate relevant from non-relevant documents, e.g., “the”, “a”, “it” and the like. Hence, IF engines are usually provided with a “stop list” of such “noise” words. Note that both stemming and stop lists are language dependent.

Some sophisticated engines also extract “phrases” as terms. A phrase is a combination of adjacent words which may be recognized by frequency of co-occurrence in a given collection or by presence in a phrase dictionary.

At the other extreme, some engines break documents and queries into “n-grams”, i.e., arbitrary strings of \( n \) consecutive characters. [Damashek, 1995] This may be done, e.g., by moving a “window” of \( n \) characters in length through a document or query one character at a time. In other words, the first \( n \)-gram will consist of the first \( n \) characters
in the document, the 2nd \( n \)-gram will consist of the 2nd through the \((n+1)\)th character, etc. (Early research used \( n = 2, n = 3 \); recent applications have used values of \( n=5 \), and \( n=6 \) but the user is free to use the value of \( n \) that works best for his application.) The window can be moved through the entire document, completely ignoring word, phrase, or punctuation boundaries. Alternatively, the window can be constrained by word separators, or by other punctuation characters, e.g., the engine can gather \( n \)-gram statistics separately for each word. [Zamora et al., IP&M, 1981] [Suen, IEEE Pattern, 1979] Thirdly, \( n \)-grams can be gathered and counted without regard to word boundaries, but then words or phrases can be evaluated in terms of \( n \)-gram statistics. [Cohen, 1995] In any case, since a single word or phrase can generate multiple \( n \)-grams, statistical clustering using \( n \)-grams has proved to be language-independent, and has even been used to sort documents by language, or by topic within language. For similar reasons, \( n \)-gram statistics appear to be relatively insensitive to degraded text, e.g., spelling errors, typos, errors due to poor print quality in OCR transmission, etc. [Pearce et al., 1996]

Numeric weights are commonly assigned to document and query terms. A weight is assigned to a given term within a given document, i.e., the same term may have a different weight in each distinct document in which it occurs. The weight is usually a measure of how effective the given term is likely to be in distinguishing the given document from other documents in the given collection. The weight is often normalized to be a fraction between zero and one. Weights can also be assigned to the terms in a query. The weight of a query term is usually a measure of how much importance the term is to be assigned in computation of the similarity of documents to the given query. As with documents, a given term may have a different weight in one query than in another. Query term weights are also usually normalized to be fractions between zero and one.

2.1 Classical Boolean Approach to IF

In the boolean case, the query is formulated as a boolean combination of terms. A conventional boolean query uses the classical operators AND, OR, and NOT. The query “\( t_1 \) AND \( t_2 \)” is satisfied by a given document \( D_j \) if and only if \( D_j \) contains both terms \( t_1 \) and \( t_2 \). Similarly, the query “\( t_1 \)
OR $t_2$” is satisfied by $D_1$ if and only if it contains $t_1$ or $t_2$ or both. The query “$t_1$ AND NOT $t_2$” satisfies $D_1$ if and only if it contains $t_1$ and does not contain $t_2$. More complex boolean queries can be built up out of these operators and evaluated according to the classical rules of boolean algebra. Such a classical boolean query is either true or false. Correspondingly, a document either satisfies such a query (is “relevant”) or does not satisfy it (is non-relevant”). No ranking is possible, a significant limitation. [Harman, JASIS, 1992] Note however that if stemming is employed, a query condition specifying that a document must contain the word “retrieve” will be satisfied by a document that contains any of the forms “retrieve”, “retrieves”, “retrieved”, “filtering”, etc.

Several kinds of refinement of this classical boolean query are possible when it is applied to IF. First, the query may be applied to a specified syntactic component of each document, e.g., the boolean condition may be applied to the title or the abstract rather than to the document as a whole.

Second, it may be specified that the condition must apply to a specified position within a syntactic component, e.g., to the words at the beginning of the title rather than to any part of the title.

Third, an additional boolean operator may be added to the classical set; a “proximity” operator. [Z39.50-1995] A proximity operator specifies how close in the text two terms must be to satisfy the query condition. In its general form, the proximity operator specifies a unit, e.g., word, sentence, paragraph, etc., and an integer. For example, the proximity operator may be used to specify that two terms must not only both occur in a given document but must be within n words of each other; e.g., $n = 0$ may mean that the words must be adjacent. Similarly, the operator may specify that two terms must be within n sentences of each other, etc. A proximity operator can be applied to boolean conditions as well as to simple terms, e.g., it might specify that a sentence satisfying one boolean condition must be adjacent to a sentence satisfying some other boolean condition. A proximity operator may specify order as well as proximity, e.g., not only how close two words must be but in what order they must occur.

The classical boolean approach does not use term weights. Or, what comes to the same thing, it uses only two weights, zero (a term is absent) and one (a term is present).
The classical boolean model can be viewed as a crude way of expressing phrase and thesaurus relationships. For example, $t_1$ AND $t_2$ says that both terms $t_1$ and $t_2$ must be present, a condition that is applicable if the two terms form a phrase. If a proximity operator is employed, the boolean condition can be made to say that $t_2$ must immediately follow $t_1$ in the text, which corresponds still more closely (though still crudely) to the conventional meaning of a “phrase.” Similarly, $t_1$ OR $t_2$ says that either $t_1$ or $t_2$ can serve as an index term to relevant documents, i.e., in some sense $t_1$ and $t_2$ are “equivalent.” This is roughly speaking what we mean when we assign $t_1$ and $t_2$ to the same class in a thesaurus. In fact, some systems use this viewpoint to generate expanded boolean conditions automatically, e.g., given a set of query terms supplied by the user, “a boolean expression is composed by ORing each … query term with any stored synonyms and then AND-ing these clusters together.” [Anick, SIGIR ‘94]

### 2.1.1 Automatic Generation of Boolean Queries

The logical structure of Boolean queries, which is their greatest virtue, is also one of their most serious drawbacks. Non-mathematical or novice users often experience difficulty in formulating Boolean queries. [Harman, JASIS, 1992] Indeed, they often misinterpret the meaning of the AND and OR operators. (In particular, they often use “AND,” set intersection, when “OR,” set union, is intended.) [Ogden & Kaplan, cited in Ogden & Bernick, 1997] This has led to schemes for automatic generation of Boolean queries. [Anick, SIGIR’94] [Salton, IP&M, 1988]

In the Anick approach mentioned above, the query terms (presumably after stemming, removing stop words, etc.) are Ored together. Each OR term is expanded with any synonyms from an on-line thesaurus. The Salton approach, by contrast, imposes a Boolean structure on the terms supplied by the user. No thesaurus is employed.

Salton starts with a natural language query. The usual stemming and removal of stop words generates a set of user terms, which are ORed together as in the Anick approach. However, Salton then looks for pairs (and triples) of these user-supplied terms that co-occur in one or more documents. Since two or three of these user terms might occur in the same document by chance, Salton then uses a formula for pairwise correlation to determine if any given pair of co-occurring terms $T_i$ and $T_j$ co-occur more frequently than would be expected by chance alone. A similar correlation formula is used for
co-occurring triples, i.e., three of the user-supplied words occurring in the same
document. Each pair or triple whose computed correlation exceeds a pre-determined
threshold is then grouped with a Boolean AND, e.g., if the pair \( t_p, t_j \) and the triple \( t_p, t_m, t_n \)
exceed the threshold, then the automatically generated Boolean query (assuming \( t \) terms)
becomes:

\[
t_1 \text{ OR } t_2 \text{ OR } \ldots \text{ OR } (t_i \text{ AND } t_j) \text{ OR } (t_l \text{ AND } t_m \text{ AND } t_n) \ldots \text{ OR } t_t
\]

It should be stressed again that the pairs are triples are drawn entirely from the terms
originally supplied by the user; no thesaurus based expansion as with Anick, and no
query expansion based on relevance feedback (see below) is employed. However,
combining these various techniques is certainly feasible.

As a further refinement, Salton ranks the terms (single terms, pairs, and triples) in the
automatically generated Boolean expression in descending order by inverse document
frequency. (See below for a definition of \( \text{idf} \). High-\( \text{idf} \) terms tend to be better
discriminators of relevance than low- \( \text{idf} \) terms.) He can estimate the number of
documents that a given term (or pair or triple) will be responsible for retrieving from its
frequency of occurrence in documents. If the total estimated number of documents that
will be retrieved by the Boolean query exceeds the number of documents that the user
wants to see, he can reduce the estimated number by deleting OR terms from the query
starting with those that have the lowest idf ranking. This gives the user, not a true
relevance ranking of documents, but at least some control over the number retrieved,
something that ordinary Boolean filtering does not provide.

2.2 Extended Boolean Approach

Even with the addition of a proximity operator, boolean conditions remain classical in
the sense that they are either true or false. Such an all-or-nothing condition tends to
have the effect that either an intimidatingly large number of documents or none at all are
retrieved. [Harman, JASIS, 1992] Classical Boolean models also tend to produce
counter-intuitive results because of this all- or-nothing characteristic, e.g., in response to
a multi term OR, “a document containing all [or many of] the query terms is not treated
better than a document containing one term.” [Salton et al., IP&M, 1988] Similarly, in
response to a multi term AND, “[A] document containing all but one query term is treated just as badly as a document containing no query term at all.” [Salton et al., IP&M, 1988] A number of extended boolean models have been developed to provide ranked output, i.e., provide output such that some documents satisfy the query condition more closely than others. [Lee, SIGIR’94] These extended boolean models employ extended boolean operators (also called “soft boolean” operators).

Extended boolean operators make use of the weights assigned to the terms in each document. A classical boolean operator evaluates its arguments to return a value of either true or false. These truth values are often represented numerically by zero (false — or in IF terms “doesn’t match given document”) and one (true — or in IF terms “matches given document”). An extended boolean operator evaluates its arguments to a number in the range zero to one, corresponding to the estimated degree to which the given logical expression matches the given document. Lee [SIGIR’94] has examined a number of extended Boolean models [Paice, 1984] [Waller et al., 1979] [Zimmerman, 1991] and proved that by certain significant (but not necessarily the only significant) criteria, a model called “p-norm” [Salton et al., CACM 1983] has the most desirable properties. By “most desirable” is meant that the p-norm model tends to evaluate the degree to which a document matches (satisfies) a query more in accordance with a human user’s judgment than the other models. For each of the other models examined, there are cases where the model’s evaluation of the degree of query/document match is at variance with a human user’s intuition. In each of those cases, the p-norm model’s evaluation of match agrees with a human user’s intuition.

Given a query consisting of n query terms \( t_1, t_2, \ldots, t_n \) with corresponding weights \( w_{q1}, w_{q2}, \ldots, w_{qn} \) and a document \( D \), with corresponding weights \( w_{d1}, w_{d2}, \ldots, w_{dn} \) for the same \( n \) terms, the p-norm model defines similarity functions for the extended boolean AND and extended boolean OR of the \( n \) terms. The extended boolean AND function computes the similarity of the given document with a query that ANDs the given terms together. Similarly, the extended boolean OR function computes the similarity of the given document with a query that ORs the given terms together. Each similarity is computed as a number in the closed interval [0, 1]. More elaborate boolean queries can obviously be composed from the AND and OR functions. The extended boolean functions for the p-norm model are given by:
\[ \text{SIM}_{\text{AND}}(d, (t_1, w_{q_1}) \text{ AND} \ldots \text{AND} (t_n, w_{q_n})) = 1 - \left( \frac{\sum_{i=1}^{n} \left( (1 - w_{d_i})^p \cdot w_{q_i}^p \right)}{\sum_{i=1}^{n} w_{q_i}^p} \right)^{\frac{1}{p}}, (1 \leq p \leq \infty) \]

and

\[ \text{SIM}_{\text{OR}}(d, (t_1, w_{q_1}) \text{ OR} \ldots \text{OR} (t_n, w_{q_n})) = 1 - \left( \frac{\sum_{i=1}^{n} \left( w_{d_i}^p \cdot w_{q_i}^p \right)}{\sum_{i=1}^{n} w_{q_i}^p} \right)^{\frac{1}{p}}, (1 \leq p \leq \infty) \]

The \( p \)-norm model has a parameter that can be used to “tune” the model; most of the other models studied by Lee also have such a parameter, though the effect and interpretation of the parameter varies with the model. The parameter \( p \) in the \( p \)-norm model can vary from one to infinity and has a very clear interpretation. At \( p = \infty \), the \( p \)-norm model is equivalent to the classical boolean model; AND corresponds to strict phrase assignment (i.e., all the components of the phrase must be present for the AND to evaluate to non-zero), OR to strict thesaurus class assignment (i.e., presence of any one member of the class is sufficient for the OR to evaluate to one; there is no additional score if two or more are present.). At low to moderate \( p \), e.g., between 2 and 5, AND corresponds to loose phrase assignment, i.e., “the presence of all phrase components is worth more than the presence of only some of the components; terms are not compulsory.” That is, the \( p \)-norm AND generalizes the strict boolean AND in the sense that a single low-weighted term substantially lowers the total similarity score, even if all the other terms have high weights. On the other hand, the \( p \)-norm AND differs from the strict boolean AND because a single zero weighted, i.e., missing, term does not reduce the total similarity score to zero. Similarly, at low to moderate \( p \), OR corresponds to loose thesaurus class assignment, i.e., “the presence of several terms from a class is worth more than the presence of only one term.” In other words, the \( p \)-norm OR generalizes the strict boolean OR in the sense that a single high-weighted term can produce a fairly high total similarity score even if all the other terms are low-weighted or missing (zero weighted). On the other hand, the \( p \)-norm OR differs from the strict boolean OR because a single high-weighted term is not enough to maximize the similarity score; additional non-zero terms will increase the total score to some degree.
At $p = 1$, the $p$-norm model reduces to the pure “vector space” model which is discussed in the next section, i.e., “terms are independent of each other; distinction between phrase and thesaurus assignment disappears.” In fact, at $p = 1$, AND and OR become identical. [Salton, et al., CACM 1983] They both become identical to cosine similarity, discussed in the next section.

The classical boolean operators AND and OR are binary, i.e., they connect two terms. However, they are also associative, i.e., $t_1 \text{ AND } (t_2 \text{ AND } t_3)$ is equivalent to $(t_1 \text{ AND } t_2) \text{ AND } t_3$. This is not true for the $p$-norm model (and some of the other extended boolean models). The $p$-norm model (and the other models with the same problem) circumvent this difficulty by defining the extended boolean operators as nary, i.e., connecting $n$ terms, rather than binary. So the above boolean expression becomes \text{AND} \ (t_1, t_2, t_3).

{Lee, SIGIR ‘94] This expression is true if and only if all three terms are present.

The $p$-norm model supports assignment of weights to the query terms as well as the document terms. The $p$-norm formulas extend to this case in a quite straightforward manner. The weights are relative rather than absolute, e.g., the query $(t_1, 1) \text{ AND } (t_2, 1)$ with a weight of one assigned to each term is exactly equivalent to the query $(t_1, 0.1) \text{ AND } (t_2, 0.1)$ with a weight of 0.1 assigned to each term. This is so because the $p$-norm formulas normalize the query weights. Relative weights are easier and more natural for the user to assign than absolute weights. It is easier for a user to say that $t_1$ is more important (or even twice as important) than $t_2$ than to say exactly how important either term is. {Lee, SIGIR ‘94]

A further degree of flexibility can be achieved in the $p$-norm model by permitting the user to assign a different value of $p$ to each boolean operator in a given boolean expression. This allows the user to say, e.g., that a strict phrase interpretation should be given to one AND in the given expression, a looser interpretation to another AND in the same expression, etc. {Salton et al., CACM 1983]

What makes the $p$-norm model superior to the alternatives surveyed by Lee? Its primary advantage is that it gives equal importance to all its operands. This does not mean that it ignores document and term weights. On the contrary, the document weights (assigned typically by the program that indexes the document collection), and the query weights (assigned typically by the user who formulates the query, although automatic
modification of these weights is discussed in a later section), are essential elements of the $p$-norm functions as given above. What “equal importance” means is that the $p$-norm functions evaluate all term weights in the same way; they do not give special importance to certain terms on the basis of their ordinal positions, i.e., any permutation of term order is equivalent to any other. Moreover, $p$-norm does not give special importance to the terms with minimum or maximum weights, to the exclusion of other terms. For example, one or two high-weighted query terms in a given document will yield a high (relatively close to 1) value of a $p$-norm OR for the given document relative to the given query. It doesn’t matter in the least which terms are highly weighted. Moreover, if other query terms are also present in the given document, they will add to the value of the OR even if they have neither the maximum nor the minimum weight in the set of query terms matching the given document (“match” terms).

A probabilistic form of extended boolean has been developed [Greiff, SIGIR ’97] in which a probabilistic OR is computed in terms of the probability of its component terms, and similarly for AND. See section on “Bayesian Inference Network Model” for further details.

The commercial IF system, Topic, supports a form of extended boolean query called a “topic.” These queries can combine strict and extended boolean operators. See discussion in section 6.4. Experiments have shown that the extended boolean model can achieve greater IF performance than either the classical boolean or the vector space model. But there is a price. Formulating effective extended boolean queries obviously involves more thought and expertise in the query domain than formulating either a classical boolean query, or a simple set of terms with or without weights as in the vector space model.

2.3. Vector Space Approach

2.3.1 Building Term Vectors in Document Space

One common approach to document representation and indexing for statistical purposes is to represent each textual document as a set of terms. Most commonly, the terms are words extracted automatically from the documents themselves, although they may also be phrases, n-grams, or, manually assigned descriptor terms. (of course, any such term-based representation sacrifices information about the order in which the terms occur in the document, syntactic information, etc.) Often, if the terms are words extracted
from the documents, “stop” words (i.e., “noise” words with little discriminatory power) are eliminated, and the remaining words are stemmed so that only one grammatical form (or the stem common to all the forms) of a given word or phrase remains. (Stop lists and stemming can sometimes be avoided if the terms are $n$-grams — see discussion below.) We can apply this process to each document in a given collection, generating a set of terms that represents the given document. If we then take the union of all these sets of terms, we obtain the set of terms that represents the entire collection. This set of terms defines a “space” such that each distinct term represents one dimension in that space. Since we are representing each document as a set of terms, we can view this space as a “document space”. [Salt on, 1983] [Salton, 1989]

We can then assign a numeric weight to each term in a given document, representing an estimate (usually but not necessarily statistical) of the usefulness of the given term as a descriptor of the given document, i.e., an estimate of its usefulness for distinguishing the given document from other documents in the same collection. It should be stressed that a given term may receive a different weight in each document in which it occurs; a term may be a better descriptor of one document than of another. A term that is not in a given document receives a weight of zero in that document. The weights assigned to the terms in a given document $D_i$ can then be interpreted as the coordinates of $D_i$ in the document space; in other words, $D_i$ is represented as a point in document space. Equivalently, we can interpret $D_i$ as a vector from the origin of document space to the point defined by $D_i$’s coordinates.

In document space, each document $D_i$ is defined by the weights of the terms that represent it. Sometimes, it is desirable to define a “term space” for a given collection. In term space, each document is a dimension. Each point (or vector) in term space is a term in the given collection. The coordinates of a given term are the weights assigned to the given term in each document in which it occurs. As before, a term receives a weight of zero for a document in which it does not occur.

We can combine the “document space” and “term space” perspectives by viewing the collection as represented by a document-by-term matrix. Each row of this matrix is a document (in term space). Each column of this matrix is a term (in document space). The element at row $i$, column $j$, is the weight of term $j$ in document $i$.
A query may be specified by the user as a set of terms with accompanying numeric weights. Or a query may be specified in natural language. In the latter case, the query can be processed exactly like a document; indeed, the query might be a document, e.g., a sample of the kind of document the user wants to retrieve. A natural language query can receive the usual processing, i.e., removal of “stop” words, stemming, etc., transforming it into a set of terms with accompanying weights. (Again, stoplists and stemming are not applicable if the queries and terms are described using n-gram terms.) Hence, the query can always be interpreted as another document in document space. Note: if the query contains terms that are not in the collection, these represent additional dimensions in document space.

An important question is how weights are assigned to terms either in documents or in queries. A variety of weighting schemes have been used. Given a large collection, manual assignment of weights is very expensive. The most successful and widely used scheme for automatic generation of weights is the “term frequency * inverse document frequency” weighting scheme, commonly abbreviated “tf*idf”. The “term frequency” (tf) is the frequency of occurrence of the given term within the given document. Hence, tf is a document-specific statistic; it varies from one document to another, attempting to measure the importance of the term within a given document. By contrast, inverse document frequency (idf) is a “global” statistic; idf characterizes a given term within an entire collection of documents. It is a measure of how widely the term is distributed over the given collection, and hence of how likely the term is to occur within any given document by chance. The idf is defined as “ln (N/n)” where N is the number of documents in the collection and n is the number of documents that contain the given term. Hence, the fewer the documents containing the given term, the larger the idf. If every document in the collection contains the given term, the idf is zero. This expresses the commonsense intuition that a term that occurs in every document in a given collection is not likely to be useful for distinguishing relevant from non-relevant documents. Or what is equivalent, a term that occurs in every document in a collection is not likely to be useful for distinguishing documents about one topic from documents about another topic. To cite a commonly-used example, in a collection of documents about computer science or software, the term “computer” is likely to occur in all or most of the documents, so it won’t be very good at discriminating documents relevant to a given query from documents that are non-relevant to the given query. (But the same
term might be very good at discriminating a document about computer science from documents that are not about computer science in another collection where computer science documents are rare.)

Computing the weight of a given term in a given document as $tf*idf$ says that the best descriptors of a given document will be terms that occur a good deal in the given document and very little in other documents. Similarly, a term that occurs a moderate number of times in a moderate proportion of the documents in the given collection will also be a good descriptor. Hence, the terms that are the best document descriptors in a given collection will be terms that occur with moderate frequency in that collection. The lowest weights will be assigned to terms that occur very infrequently in any document (low-frequency documents), and terms that occur in most or all of the documents (high frequency documents).

### 2.3.2 Normalization of Term Vectors

To allow for variation in document size, the weight is usually “normalized”. Two kinds of normalization are often applied. [Lee, SIGIR ‘95] The first is normalization of the term frequency, “$tf$”. The $tf$ is divided by the “maximum term frequency,” $tf_{max}$. The “maximum term frequency” is the frequency of the term that occurs most frequently in the given document. So the effect of normalizing term frequency is to generate a factor that varies between zero and one. This kind of normalization has been called “maximum normalization” for obvious reasons. A variation is the formula $0.5 + (0.5*\frac{tf}{tf_{max}})$ which causes the normalized $tf$ to vary between 0.5 and 1. In this form, the normalization has been called “augmented normalized term frequency”. The purpose and effect of term frequency normalization (in either form) is that the weight (the “importance”) of a term in a given document should depend on its frequency of occurrence relative to other terms in the same document, not its absolute frequency of occurrence. Weighting a term by absolute frequency would obviously tend to favor longer documents over shorter documents.

However, there is a potential flaw in “maximum normalization.” The normalization factor for a given document depends only on the frequency of the most frequent term(s) in the document. Consider a document $D_1$ in which most of the terms occur with frequencies in proportion to their importance in discriminating the document’s primary
topic. Now suppose that one term has a disproportionately high frequency, e.g., important terms \( t_1, t_2, \) and \( t_3 \) each occur twice in \( D_1 \) but for some stylistic reason equally important term \( t_4 \) occurs six times, the maximum for any term in \( D_1 \). Then the frequency of \( t_4 \) will drag down the weights of terms \( t_1, t_2, \) and \( t_3 \) by a factor of three in \( D_1 \) relative to their weights in some other similar document \( D_2 \) in which \( t_1, t_2, t_3, \) and \( t_4 \) have equal frequencies. (The same problem arises with the “augmented normalized term frequency” but to a less extreme degree since the high frequency term will have a weight of one as with maximum normalization but it cannot drag the weights of the other terms below 0.5.)

A commonly-used alternative to normalizing the term frequency is to take its natural log plus a constant, e.g., \( \log (tf) + 1 \). This technique, called “logarithmic term frequency,” doesn’t explicitly take document length or maximum term frequency into account but it does “reduce the importance of raw term frequency in those collections with widely varying document length.” It also reduces the effect of a term with an unusually high term frequency within a given document. In general, it reduces the effect of all variation in term frequency, since for any two term frequencies, \( tf_1 \) and \( tf_2 > 0 \) such that \( tf_2 > tf_1 \):

\[
\frac{\log tf_2 + 1}{\log tf_1 + 1} < \frac{tf_2}{tf_1}
\]

The second kind of normalization is by vector length. After all of the \( tf*idf \) term weights for a given document, i.e., all the components of the document vector, have been calculated, every component of the vector is divided by the Euclidean length of the vector. The Euclidean length of the vector is the square root of the sum of the squares of all its components. Dividing each component by the Euclidean length of the vector is called “cosine” normalization because the normalized vector has unit length and its projection on any axis in document space is the cosine of the angle between the vector and the given axis.

Augmented maximum (term frequency) normalization and cosine normalization can be used separately or together.
Cosine normalization reduces the problem (described above) of vector component weights for a given document being distorted by a single term with unusually high frequency. (But see the discussion below of pivoted unique normalization which further addresses the problem.) The normalization factor (vector length) is a function of all the vector components so the effect of a single term with a disproportionately high frequency is diluted by the weights of all the other terms. Furthermore, the normalization factor is a function of each \( \text{tf*idf} \) term weight, not just the \( \text{tf} \) factor of that weight. So, the weight of a high frequency term may also be lessened by its idf factor.

However, as Lee has pointed out, situations exist in which maximum normalization may actually do better than cosine normalization. Consider a case where document \( D_1 \) deals with topic \( T_A \) and contains a set of terms relevant to \( T_A \). Now consider document \( D_2 \) which deals with \( T_A \) and also deals with several other topics \( T_B, T_C, \) etc. Suppose that \( D_2 \) contains all the terms that \( D_1 \) contains, i.e., terms relevant to \( T_A \), but also contains many other terms relevant to \( T_B, T_C, \) etc. Since cosine normalization of a given document takes into account the weights of all its terms, the effect is that the weights of the terms relevant to \( T_A \) will be dragged down in \( D_2 \) (relative to the weights of the same terms in \( D_1 \)) by the weights of the terms relevant to \( T_B, T_C, \) etc. As a result, a user trying to retrieve documents relevant to \( T_A \) will be much more likely to retrieve \( D_1 \) than \( D_2 \) even if they both cover \( T_A \) to the same extent. Maximum normalization will do better in this case provided that the maximum frequency term relevant to \( T_A \) in \( D_2 \) is about as frequent as the maximum frequency term in \( D_2 \) relevant to any of the other topics. In that case, none of the other topics will drag down the weights of \( T_A \)'s terms in \( D_2 \). Lee concludes that in some cases, better precision and recall can be achieved by using each normalization scheme for filtering separately and then merging the results of the two filtering runs. (Merging filtering runs is discussed further below.)

Cosine normalization, as noted above by Lee, tends to favor short documents over long ones, especially in the case where the short document is about a single topic relevant to a given query, and the longer document is about multiple topics of which only one is relevant to the given query. Singhal et al. [SIGIR ‘96] have investigated this problem, and produced a new weighting scheme to correct the problem. They studied fifty queries applied to a large document collection (741,856 documents); queries and documents
were taken from the TREC 3 competition. Their study compared probability of filtering to probability of relevance as functions of document length. The study confirmed the expectation that short documents were more likely to be retrieved than their probability of relevance warranted, while longer documents were less likely to be retrieved than their probability of relevance warranted. This pattern was found to apply to query sets that retrieved relevant documents from six diverse sub-collections of the TREC collection. A natural consequence is that for any collection and query set to which the pattern applies, there will be a “crossover” document length for which the two probability curves intersect, i.e., a document length for which the probability of relevance equals the probability of filtering.

These observations led Singhal et al. to develop a “correction factor”, a function of document length that maps a conventional “old” document length normalization function, e.g., cosine normalization, into a “new” document normalization function. The correction factor rotates the old normalization function clockwise around the crossover point so that normalization values corresponding to document lengths below the crossover point are greater than before (so that the probability of filtering for these documents is decreased), and normalization values corresponding to document lengths above the crossover point are less than before (so that the probability of filtering for these documents is increased). (Remember that term weights for a given document are divided by the normalization factor.) The crossover point is called the “pivot”. Hence, the new normalization function is called “pivoted normalization.”

Note that since the pivoted normalization method described below is based on correcting the document normalization so that the distribution of probability of filtering coincides more closely with the probability of relevance (as a function of document length), this weighting method could legitimately be called “probabilistic”. However, it differs from the probabilistic methods discussed below in section 7, because the probability distributions have been determined experimentally, by observing actual TREC collections, rather than being derived from a theoretical model.

The pivoted normalization is easily derived. Before the normalization is corrected, i.e., pivoted, the relation between new normalization and old normalization is:

\[ \text{new normalization} = \text{old normalization} \]
This is a straight line with slope one through the origin of a graph, with a new normalization vertical axis, and an old normalization horizontal axis. This line is rotated clockwise around the pivot, i.e., around the normalization value corresponding to the crossover document length. Call this value “pivot.” After the rotation, the form of the new line (by elementary analytic geometry) is:

\[ \text{new normalization} = \text{slope} \times \text{old normalization} + K \]

where the slope of the new line is less than one and \( K \) is a constant. (Note that although the old normalization function, e.g., cosine normalization, is not a linear function of term weights, the new normalization is a linear function of the old normalization.) \( K \) is evaluated by recognizing that since the line was rotated around the pivot point, new normalization equals old normalization at the pivot point. Hence, substitute pivot for both new normalization and old normalization in the above linear equation, solve for \( K \) and then substitute this value of \( K \) back into the equation. The result (with new normalization now called pivoted normalization) is:

\[ \text{pivoted normalization} = \text{slope} \times \text{old normalization} + (1.0 - \text{slope}) \times \text{pivot} \]

where slope and pivot are constant parameters for a given collection and query set. Since the ranking of documents in a given collection for a given query set is not affected if the normalization factor for every document is multiplied (or divided) by the same constant, these two parameters can be reduced to one by dividing the above normalization function by the constant \((1.0 - \text{slope}) \times \text{pivot}\). Singhal et al. found that the optimum value of this parameter was surprisingly close to constant across a variety of TREC sub-collections. Hence, an optimum parameter value learned from training experiments on one collection could be used to compute normalization factors for other collections.

Singhal et al. also examined closely the role of term frequencies and term frequency normalization in term weighting schemes. First, they found (by studying the above experiments) that though, as noted above, cosine normalization favors short documents over long ones, it also favors extremely long documents. This phenomenon is magnified by pivoted normalization. Further, they noted that term frequency is not an important factor in either cosine normalization or document filtering. This is because (1) the majority of terms in a document only occur once, and (2) \( \log(tf) + 1 \) is commonly used in place of raw term frequency, which has a “dampening effect” for \( tf > 1 \). Hence, the
cosine normalization factor for a given document will be approximately equal to the square root of the number of unique terms in the given document, i.e., it increases less than linearly with number of unique terms. But document filtering is generally governed by the number of term matches between document and query, and hence making the usual simplifying assumption that occurrence of a given term in a given document is independent of the occurrence of any other term, the probability of a match between query and document varies linearly with the number of unique terms. The purpose of the document normalization is to adjust the term weights for each document so that the probability of retrieving a long document with a given query is the same as the probability of retrieving a short document. The conclusion is that cosine normalization reduces term weights by too little for very large documents. Singhal et al. propose to remedy this situation by replacing the cosine normalization value, i.e., the old normalization, by \( \text{# of unique terms} \), in the pivoted normalization function.

Singhal et al. argue further that maximum normalization, i.e., \( \frac{tf}{tf_{max}} \), is not the optimum method of normalizing term frequency because what matters is the frequency of the term relative to the frequencies of all the other descriptor terms, not just relative to the frequency of the most frequently occurring term. Hence, they propose using the function:

\[
\frac{1 + \log(tf)}{1 + \log(\text{average } tf)}
\]

for normalized term frequency. This function has the property that its value is one for a term whose frequency is average for the given document, greater than one for terms whose frequencies are greater than average, and less than one for terms whose frequency is less than average.

Hence, Singhal et al. propose weighting each term in a document by the above term frequency normalization function divided by the pivoted normalization, i.e.,

\[
\frac{1 + \log(tf)}{1 + \log(\text{average } tf)} \times \left( \frac{slop \times \text{# of unique terms} + (1.0 - slope) \times pivot}{slop \times \text{# of unique terms} + (1.0 - slope) \times pivot} \right)
\]

Note that the \( idf \) is absent from this weighting function. This is because, for reasons explained in the next section, the \( idf \) is normally used as a factor in the weights of query terms rather than document terms. That is, if a given term occurs in a query and also in some documents in the collection being queried, the \( idf \) will be used as a factor in the
weight of that term in the query vector rather than in the corresponding document vectors.

Singhal et al. tested this improved term weighting function against a set of TREC sub-collections and found that for optimum parameter values, it performed substantially better than the more familiar product of $tf/tf_{max}$ and $1/cosine$ normalization.

It should be noted that this improved weighting scheme compensates for both of the problems noted by Lee. The effect of a term with a disproportionately high frequency in a given document is greatly reduced by the new term frequency normalization function, partly because the frequency of the given term is divided by the average term frequency rather than the maximum term frequency, and partly because both the given term frequency and the average term frequency are replaced by their logs. The advantage of a short document dealing entirely with a topic relevant to a given query, over a longer document dealing with the relevant topic and several non-relevant topics, is compensated by pivoted normalization which reduces the probability of filtering of short documents and increases the probability of filtering of long documents.

All of the normalization schemes discussed above (and in the following section) are based on one underlying assumption: that document relevance is independent of document length. Relevance is assumed to be wholly about how much a given document is about a given topic. Variation of document length is viewed as a complication in computing document relevance. Hence, all of the normalization schemes are aimed at factoring out the effects of document length.

If document $D_2$ is longer than document $D_1$, relevance computation is assumed to be distorted in one of two ways. If $D_2$ is largely or entirely about the same topic $T_i$ as $D_1$, then relevance is distorted by the fact that terms characteristic of the given topic will tend to occur with greater frequency in $D_2$. If $D_2$ is about a number of different topics, $T_i$, $T_j$, $T_k$, etc., and only a small part of $D_2$ is about the same topic $T_i$ as $D_1$, then the material about the other topics dilutes the effect of the relevant material, making $D_2$ seem less relevant to $T_i$ than $D_1$, even though both documents may contain the same information about $T_i$. Normalization is largely aimed at overcoming these two kinds of distortion.

Completely ignored is a third possibility: that the user actually prefers either short or long documents about $T_i$. If the user prefers the longer document, $D_2$, e.g., she needs all
the details it provides, then $D_2$ is more relevant to $T_i$ relative the users needs. Or to use a term that may be more appropriate, $D_2$ may be more pertinent or more useful to the user in meeting her present information need, as expressed by $T_i$. Of course, $D_1$ may be more useful; perhaps the user needs a concise summary of the main facts or ideas about $T_i$, and has neither time nor need for a more detailed exposition. In either case, document size is an important parameter in computing the documents relevance for purposes of selection and ranking in the filtering set returned to the user. This indicates that either the document and topic vectors should not be normalized, or that the document size should enter explicitly into the document topic similarity computation. This issue is discussed further in section 2.3.4, which discusses document-topic similarity functions.

2.3.3 Classification of Term Vector Weighting Schemes

Since the various alternatives discussed above for computing and normalizing term weights can be (and have been) used in a variety of combinations, a conventional code scheme (associated with a popular IF research engine called the SMART system) has been defined and widely adopted to classify the alternatives. [Salton, IP&M, 1988] [Lee, SIGIR ‘95] See Table 1.

The weight of a given term is specified as the product of a term frequency factor, a document frequency factor, and a document length normalization factor. For each of these three factors, two or more alternatives are available. Each alternative for each factor is given a code. See table 1. The codes for the term frequency factor are: “b” (term frequency is ignored; the term frequency factor is one if the term is present in the given document, zero otherwise), “n” (use the raw term frequency, the number of times the term occurs in the given document), “a” (use the “augmented normalized term frequency” as defined in the previous section), “l” (use the “logarithmic term frequency” as defined in the previous section), and “L” (use “average term frequency based normalization” as defined in the previous section). (A code for the pure “maximum normalization” does not appear to have been defined.) The codes for the document frequency factor are; “n” (use 1.0, document frequency factor is ignored), “t” (use “idf” as the document frequency factor). The codes for the normalization factor are: “n” (use 1.0; no document length normalization is used), “c” (use cosine normalization, i.e., $1/(Euclidean\ vector\ length)$), and “u” (use “pivoted unique normalization” as discussed in the previous section). A weighting scheme is constructed in “Chinese menu” form: one
from column A (term frequency factor), one from column B (document frequency factor), and one from column C (document normalization factor). For example, “lnc” means, “Compute the weight of each term in a given document as the product of the logarithmic term frequency (l) of the given term, 1.0 (ignore the idf of the term), and the cosine normalization factor (c) of the document vector for which the term’s weight is being computed.” (Multiplying by the cosine normalization factor is equivalent to dividing by the Euclidean vector length as defined in above.) As a further refinement, it is common to use a different weighting scheme for the query than for the documents in the collection being queried. Therefore, the complete specification of the weighting scheme involves two triples, e.g., lnchl describes a scheme where the document vectors are weighted as above, and the query vectors are weighted the same except that each query term weight is also multiplied by the idf of the given term in the collection to which the query is being applied. (Note that the query is weighted as a “document” so that the term frequency factor

Table 1: Components of schemes for weighting given term in given document

<table>
<thead>
<tr>
<th>Code</th>
<th>Formula for Component</th>
<th>Description of Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1.0</td>
<td>Term frequency = 1 if term is in given document, = 0 if term is not in given document.</td>
</tr>
<tr>
<td>n</td>
<td>tf</td>
<td>“Raw” term frequency, i.e., number of occurrences of term in given document.</td>
</tr>
<tr>
<td>a</td>
<td>(0.5 + 0.5 \frac{tf}{max,tf})</td>
<td>“Augmented” term frequency. First, term frequency of given term is normalized by frequency of most frequent term in document (“maximum” normalization) to allow for importance of term relative to other terms in document. Then, it is further normalized (“augmented”) so resulting value is in range from 0.5 to 1.0.</td>
</tr>
<tr>
<td>l</td>
<td>(ln,tf + 1.0)</td>
<td>Logarithmic term frequency. This reduces importance of raw term frequency, e.g., if (t_2) has twice the frequency of (t_1) in given document, the ratio of the logs will be much smaller.</td>
</tr>
<tr>
<td>L</td>
<td>(1 + \log \frac{tf}{1 + \log (average,tf)})</td>
<td>Average term frequency based normalization. See discussion in previous section.</td>
</tr>
</tbody>
</table>
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

### Document Frequency (Number containing Term) within Collection

<table>
<thead>
<tr>
<th>$n$</th>
<th>1.0</th>
<th>Number of documents containing given term is ignored. Original term frequency is not modified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$\ln \frac{N}{n}$</td>
<td>Original term frequency is multiplied by inverse document frequency (idf) where $N$ is the total number of documents in the collection, and $n$ is the number of documents containing the given term. Hence, term that occurs in many documents counts for less than term that occurs in few (or one).</td>
</tr>
</tbody>
</table>

### Document Length Normalization Component

<table>
<thead>
<tr>
<th>$n$</th>
<th>1.0</th>
<th>Variation in document length is ignored.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>$\frac{1}{\sqrt{\sum w_i^2}}$</td>
<td>Weight of given term in given document is normalized by the length of the document’s term vector, so that long documents are not favored over short documents.</td>
</tr>
<tr>
<td>$u$</td>
<td>$\frac{1}{(slope \cdot # \text{ of unique terms}) + (1 - slope) \cdot pivot}$</td>
<td>Pivoted normalization. See previous section.</td>
</tr>
</tbody>
</table>

measures term frequency within the query, and the document length factor normalizes for query length. Only the idf factor in the weight of a query term is a measure of the distribution in the collection being queried, not in the query itself.) This scheme has exhibited high filtering effectiveness for the Text Filtering Conference (TREC) query sets and collections. [Lee, SIGIR ‘95] However, Lnultc weighting exhibited even better effectiveness against TREC3 and TREC 4 query sets and collections. [Singhal et al., SIGIR ‘96] [Buckley et al., TREC 4]

The weighting scheme classification described above is open-ended. Indeed, the SMART team, originators of this classification, have only recently added the $L$ option to the term frequency factor, and the $u$ option to the document length normalization factor.

Note that although the idf of a given term is a statistic that characterizes that term relative to a given collection of documents, not relative to a query, it is common to use the idf to weight the occurrence of the given term in queries being applied to the collection, not to weight its occurrence in the document vectors that describe the collection itself. The Inc-Itc and Lnul-Ltu weighting schemes are examples. There are
simple reasons for this. First, it is more efficient for purposes of collection maintenance. Whenever new documents are added to the collection (or old documents are removed), the idf must be recomputed for each descriptor term in the affected documents. It would be inefficient to recompute the weight of such a term in every document in which it occurs. Moreover, it is unnecessary for the purposes of a query/document similarity calculation, since the document ranking produced for a given query will be exactly the same whether the idf’s enter the computation as factors in the query term weights, or factors in the document term weights, or both.

In a weighting scheme like $tf\cdot idf$, the normalized term frequency of a given term in a given document is multiplied by its idf so that “good” descriptor terms (which characterize only a relatively small number of documents in a given collection) are weighted more heavily than “bad” descriptor terms (which are so common that they occur in a great many documents in the given collection, and hence are of little value in discriminating between relevant and non-relevant documents).

An alternative approach is to subtract from the normalized term frequency of the given term the “average” normalized frequency of the term averaged over all the documents in the given collection. Here, “average” may be “mean”, “median”, “or some other measure of commonality”. [Damashek, Science, 1995] (This is equivalent to subtracting from each document vector a “centroid” vector, i.e., a vector that is the average of all the document vectors in the collection.) Note that a term that occurs in a large proportion of the documents in the given collection will have a larger average term frequency than a term that occurs in only a few documents. Hence, the effect of subtracting the average is to reduce the weight of commonly used terms by more than the weight of rarer terms. The centroid is a measure of commonality, of terms that are too widely used to be good document descriptors.

### 2.3.4 Computation of Similarity between Document and Query

Once vectors have been computed for the query and for each document in the given collection, e.g., using a weighting scheme like those described above, the next step is to compute a numeric “similarity” between the query and each document. The documents can then be ranked according to how similar they are to the query, i.e., the highest ranking document is the document most similar to the query, etc. While it would be too much to hope that ranking by similarity in document vector space would correspond
exactly with human judgment of degree of relevance to the given query, the hope (borne out to some degree in practice) is that the documents with high similarity will include a high proportion of the relevant documents, and that the documents with very low similarity will include very few relevant documents. (Of course, this assumes that the given collection contains some relevant documents, an assumption that holds in TREC experiments but which can’t be guaranteed in all practical situations.) Ranking of course, allows the human user to restrict his attention to a set of documents of manageable size, e.g., the top 20 documents, etc.

The usual similarity measure employed in document vector space is the “inner product” between the query vector and a given document vector. [Salton, 1983] [Salton, 1989]

The inner product between a query vector and a document vector is computed by multiplying the query vector component (i.e., weight), \( QT_i \) for each term \( i \), by the corresponding document vector component weight, \( DT_i \) for the same term \( i \), and summing these products over all \( i \). Hence the inner product is given by:

\[
\sum_{i=1}^{N} QT_i \cdot DT_i
\]

where \( N \) is the number of descriptor terms common to the query and the given document.

If both vectors have been cosine normalized, then this inner product represents the cosine of the angle between the two vectors; hence this similarity measure is often called “cosine similarity.” The maximum similarity is one, corresponding to the query and document vectors being identical (angle between them zero). The minimum similarity is zero corresponding to the two vectors having no terms in common (angle between them is 90 degrees).

One problem with cosine similarity, noted by both Salton and Lee and discussed above, is that it tends to produce relatively low similarity values for long documents, especially (as Lee points out) when the document is long because it deals with multiple topics. But Lee’s solution (mentioned above and discussed in more detail in section 2.3.2) is not to use a more complicated similarity measure in place of cosine similarity, but rather to merge the result of a filtering using cosine similarity with the result of a retrieval using term frequency normalization, e.g., maximum normalization. In other words, Lee supplements cosine similarity rather than replaces it, thereby getting the advantages of two relatively simple similarity measures. And the solution of Singhal

32
et al., discussed above in section 2.3.2, is to develop improved normalization factors for term weighting, factors that do a better job of normalizing for document length and term frequency during a single filtering run, thereby eliminating the need for fusion of separate runs.

In an earlier approach from the same research group, Salton and Buckley [TREC 2, 1994] dealt with the problem of long documents by combining the usual cosine similarity of query and document (“global” similarity) with similarity of the query to parts of the document (“local” similarity). The parts they tried included sentences and paragraphs. In other words, if two documents $D_1$ and $D_2$ have comparable similarity to a given query, but $D_1$ also contains a sentence or paragraph that is particularly similar to the query, then $D_1$ will be given a higher similarity value than $D_2$. They have also tried combining multiple local similarity measures, e.g., sentence and paragraph similarity, with the global similarity. However, the Singhal et al. enhanced term weighting ($Lnu$) method, by improving document length normalization, and term frequency normalization, reduces the importance of such local similarity measures. Buckley et al. [TREC 4] report that using $Lnu$ weighting reduces the improvement gained by their local/global similarity measures from 15% to 3%. They continue to experiment with sophisticated similarity measures that take into account the actual location of every descriptor term. However, their motivation is now less to correct for document length bias, and more to create a similarity measure to which non-statistical, e.g., linguistic and certainty, factors at the term level can be added.

Another approach to the problem of large, or multi-topic, documents is to break each large document into sections, commonly called “passages,” and treat each passage as a “document.” [Callan, SIGIR ‘94] In other words, the system computes the similarity between each passage and the user’s query. This enables the system to determine the “best” passage, i.e., the passage in a given document most similar (and hence hopefully most relevant) to the query. This passage then comes to represent the document in any further computations or retrievals involving the same query or any similar information need. Passages have been calculated in a number of ways: sections as determined by document markup [Wilkinson, SIGIR ‘94], passages as delimited by the author, e.g., sections, paragraphs, sentences, etc. [Salton et al., SIGIR ‘93], clusters of paragraphs, fixed size passages, etc. A document may be partitioned into fixed size disjoint passages.
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

Fixed length passages may be a fixed number of terms, e.g., 50 words to 600 words each [Kaszkiel et al., SIGIR ‘97], with each passage beginning immediately after the preceding passage. Zobel et al. [IP&M, 1995] employ passages of 1000 to 2000 words, which they call pages; the passages are generated by accumulating paragraphs until the desired length is reached, so that the passages are not exactly fixed length, but always end on a paragraph boundary. Alternatively, fixed-size passages may be overlapped, e.g., if the passages are each $p$ terms long, each passage may start $p/2$ terms beyond the start of the preceding passage. [Callan, SIGIR ‘94] These fixed size, overlapping passages have been called windows, because they can be viewed as obtained by sliding a window $p$ terms long over the text. “[E]xperiments have shown that fixed size passages are at least as effective — and marginally more efficient than their varying counterparts.” [Allan, SIGIR ‘95]

The superiority of fixed length, overlapping and non-overlapping, passages is also supported by the experiments of Kaszkiel et al. [SIGIR ‘97] discussed below.) These passage approaches differs from the Salton and Buckley “global/local approach” in that only local similarities between a query and the passages of a document are computed. Documents are ranked according to the best local similarity in the given document for the given query, i.e., the passage in a each document that is most similar to the given query. Presumably, the benefit of this approach too is reduced by incorporation of an improved document normalization scheme such as $Lnu$. (However, Kaszkiel et al. found that the pivoted document length normalization improved passage length filtering for all those cases where the passages varied substantially in length.)

Hearst et al. [SIGIR ‘93] approach global/local similarity in a novel way. First, they attempt to break a given document into motivated segments, i.e., variable length segments such that the boundary between one segment and the next is the boundary between one subtopic and the next. These segments, called “tiles,” may be multi-paragraph units.

The “tiling” method begins by partitioning the document into “blocks,” such that each block is $k$ sentences long. As a heuristic, Hearst chooses $k$ to be the average paragraph length in sentences. Note that this means that the length of a block in sentences may vary somewhat from one document to the next. Moreover, since the block size is a fixed
number of sentences, blocks will vary somewhat in size (measured in terms) even within a given document, although they will be approximately equal.

The blocks are combined into topic-based segments, on the assumption that two consecutive blocks are likely to be about the same topic if they are statistically similar, and likely to be about different topics if they are not similar. Similarity is calculated for every pair of consecutive blocks in a given document. The similarity measure employed is the standard cosine similarity, with terms weighted according to the standard $tf*idf$ formula. The novelty in employing this formula is that the blocks are treated as the “documents,” while the document is treated as the document “collection.” Hence, in computing the weight of term $t_i$ in block $B_j$ of document $D_k$, $tf_i$ is the unnormalized frequency of term $t_i$ in $B_j$, while $idf_i$ is the $idf$ of $t_i$ in the collection of blocks comprising $D_k$, i.e., it will be zero if it is present in every block of $D_k$, much higher if it is only present in one or two blocks of $D_k$. The effect is that two consecutive blocks $B_j$ and $B_{j+1}$ will be very similar if they share a number of terms, the terms are relatively frequent in $B_j$ and $B_{j+1}$, and are found in few other blocks of $D_k$. By contrast, terms that are shared by $B_j$ and $B_{j+1}$ but are spread fairly uniformly throughout the entire document, will contribute much less to the inter-block similarity.

These consecutive block similarities are then graphed against block position in the document. The result is a function that peaks where consecutive blocks are very similar, and drops where they are very dissimilar. After this function is smoothed to eliminate small local fluctuations, the result is a function with peaks and valleys. The low points of the valleys are then taken to be the boundaries of the tiles, segments that are each assumed to be about a single local subtopic.

These tiles can be used in a number of ways. Hearst et al. propose a two stage query, where the user can request documents about topic $T_1$, matching $T_1$ against an entire document, and about subtopic $T_2$, matching $T_2$ against each block in document $D_k$ that satisfies $T_1$. Note that this is different than combining a local and global similarity into a single similarity measure as Salton and Buckley do. In the proposed Hearst approach, a document must separately satisfy both the global similarity to $T_1$ and the local similarity to $T_2$. Neither topic will unduly dilute
the calculation of similarity for the other topic. As a further enhancement, terms associated with the global or “background” topic $T_1$ could be eliminated when searching tiles for $T_2$.

Tiles have also been used by Hearst et al. in a single stage query mode. Here, the idea is that each document in a collection is segmented into tiles, and each tile is then term indexed as a separate “virtual” document. The index for a given tile also identifies the actual document from which it came. Given a query, $Q$, the best (most similar) $N$ tiles ($N = 200$ in the reported experiment) are retrieved, and grouped by document of origin. Then, the similarity scores of the tiles in each group, i.e., originating in the same document, are summed. The actual documents are then ranked according to these sums. In the reported experiment using TREC data, filtering using these tile similarity sums produced substantially better precision than querying directly on indexes of the full text (actual) documents. Apparently, the tile method favored those portions of a document that were most relevant to a given query.

Kretser et al. [SIGIR ’99] carry the concept of locality-based similarity even farther than Hearst does with her tiles. They view a document collection as a sequence of words. They calculate a “score” for each word position in the sequence. Hence, the position of a word is its position relative to the start of the collection, not relative to the document in which it occurs. Given a query $Q$ defined as a set of terms (words) $t$, the Kretser system computes a score $C_Q(x)$ for every position $x$ in the collection. The score for a given word position $x$ depends on whether a query term $t$ occurs at that position, and on whether query terms occur within a certain distance of that position. The farther a query term occurs from the word position $x$ being evaluated, the less its influence, the smaller the “contribution” it makes to $C_Q(x)$. The score $C_Q(x)$ at word position $x$ is the sum of the contributions $c_t(x,l)$ where $t$ is any query term, $l$ is any word position in the document collection at which $t$ occurs, and $c_t(x,l)$ is the contribution that query term $t$ at word position $l$ makes to the score at word position $x$.

Kretser et al. tried out four different shapes for the contribution function $c_t(x,l)$: triangle, cosine, circle, and arc. However, in all four cases, $c_t(x,l)$ was a function of (1) $d = |x-l|$, the maximum distance from $x$ at which a query term can make a contribution to the score at $x$, (2) $h$, the “height” or peak contribution, which is the contribution made by a query term
CHAPTER 2: APPROACHES TO INFORMATION FILTERING

$t$ at position $x$, and (3) $s_r$, the “spread” of the contribution function for query term $t$, which determines the maximum distance, $d_{\text{max}}$, at which $t$ can contribute to the score at word position $x$. Note that the height and spread are both functions of $t$ itself. A scarce term $t$ will have a greater spread; in other words, an occurrence of $t$ is able to influence the score at $x$ at a greater distance from $x$ than an occurrence of a common term. Similarly, a scarce term $t$ contributes more proportionately than a common term to the score at $x$ if it occurs at $x$, or if it occurs at any distance from $x$ within its spread. It is noteworthy that both the spread and the height are collection-based, not document-based functions. That is, “scarcity” means scarcity within the collection, not scarcity within any given document. For example, the “spread” is defined as $n/f_t$ where $n$ is the number of unique words in the collection, and $f_t$ is the frequency of $t$, i.e., the number of times term $t$ occurs in the collection. However, document boundaries do play one role in the computation of the score at word position $x$. The spread is not allowed to cross a document boundary. Hence, the score at a word position near the beginning or end of a document cannot be influenced by a query term in an adjacent document.

The scheme described above generates a similarity score for every position $x$ in the document collection relative to a given query $Q$. The similarities must be computed at runtime, when the query $Q$ is supplied. However, as with conventional document-based systems, the collection is indexed in advance as an inverted list, to facilitate the runtime similarity calculation. But Kretser’s index must specify word positions for every term in the collection, not just document occurrences as for conventional IF systems. Moreover, since it is intended that the query engine will return a ranked list of documents, the document boundaries must also be stored in the index. Various strategies are used to compress all of this information efficiently. The compression ratio varies with the collection. For the four collections tested by Kretser et al., the ratio of index size to collection size varied from 18.9% to 23.1%. This compares with 2.6% to 6.8% for the corresponding document level indexes.

At runtime, given a query $Q$, the system computes the similarity score of every word position relative to $Q$. To rank documents, the system finds the highest scoring word position, and adds its score to an (initially zero) accumulator for the document in which it occurs. Then, it repeats the process for the next highest scoring word position. This 2nd word position may be in the same document, in which case its score is added to the
same accumulator. Or, it may occur in another document, in which case, it is added to a new accumulator. The process continues until $r$ non-zero accumulators have been generated, where $r$ is the number of documents to be returned to the user. The documents are presented to the user in the order of the scores in their accumulators.

Kaszkiel et al. [SIGIR ‘97] performed experiments to compare various approaches to passages, including Hearst’s tiles, Zobel’s pages, Callan’s fixed-length overlapping windows, non-overlapping fixed length passages, Wilkinson’s sections, and paragraphs. Twelve different fixed length passage sizes from 50 words to 600 words were tried. The results indicated that fixed length passages, both overlapping and non-overlapping, of 150 words or more were “simple; highly effective; robust” for document filtering. They scored as well or better than both whole document filtering, and other passage methods, over the full range of experiments, e.g., for different data sets, and different document normalization schemes. Several limitations of these experiments should be noted, however. Kaszkiel et al. note a couple of limitations themselves: First, their results do not rule out the possibility of further improvement through “combination of passage level and document-level evidence” Second, they did not exploit (as did some of the passage methods they cite, e.g., tiling) the possibility of achieving better document ranking by combining multiple passage similarities for each document. There are other limitations that they do not mention. Their study focused on using passage similarities to rank the whole documents in which the passages occur. They do not study passages as units of filtering themselves; indeed, it would be difficult for them to do so, since they employ TREC data for training and testing; TREC data only provides relevance judgments for whole documents, not for passages (let alone the diverse kinds of passages compared here). Similarly, they do not study the interactive uses of passages, e.g., the presentation high-ranking passages to the use, as a basis for selecting either documents, or other similar passages.

Most of the vector space similarity formulas discussed above assume that the vectors have been normalized, with the intended effect that document length is factored out. In other words, the intention is that a short document about topic $T_i$, a long document containing a short passage about $T_i$, and a long document entirely about $T_i$ will yield the same similarity to the topic statement $T_i$ itself. But as was noted in section 2.3.2, often the user has a preference for either longer or shorter documents about the topic of
interest. This problem has not usually been addressed in IF research, but a number of possibilities are available.

One alternative, mentioned back in 2.3.2, is not to normalize the vectors at all. The effect of taking the inner product of unnormalized vectors (no term frequency normalizations, no vector length normalization) is to generate similarity values that take into account both relevance and size. A short document \( D_1 \) relevant to \( T_i \) will receive a larger similarity value than a document \( D_0 \) totally non-relevant, a long document \( D_3 \) containing a short passage relevant to \( T_i \) will receive a similarity value roughly equal to that received by \( D_2 \), and a long document \( D_4 \) largely or entirely about \( T_i \) will receive the largest similarity value of all. This works well if the user prefers longer, more detailed documents like \( D_4 \). But what if the user prefers shorter documents? Simple functions are available for inverting the similarity function just described so that the similarity values of the inverted function decrease as the similarity values of the original function increase. But this has the perverse effect of giving the highest similarity values to documents that are non-relevant!

Another alternative is to make size a separate and distinct parameter. Size can be made a separate component of both the topic description vector and each document vector. Hence, a match on size category, e.g., small, medium, large, could increase the relevance by an amount that depended on the term weights assigned to that component. Or size could be taken out of the vector space altogether, e.g., the algorithm could test the user’s size preference first, and then perform a different vector inner product computation based on the user’s preference. The vectors would be unnormalized if the user preferred large documents, normalized (especially pivoted normalization) if the user wanted equal preference to be given to documents about the given topic independently of size. Cosine normalization or some other normalization scheme specifically designed to favor short relevant documents over long relevant documents would be applied to the vectors if the user preference were for “short and sweet.”

The inner product and its normalized form, cosine similarity, are not the only similarity functions employed to compare a document vector with a topic vector (although they are by far the most widely used). A variety of “distance” functions, and other term
matching functions are available. For example, a family of distance metrics [Korfhage, 1997] is given by:

$$L_p(D_1, D_2) = \left[ \sum_i |d_{1i} - d_{2i}|^p \right]^{1/p}$$

These metrics compute the distance in vector space between vectors $D_1$ and $D_2$ in terms of the components $d_{1i}$ of $D_1$, the components $d_{2i}$ of $D_2$, and a parameter $p$ that determines which chooses a specific metric from the family. If $p=1$, the metric is the city block distance, i.e., distance measured as number of city blocks from one street intersection (corner) to another in a city where the streets are laid out as a rectangular grid. If $p=2$, the metric is the familiar Euclidean distance, i.e., the straight line distance in the vector space. (This is the same metric that is used for computing the length of a vector for purposes of Euclidean normalization.) If $p=\infty$, the metric is the maximal direction distance. That is, as $p$ tends to infinity, the largest difference $|d_{1i} - d_{2i}|$ tends to dominate all the others, and the function reduces to the absolute value of this maximum difference. Since each vector component corresponds to one dimension, one direction, in vector space, each difference between a pair of corresponding components is the distance between the vectors in a given direction. The maximal direction distance metric is the distance along the dimension where the vectors are farthest apart.

Apart from such distance metrics, there are a host of similarity formulas that “normalize” by avoiding term frequencies altogether, i.e., functions that only count the number of terms that match and (sometimes) the number of terms that don’t match. One such popular function is Dice’s coefficient [van Rijsbergen, 1979]:

$$\text{Dice} = \frac{2w}{n_1 + n_2}$$

where $w$ is the number of terms common to vectors $D_1$ and $D_2$, $n_1$ is the number of non-zero terms in $D_1$, and $n_2$ is the number of non-zero terms in $D_2$. Note that the denominator here performs a kind of normalization, so that a short document $D_1$ will get a high score relative to a short topic description $D_2$ to which it is relevant. A long document $D_3$ relevant to $D_2$ will get a lower Dice score provided that the additional text in $D_3$ contains terms that are not in $D_1$ and also not in the topic description (greater $n_1$, same $w$). This could happen if $D_3$ contains long sections not relevant to $D_2$, It could also
happen if $D_3$ contains additional discussion of the topic described by $D_2$, but this additional discussion uses terms that were overlooked by the user who specified topic $D_2$. On the other hand, if $D_3$ and $D_1$ contain most of the same topic-relevant terms that $D_2$ contains, but $D_3$ just uses them more frequently and uses few additional terms that $D_1$ doesn’t use, then $D_3$ and $D_1$ will receive similar Dice scores despite their difference in length.

Another common similarity function is Jaccard’s coefficient [van Rijsbergen, 1979]:

$$\text{Jaccard}(D_1, D_2) = \frac{w}{N - z}$$

where $w$ (as before) is the number of terms common to vectors $D_1$ and $D_2$, $N$ is the total number of distinct terms (not term occurrences!) in the vector space (union of all document and topic vectors), and $z$ is the number of distinct terms (not term occurrences!) that are neither in $D_1$ nor in $D_2$. In other words, $N-z$ is the total number of distinct terms that occur in $D_1$ or $D_2$ or both. Note that the value of the Jaccard function is lower, the more distinct terms are either in $D_1$ but not $D_2$ or vice versa. It doesn’t matter whether the mismatch is caused by non-relevance, or difference in document length. On the other hand, it doesn’t matter how frequently a mismatching term (or a matching term) occurs in either $D_1$ or $D_2$.

The above schemes for computing query/document similarity assume the existence of a (relatively) static collection of documents to which each query formulated by a user is applied. At the other extreme, we have the “routing” case in which documents arrive in a constantly changing incoming stream, and each document must be “routed” to one of $N$ boxes corresponding to $N$ preselected topics. In sharp contrast to the collection filtering case, the “queries,” i.e., topics or information needs, are fixed while the supply of documents is very dynamic. How can the vector space approach (or any statistical approach) be applied to a situation in which there is no fixed document collection for which collection-wide statistics such as $idf$ can be computed? The usual answer is to provide a “training set” of (one hopes) “typical” documents for which statistics can be calculated. Obviously, the hope is that all the subsequent documents received by one’s system will have the same statistical properties as the training set. (The alternative is to
update the training set regularly, which of course requires retraining the system regularly too.)

### 2.3.5 Latent Semantic Indexing (LSI) — An Alternative Vector Scheme

In the traditional vector space approach to IF described above, a vector “space” is defined for a collection of documents such that each dimension of the space is a term occurring in the collection, and each document is specified as a vector with a coordinate for each term occurring in the given document. The value of each coordinate is a weight assigned to the corresponding term, a weight intended to be a measure of how important the given term is in characterizing the given document and distinguishing it from the other documents in the given collection. This approach is an effective first approximation to the statistical properties of the collection, but it is nevertheless an oversimplification. Its major limitation is that it assumes that the terms are independent, orthogonal dimensions of the document space. Adding a new term to the space, e.g., a term that was previously omitted because it wasn’t considered a good discriminator, has no effect whatever on the existing terms defining the space. (Adding a new document to the collection not only adds new terms to the space but also does affect the weights of the existing terms because it affects their idf’s. But this is a term-document relationship, not a term-term relationship.) Hence, relationships among the terms, e.g., the fact that certain terms are likely to co-occur in documents about a given topic because they all refer to aspects of that topic, are ignored. Similarly (and more subtly), the traditional term vector approach ignores the fact that term A and term B may occur in similar contexts in two distinct documents because they are synonyms.

The traditional vector space approach has another feature that can be a drawback in some applications: Since the number of terms that occur in a collection can be large (even after “noise” words have been deleted with a stop list, and variant forms of the same word have been eliminated by stemming), the traditional term-based document space has a large number of dimensions.

A new vector space approach called Latent Semantic Indexing (LSI) [Deerwester et al., JASIS, 1990] attempts to capture these term ‘term statistical relationships’. In LSI, the document space in which each dimension is an actual term occurring in the collection is replaced by (recalculated as) a much lower dimensional document space called k-space (or LSI space) in which each dimension is a derived concept, a “conceptual index,” called an LSI “factor” or “feature.” These LSI factors are truly independent statistically,
i.e., uncorrelated, in a way that terms are not. Hence, LSI factors are “information rich” [SIGIR ‘94, Hull] in the sense that they capture the term-term relationships that ordinary term-based document space does not. Documents are represented by LSI factor vectors in k-space just as they are represented by term vectors in traditional term-based document space. Vector similarity can be calculated in the same way in k-space as in traditional document space. However, documents and queries dealing with the same topic that would be far apart in traditional document space (e.g., because they use different but synonymous terms) may be close together in k-space.

As Bartell, et al. [SIGIR ‘92] explain (they speak of “keywords” rather than “terms”):

[I]ndividual keywords are not adequate discriminators of semantic content. Rather the indexing relationship between word and document is many-to-many: A number of concepts can be indexed by a single term [polysemy], and a number of terms can index a single concept [synonymy] … [Hence] some relevant documents are missed (they are not indexed by the keywords used in the query, but by synonyms) and some irrelevant documents are retrieved (they are indexed by unintended senses of the keywords in the query). LSI aim[s] at addressing these limitations. This technique maps each document from a vector space representation based on keyword frequency to a vector in a lower dimensional space. Terms are also mapped into vectors in the reduced space. The claim is that the similarity between vectors in the reduced space … may be a better filtering indicator than similarity measured in the original term space. This is primarily because, in the reduced space, two related documents may be represented similarly even though they do not share any keywords. This may occur, for example, if the keywords used in each of the documents co-occur frequently in other documents.

In other words, if document $D_1$ uses term $t_A$ and document $D_2$ uses equivalent term $t_B$, LSI will effectively recognize this equivalence statistically if $t_B$ and $t_A$ co-occur frequently in similar contexts in other documents. Berry et al. [SIAM Review, 1995] offer a good example:

Consider the words car, automobile, driver, and elephant. The terms car and automobile are synonyms, driver is a related concept and elephant is unrelated. In most [e.g., traditional term vector] filtering systems, the query automobiles is no more likely to retrieve documents about cars than documents about elephants, if the precise term automobile was not used in the documents. It would be preferable if a query about automobiles
also retrieved articles about *cars* and even articles about *drivers* to a lesser extent. The derived [LSI] $k$-dimensional feature space can represent these useful term relationships. Roughly speaking, the words *car* and *automobile* will occur with many of the same words [i.e., in the same “context”] (e.g., *motor, model, vehicle, chassis, carmakers, sedan engine, etc.*), and they will have similar representations in $k$-space. The contexts for *driver* will overlap to a lesser extent, and those for *elephant* will be quite dissimilar.

(To be fair about it, a good query submitted to a traditional term-based system would use more terms than “automobile”, e.g., perhaps some of the other contextual words mentioned in the passage quoted above.)

In other words, the traditional term-based vector space model assumes term independence. “Since there are strong associations between terms in language, this assumption is never satisfied [though it may be] a reasonable first order approximation.” [SIGIR ’94, Hull] LSI attempts to capture some of these semantic term dependencies using a purely statistical and automatic method, i.e., without syntactic or semantic natural language analysis and without manual human intervention.

LSI accomplishes this by using a method of matrix decomposition called Singular Value Decomposition (SVD). LSI takes the original document-by-term matrix describing the traditional term-based document space as input. It produces as output three new matrices: $T$, $S$, and $D$ such that their product $T*S*D$ captures this same statistical information in a new coordinate space, $k$-space, where each of the $k$ dimensions represents one of the derived LSI “features” or “concepts” or “factors.” “[T]hese factors may be thought of as artificial concepts; they represent extracted common meaning components of many different words and documents.” [JASIS, 1990, Deerwester *et al.]* $D$ is a “document” matrix. Each column of $D$ is one of the $k$ derived concepts. Each row of $D$ is the vector for a given document, specified in terms of the $k$ concepts. The matrix element for the $j$th concept in the $i$th document represents the strength of association of concept $j$ with document $i$. Hence, $D$ specifies documents in $k$-space. Similarly, $T$ is a term matrix. Each column as before is one of the $k$ derived concepts. But in $T$, each row is a vector in $k$-space describing a term in the original collection, a term in the original term-by-document matrix that characterized the collection. Hence, “[e]ach term [in this matrix] is then characterized by a vector of weights indicating its strength of association with each of these underlying concepts.” [JASIS, 1990,
Deerwester et al.] In other words, each term vector (i.e., row) in $T$ is “a weighted average of the different meanings of the term.” [SIGIR ’94, Hull] The diagonal elements in the 2nd matrix $S$ assign weights (called “singular values”) to the $k$ LSI factors according to their significance. This allows the user to have some control over how many dimensions $k$-space is to have.

“[Some of] [t]he power of this decomposition comes from the fact that the new factors are presented in order of their importance (as measured by the diagonal of $S$). Therefore, the least important factors can easily be removed by truncating the matrices $T$, $S$, and $D$, i.e., by deleting some of the rightmost columns of these matrices. The remaining $k$ columns [are] called the LSI factors.” SIGIR ‘94, Hull] Note that $k$ is a parameter under the user’s control. Reducing $k$ can eliminate “noise”, e.g., “rare and less important usages of certain terms.” However, if the number of dimensions (LSI factors) is too low, important information may be lost. The optimum number of dimensions obviously depends on the collection and the task. One report finds that improvement starts at about 10 or 20 dimensions, peaks between 70 and 100, and then decreases. [SIAM Review, Berry et al., 1995] As the number of LSI factors approaches the number of terms, performance necessarily approaches that of standard vector methods. Another report says that the optimum number of dimensions is usually between 100 and 200.

Projection of a set of documents into $k$ space is optimal in the sense that the projection “is guaranteed to have, among all possible projections to a $k$-dimensional space, the lowest possible least square distance to the original documents. In this sense, LSI finds an optimal solution to the problem of dimensionality reduction.” {Schutze et al., SIGIR ’97]

What does it mean to say that the $k$ factors derived by the LSI procedure correspond to “artificial” concepts? It means that no attempt is made to interpret these $k$ concepts, e.g., to describe them in simple English. Indeed, in many cases, it may not be possible to summarize these concepts, to explain what each one “means.” What one can say is that a given document is heavily weighted with regard to concept 1, doesn’t deal at all with concept 2, is lightly weighted with respect to concept 3, etc.

What good does it do describe a document in terms of the relative importance to the document of $k$ concepts, if one doesn’t know what the $k$ concepts mean? For a single document, such a description may have no value at all. But if one has a 2nd document also described in terms of weights of those same $k$ concepts, then one can say how
similar the documents are (in $k$-space). And, if one of those “documents” is a query (or a sample document used as a query, or the centroid of a set of sample documents), then one can say how close the given document is to the given query, in $k$-space of course. Following the usual vector space similarity methods, e.g., calculating the cosine similarities, one can rank documents by how similar they are to the query, in $k$-space. Similarity in $k$-space is more statistically meaningful, and therefore, one hopes, more semantically meaningful, than similarity in conventional term space, because the $k$ concepts reflect statistical correlations in the document population, while the original terms do not.

Since one can compute query-document similarities using the $k$-by-document matrix, $D$, alone, what is the value of the term-by-$k$ matrix, $T$? One answer is that it allows you to compute term similarities. Presumably, two terms are very similar if they co-occur, i.e., are strongly correlated, with many of the same other terms. Hence, such a similarity can be used to suggest to a user who enters a query, other terms, statistically similar to the terms he used, which could be added to his query. Or the similarities can be used to construct automatically a domain-dependent or collection-dependent thesaurus.

Notice, by the way, that although each row of matrix $T$ is called a “term vector,” the phrase is used quite differently in LSI terminology than in conventional vector space terminology. In the conventional vector space approach, a “term vector” is a vector in document space describing a document in terms of weights assigned to each term for the given document. In LSI, both terms and documents are described in LSI factor $k$-space. A term vector is a vector describing a given term in LSI $k$-space in terms of the weights assigned to the LSI factors for the given term. A document is described in LSI by a document vector specifying the weights assigned to the LSI factors for the given document.

Hearst, et al. [Text Filtering Conference, TREC 4] point out an additional advantage of LSI with respect to the routing or classification of documents:

The routing task can be treated as a problem of machine learning or statistical classification. The training set of judged documents is used to construct a classification rule which predicts the relevance of newly arriving documents. Traditional learning algorithms do not work effectively when applied to the full vector space representation of the document collection due to the scale of the problem … In the vector space model, one dimension is reserved for each unique term in the collection. Standard classification techniques cannot operate in such a
high dimensional space, due to insufficient training data and computational restrictions. Therefore, some form of dimensionality reduction must be considered … [One approach is to] apply Latent Semantic Indexing (LSI) to represent documents by a low-dimensional linear combination of orthogonal indexing variables.

Berry, et al. {SIAM Review, 1995} discuss another advantage of LSI. It is especially useful for noisy input:

Because LSI does not depend on literal keyword [i.e., term] matching, it is especially useful when the input text is noisy, as in OCR (optical character recognition), open input, or spelling errors. If there are scanning errors, and a word [name in this case] (Dumais) is misspelled (as Duniais), many of the other words in the document will be spelled correctly. If these correctly spelled context words also occur in documents that contain a correctly spelled version of Dumais, then Dumais will probably be near Duniais in the $k$-dimensional [LSI factor] space.

On the other hand, LSI has some serious drawbacks too. As Hull [SIGIR ‘94] points out:

While a reduced representation based on a small number of orthogonal variables might appear to cut storage costs substantially [compared to the traditional term-based vector space model], the opposite is actually true … [The LSI representation requires storage of a substantially larger set of values.] In addition the LSI values are real numbers while the original term frequencies [weights] are integers, adding to the storage costs. Using LSI vectors, we can no longer take advantage of the fact that each term occurs in a limited number of documents, which accounts for the sparse nature of the term by document matrix.

Another disadvantage is that “the LSI solution is also computationally expensive for large collections … [However], it need only be constructed once for the entire collection [assuming a relatively static collection so] performance at filtering time is not affected.” [SIGIR ‘94, Hull]

In one respect, LSI degrades filtering time performance too. In a conventional vector space search using an inverted index, only documents that have some terms in common with the query must be examined. If the query is well-formulated, i.e., is composed of terms that are not overly common and serve to distinguish relevant from non-relevant documents, many documents will not contain any terms in common with the query and hence will not need to be examined at all. “With LSI, however, the query must be compared to every document in the collection.” [SIGIR ‘94, Hull] But this is not as great
a disadvantage for LSI as it may at first appear. If (as will commonly be the case) the number of terms in a conventional query is greater than the number of factors in its LSI representation, the vector similarity calculation for a given document in conventional term space will take more time to compute than the corresponding calculation in LSI vector space. Moreover, most conventional vector space approaches use some form of query expansion to modify and expand the user’s original query. (This is discussed below in the section on Query Expansion and Refinement.) LSI can be viewed as a special kind of query “expansion”. [SIGIR ‘94, Hull] Conventional query expansion often results in a great increase in the number of terms in the query, whereas LSI may actually reduce the number of terms.

Given that the original LSI solution is computationally expensive, the question arises whether this expense must be incurred repeatedly each time new messages are added to the document collection. (This is an important consideration in any case where the document collection is not static, but it is especially important if the LSI technique is applied to a routing application; in such an application, the document “collection” is an incoming stream that is continually changing see section 8.) Fortunately, it is not always necessary to do a complete re-computation every time a new document arises. To begin with, the addition of one document to a large collection is not likely to have a very significant effect on the LSI computation, so it may be possible to ignore the effect. Secondly, there are two approaches to updating the LSI computation without re doing the entire computation. [Berry et al., 1995]

The first, called “folding in,” is the cheapest. Basically, you don't recompute the $k$ factors at all. Nor do you recompute the weights of the $k$ factors for existing documents or terms. Instead each new document just becomes a new column in the document matrix, described in terms of the original $k$ factors. Similarly, if the new document contains some new terms, each new term becomes a new row in the term matrix, again defined in terms of the original $k$ factors. So the process is relatively fast and cheap, but there is some degradation; not all the new correlations are being absorbed. Hence, all the original documents and terms occupy the same positions in $k$-space that they did before the folding-in of new documents occurred.

There is a more sophisticated (and naturally more computationally expensive) technique called “SVD updating.” It starts with the original LSI database, just like folding
in, but the weights associated with the k factors for each document and term are recomputed so that the addition of new documents and terms affects the positions of the existing documents and terms in k-space. Hence, the approximation is much better.

If these two updating procedures prove inadequate, the remaining alternative is to redo the LSI computation from scratch.

The fact that (as noted above) an LSI term vector is “a weighted average of the different meanings of the term” can be either an advantage or a disadvantage. It is an advantage if the reduced representation in LSI removes “some of the rare and less important usages” of the given term, i.e., usages that are not relevant to the topic of the query. On the other hand, if the “real meaning [as used in the query and the relevant documents] differs from the average meaning [as captured by the LSI term vector], LSI may actually reduce the quality of the search.” [Hull, SIGIR ‘94]

One more possible drawback of LSI is pointed out by Shutze et al. [SIGIR ‘95] LSI is not required for, and may actually degrade filtering of, documents that are well described by a few terms. They give the example of a document about the Hubble Space Telescope, which can be very well retrieved by the single word “Hubble.” LSI may actually obscure the key evidence, the presence or absence of the term “Hubble,” in this case. On the other hand, LSI is much more effective “if there is a great number of terms which all contribute a small amount of critical information.” This is particularly true if each of the documents on the desired topic only contains a subset of these terms. In such a case, LSI is more effective than a term-based classifier at combining evidence to identify documents about the given topic.

Similarity (between query and document) in LSI vector space is usually calculated by the inner product similarity measure as in the traditional document vector space approach. The normalized inner product, i.e., “cosine” similarity is generally used. However, Bartell et al. have shown that the “inner product similarities between documents in the original [term] space are optimally preserved by the inner products [not normalized inner products] between corresponding vectors in the reduced space.” [Bartell, SIGIR ‘92] Hence, cosine normalization should be computed in the original term space and the LSI calculation then applied to these normalized vectors.

In any case, the value of LSI lies in (1) the elimination of spurious similarities, i.e., due to the same term being used in two different ways, and (2) the detection of similarities in LSI space that are invisible in ordinary term space, i.e., due to different but synonymous terms being used in similar contexts in different documents.
2.3.6 Vectors Based on \( n \)-gram Terms

The \( n \)-gram approach is in some respects the ultimate in vector space (and more generally, in statistical) approaches to IF. In the traditional vector space approaches described above, the dimensions of the document space for a given collection of documents are the words (or sometimes phrases) that occur in the collection; more precisely, they are the terms that remain after stemming, and removal of words that appear on a stoplist. By contrast, in the \( n \)-gram approach, the dimensions of the document space are \( n \)-grams, strings of \( n \) consecutive characters extracted from the text without regard to word length, and often completely without regard to word boundaries. Hence, the \( n \)-gram method is a remarkably “pure” statistical approach, one that measures the statistical properties of strings of text in the given collection without regard to the vocabulary, lexical, or semantic properties of the natural language(s) in which the documents are written.

The \( n \)-gram length (\( n \)) and the method of extracting \( n \)-grams from documents vary from one author and application to another. “Zamora uses trigram \([n = 3]\) analysis for spelling error detection” [Pearce & Nicholas, JASIS, 1996]. Damashek uses \( n \)-grams of length 5 and 6 for clustering of text by language and topic (see below). He uses \( n = 5 \) for English and \( n = 6 \) for Japanese [Damashek, Science, 1995]. Pearce and Nicholas follow Damashek in using 5-grams to support a dynamic hypertext system [JASIS, 1995]. Some authors [Zamora et al., IP&M, 1981]; [Suen, IEEE Pattern, 1979] draw \( n \)-grams from all the words in a document but use only \( n \)-grams wholly within a single word. Others [Cavnar, TREC-2, 1993]; [Yannakoudakis et al., IP&M, 1982]; [Damashek, Science, 1995] also use \( n \)-grams that cross word boundaries, i.e., that start within one word, end in another word, and include the space characters that separate consecutive words [Pearce & Nicholas, JASIS, 1996].

Damashek’s sliding window approach [Science, 1995] is one of the most recent and inclusive, and the first to offer “convincing evidence of the usefulness [of the \( n \)-gram approach] for the purpose of categorizing text in a completely unrestricted multilingual environment.” Minimal preprocessing is required. Numbers and punctuation characters are usually removed. What remains is the alphabet plus the space character (27 characters for English). (Sometimes, no preprocessing is done at all.) The \( n \)-grams characterizing a document are then obtained by moving a window \( n \) characters in
length through a document or query one character at a time. In other words, the first $n$-gram will consist of the first $n$ characters in the document, the 2nd $n$-gram will consist of the 2nd through the $(n+1)$th character, etc. Hence, there will be an $n$-gram starting with every character in the document (after preprocessing) except for the last $n—1$ characters. Each document can then be specified as a vector of $n$-grams, one for each distinct $n$-gram in the document. Each component can be weighted just as the components of a conventional term vector are weighted, e.g., Damashek uses normalized $n$-gram frequency, the number of occurrences of the given $n$-gram in the given document divided by the total number of occurrences of all $n$-grams in the document. Similarly, Damashek computes the similarity between two documents using the familiar cosine similarity measure, which is just as useful in an $n$-gram document space as in a term-based document space.

Once the similarities among the $n$-gram-based document vectors have been computed, the documents can be clustered using a method such as those discussed in section 3.8. Damashek found this approach to be extremely effective for classifying a mixed-language collection of text documents, generating a distinct cluster for each distinct language. Indeed, such methods have been used to cluster documents hierarchically, i.e., by language group and by individual language within group. The beauty of this method is that it amounts to “blind clustering,” i.e., the documents are classified without any prior linguistic knowledge. That is, there is no prior knowledge of the individual languages, or even of how many languages or language groups are involved.

A document space of $n$-gram-based vectors can be clustered by topic, as well as language (or by topic within language). However, clustering by topic introduces a problem not usually present in language clustering. When documents are blind-clustered by language, stoplists are not only not available (since they are language-dependent); they are also inapplicable. The very words that are typically placed on stoplists because they occur across most topics within the documents of a given language are the words that best characterize the language statistically. Putting it another way, words that have very little semantic content are good discriminators of a language since the objective is to classify all documents written in the given language regardless of what topic they discuss. Hence, Damashek employs a language-independent method of removing the “noise”, the data that is common across topics. He translates the axes of the vector space
so that the new origin is at the mean (the centroid) of all the document vectors. In that way, the origin becomes “a location that characterizes the information one wishes to ignore,” the information common to the collection of documents. This location is equivalent to the “noise,” the data that does not serve to distinguish one document from another. Translating the origin is equivalent to subtracting the centroid from each document vector, subtracting the common information one wishes to ignore. The document similarities can then be recalculated relative to this new origin.

But the technique of subtracting the centroid can be used to accomplish more than merely suppressing noise. It can also be used to recognize that two clusters of documents with different primary topics share a secondary topic. (Damashek offers the example of one cluster dealing with the primary topic of health care reform, and another cluster dealing with the primary topic of communicable diseases; they may share a secondary topic of AIDS epidemiology.) If one subtracts from each cluster its centroid (corresponding to its primary topic, the topic that all the documents share), the two clusters may become superimposed in document space, reflecting the secondary topic that they share. In this kind of application, the common data that is removed is not devoid of semantic content. It is merely common to a set of documents and hence not useful for distinguishing them. Hence, it has been called the “context” or the “background” [Cohen, JASIS, 1995] rather than “noise.”

A more sophisticated n-gram weighting scheme based on the $G^2$ statistic [Bishop et al., 1975] has been used by Cohen [JASIS, 1995] to distinguish the background of a cluster of documents from the highlights, i.e., the words in a document that distinguish the document from its neighbors and hence can serve as an automatically-generated abstract that tells a user very rapidly what a given document is “about.” Note that Cohen’s method, although it highlights words, identifies the words to be highlighted by statistical characteristics of the n-grams of which they are composed. Moreover, Damashek’s sliding window approach is employed, enabling phrase as well as word high- lights to be identified. As before, the technique is language-independent.

Damashek’s use of cosine similarity illustrates a general point: almost any method for weighting terms, normalizing terms, or normalizing term-based vectors is as applicable when the terms are n-grams as when the terms are words. Similarly, query expansion based on relevance feedback, discussed in section 3.6 below, is as applicable to n-grams
as to words. (In practice, query expansion has not usually been combined with Damashek’s \(n\)-gram method, because emphasis has been placed on its power for fast, inexpensive clustering, and interactive browsing.) The converse is also true: A method like centroid subtraction, applied above to \(n\)-gram vectors, is applicable to word-based term vectors as well. Language-dependent methods such as stoplist removal are not directly applicable to \(n\)-gram vector representations, but as noted below, can be combined effectively with \(n\)-gram analysis.

Because the sliding window used to obtain \(n\)-grams allows the system to obtain many slices of a given word, the performance of an \(n\)-gram system is remarkably resistant to textual errors, e.g., spelling errors, typos, errors associated with optical character recognition, etc. Damashek [Science, 1995] artificially corrupted his text (15% of the characters in error) and found that most of the documents in an uncorrupted cluster were still identified as belonging to the cluster (17 of 20). In a hypertext application [Pearce and Nicholas, JASIS, 1996], “the dynamic linkage mechanisms … are tolerant of garbles in up to 30% of the characters in the body of the text.” Again, a major virtue of the sliding window \(n\)-gram approach for tolerance of garbles is that it is language-independent; it does not depend on prior linguistic knowledge.

A pure \(n\)-gram analysis does not use language-specific and semantic clues, i.e., stemming, stoplists, syntactically-based phrase detection, thesaurus expansion, etc. This theoretically limits its performance compared to methods that make effective use of language-specific as well as statistical clues. However, this limitation is currently more theoretical than actual, because most contemporary filtering methods make only limited use of language-specific and semantic methods (a situation that may change in the future). Moreover, as noted below, \(n\)-gram analysis can be combined with word-based, syntactic, and semantic methods in a variety of ways. Further, as discussed above, many methods, e.g., automatic query expansion based on relevance feedback, are as applicable to \(n\)-gram analysis as to word-based analysis. On the other hand, precisely because \(n\)-gram analysis is language-independent, it is especially well-suited (given an adequate training set) to classification or clustering of documents by language. It should be noted here that the “languages” that this technology can classify are not restricted to natural languages, but can also include programming languages, indeed any class of “language” or representation that has distinguishable statistical properties.
The disadvantage of \textit{n}-gram analysis with respect to language-specific processing is actually less serious than it might seem. Algorithms that use \textit{n}-gram counts can be combined with identification of word boundaries to recognize roots shared by different words, conferring some of the same advantage as stemming algorithms, but with the further advantage of language independence. In some respects, these algorithms are better than conventional stemmers that only remove suffixes, e.g., they can recognize the resemblance of “quake” and “earthquake” [Cohen, JASIS, 1995]. (Of course, they can be deceived by spurious resemblances too.) And centroid subtraction, discussed below, provides some of the same benefits as stopword removal, but again in a language-independent fashion. Indeed, it is not always necessary to choose between \textit{n}-gram analysis, and language-dependent methods. They can be combined. For example, language-dependent methods have been used successfully in conjunction with \textit{n}-gram analysis to improve performance, e.g., if the language in which a collection of documents is written is known, words on a stoplist can be removed from each document before \textit{n}-gram analysis is applied. [Onyshkevych, PC] Similarly, conventional stemming can be applied before \textit{n}-gram analysis. However, the future extension of textual analysis to sophisticated semantic and knowledge-based methods will obviously have to be word-based.

Apart from language independence, the greatest virtue of \textit{n}-gram analysis is that it is very simple to implement, and can run very fast (although the redundancy associated with a sliding window approach may make it expensive with respect to storage). Hence, another very effective way to combine \textit{n}-gram analysis with language-based methods is to perform a two-stage search process. First, \textit{n}-gram methods can be used to zero in rapidly, e.g., in an interactive browsing mode, on document clusters in a domain of interest; then, more refined and expensive language-based methods can be used to refine the search and perform the final filtering.

But perhaps the feature of the \textit{n}-gram approach that most sets it apart from other statistical methods, is its ability to group documents without any prior knowledge about the documents being grouped. Indeed, in this realm of “blind” clustering, it appears to have no serious competition.
2.4. Probabilistic Approach

There is no clear line separating probabilistic from statistical methods of IF. Indeed, there is a very close connection since probabilities are often calculated on the basis of statistical evidence. Most of the literature on probabilistic IF assumes that the evidence is so calculated. Of course, given a formula based on a probabilistic model, any source of evidence can be used to compute the probabilities to be plugged into the formula, but as a practical matter the evidence is usually statistical; in fact, it is may sometimes be the very same evidence, e.g., tf’s and idf’s, used in statistical, e.g., vector space, methods.

2.4.1 What Distinguishes a Probabilistic Approach?

Then what distinguishes a true probabilistic methodology from other statistical approaches? According to Cooper et al. [SIGIR ‘92]:

In a thoroughgoing probabilistic design methodology, serious use is made of formal probability theory and statistics to arrive at the estimates of probability of relevance by which the documents are ranked. Such a methodology is to be distinguished from looser approaches for instance the “vector space” filtering model in which the retrieved items are ranked by a similarity measure (e.g., the cosine function) whose values are not directly interpretable as probabilities.

2.4.2 Advantages and Disadvantages of Probabilistic Approach to IF

Cooper et al. [SIGIR ‘92] list four potential advantages of a true probabilistic design methodology:

1. “One has grounds for expecting filtering effectiveness that is near optimal relative to the evidence used.”

2. There should be “less exclusive reliance on traditional trial and error filtering experiments … to discover the parameter values that result in best performance.” (As examples of such trial and error, consider the variety of term weighting schemes that have been tried in varied vector space experiments or the trials required to determine optimum values for the parameters, A, B, and C in the Rocchio relevance feedback formula, discussed in a later section.)

3. “[A]n array of more powerful statistical indicators of predictivity and goodness of fit [than precision, recall, etc.] become available.”
4. “[E]ach document’s probability-of-relevance estimate can be reported to the user in ranked output … [I]t would presumably be easier for most users to understand and base their stopping behavior [i.e., when they stop looking at lower ranking documents] upon … a ‘probability of relevance’ than [a cosine similarity value].” In general, actual estimates for each document of probability of relevance are more useful to a user than mere ranking of documents by probability of relevance (let alone ranking by some other similarity function). [Turtle and Croft, ACM Trans IS, 1991]

Yet, probabilistic methods have not yet been as widely used as these advantages would suggest. Moreover, where they have been used, they have achieved filtering performance (measured by precision and recall) comparable to, but not clearly superior to, non-probabilistic methods. [Cooper, SIGIR ‘94] Cooper identifies various reasons for these shortfalls:

1. “[A]dvocates of nonprobabilistic methods … regard the formulation of exact statistical assumptions as an unnecessary theoretical burden on the researcher. They maintain (with some plausibility) that the time and effort spent on such analysis would be better spent on ad hoc experimentation using formalisms looser and friendlier than probability theory.”

2. “The estimation procedures used in probabilistic IF are usually based on statistical simplifying assumptions or ‘models’ of some sort. The filtering clues that bear on a document’s probability of usefulness must somehow be combined into a single relevance probability, and modeling assumptions are needed to accomplish the combining. Typically, the assumptions adopted for the task are crude and at best only approximately true … The introduction of simplifying assumptions known to be less than universally valid surely compromises to some degree the accuracy of the probability estimates that result.” (Of course, similar simplifying assumptions are tacitly used in all other statistical approaches, e.g., the assumption of term independence in the vector space model.)

3. The assumptions underlying some IF models, most notably the widely used (and misnamed) “Binary Independence” IF model, can lead to logical inconsistencies. [Cooper, SIGIR ‘91, ACM Trans IS, 1995] Successes that have been achieved in spite of the inconsistency of the theoretical model are due to the fact that the actual assumptions used in practice are different than the assumptions of the theoretical model, and stronger than they needed to be.
In a probabilistic method, one usually computes the “conditional” probability $P(D|R)$ that a given document $D$ is observed on a random basis given event $R$, that $d$ is relevant to a given query. [Salton, ATP, 89] [van Rijsbergen, 1979] If, as is typically the case, query and document are represented by sets of terms, then $P(D|R)$ is calculated as a function of the probability of occurrence of these terms in relevant vs. non-relevant documents. The term probabilities are analogous to the term weights in the vector space model (and may be calculated using the same statistical measures). A probabilistic formula is used to calculate $P(D|R)$, in place of the vector similarity formula, e.g., cosine similarity, used to calculate relevance ranking in the vector space model. The probability formula depends on the specific model used, and also on the assumptions made about the distribution of terms, e.g., how terms are distributed over documents in the set of relevant documents, and in the set of non-relevant documents.

More generally, $P(D|R)$ may be computed based on any clues available about the document, e.g., manually assigned index terms (concepts with which the document deals, synonyms, etc.) as well as terms extracted automatically from the actual text of the document. Hence, we want to calculate $P(D|A, B, C, ...)$, i.e., the probability that the given document, $D$, is relevant, given the clues $A, B, C$, etc. As a further complication, the clues themselves may be viewed as complex, e.g., if the presence of term $t$ is a clue to the relevance of document $D$, $t$ may be viewed as a cluster of related clues, e.g., its frequency in the query, its frequency in the document, its $idf$, synonyms, etc. This has led to the idea of a “staged” computation, in which a probabilistic model is first applied to each composite clue (stage one), and then applied to the combination of these composite clues (stage two). [Cooper et al., SIGIR ’92] This is discussed further below, in the section on Logistic Regression. It has also led to the idea of an “inference net” [Turtle & Croft, ACMtransIS, 1991] in which rules can be specified for combining different sources of evidence (automatically extracted index terms, manually assigned index terms, synonyms, etc.) to compute a “belief” that an information need has been satisfied by a given document. (See below, in the section on the Bayesian Inference Network Model.)

### 2.4.3 Linked Dependence

As noted above, the first step in most of these probabilistic methods is to make some statistical simplifying assumption. The objective is to replace joint probabilities (the probability of occurrence of two or more events, $A, B$, etc.) by a separate probability for
each event. The most widely used assumption is “Binary Independence” which Cooper [SIGIR ‘91, ACM Trans IS, ‘95] points out should really be called “Linked Dependence” [Cooper et al., SIGIR ‘92]. The model has been called “Binary Independence” because it has been assumed that to arrive at the model one must make the simplifying assumption that the document properties that serve as clues to relevance are independent of each other in both the set of relevant documents and the set of non-relevant documents, an “implausible presumption” [Cooper et al., SIGIR ‘92]. Cooper shows that it is sufficient to make the weaker “Linked Dependence” assumption that these properties are not independent but that the same degree of dependence holds for both the relevant document set and the non-relevant document set. In symbols (for two properties, A and B):

$$P(A \cap B | R) = K \cdot P(A | R) \cdot P(B | R)$$

$$P(A \cap B | \neg R) = K \cdot P(A | R) \cdot P(B | \neg R)$$

where $\neg R$ means non-relevant. $K$ is a crude measure of the dependence (the “linkage”) of properties A and B. If $K = 1$, this reduces to the pure independence property; the joint probability equals the product of the individual properties. Cooper points out that the weaker linked dependence assumption is sufficient to satisfy the requirements of models that have hitherto used the binary independence assumption. “This weaker assumption, though still debatable, has at least the virtue of not denying the existence of dependencies.” Therefore, efforts to remedy the clearly erroneous assumption of pure independence by providing empirical term co-occurrence information are less theoretically important than previously thought; the actual error that needs to be remedied is the assumption that dependencies are the same for relevant and non-relevant document sets. Of course, the latter may still be a significant consideration.

### 2.4.4 Bayesian Probability Models

Conditional (“Bayesian”) probabilistic models relate the *prior* probability of document relevance, i.e., the probability that a document selected at random will be relevant, to the *posterior* probability of relevance, i.e., the probability that an observed document is relevant, given the observed (or computed) features of the document. These features may include terms in the document, term statistics, manually assigned descriptors, phrase, etc., indeed all the clues employed in other statistical methods. Bayesian models vary
primarily in how the “posterior” probabilities are calculated. They may also vary in how multiple sources of probabilistic evidence are combined, and in how probabilistic and non-probabilistic evidence are combined.

An important feature of Bayesian models is that one starts with a set of prior probabilities that must sum to 1, and one ends up with a set of posterior probabilities that must also sum to 1. [Winkler et al., Stat] (Some probabilistic methodologies may yield document filtering status values (RSVs) that are not true probabilities, but rather are monotone to the corresponding true probabilities. In that case, the RSVs can be used for document ranking but do not sum to one. However, in a true probabilistic model, they can always be converted to true probabilities (normalized) which do sum to one and which will be more meaningful to an end user.) One can view Bayesian probability as applied to IF as a process of “redistributing” probabilities of relevance from the prior probability distribution over all the documents in a given collection to a posterior distribution e.g., over the set of retrieved documents. Crestani and van Rijsbergen [SIGIR ‘95] call this “probability kinematics.” They view this kinematics as a flow of probabilities among the terms serving as descriptors of a document collection. Initially, one has a prior distribution of the probabilities that terms, e.g., are good document descriptors (perhaps based on idf’s). Given a particular document $D_i$, one has a “flow” of probabilities from terms not in $D_i$ to terms in $D_i$, e.g., perhaps based on simple term occurrence or term frequencies in the given document. These posterior probabilities for the terms that describe $D_i$ also sum to one. The probability of the given document being relevant to a given query is then the sum of the posterior probabilities for the document terms that are also in the query. More generally, this probability may also depend on, e.g., term frequencies or other statistical characteristics of the query.

### 2.4.4.1 Binary Independence Model

“The simplest of these models is based on the presence or absence of independently distributed terms in relevant and non-relevant documents,” [Shaw, IP&M, 1995] i.e., the distribution of any given term over the collection of documents is assumed to be independent of the distribution of any other term. Put another way, the probability of any given term occurring in a relevant document is independent of the probability of any other term occurring in a relevant document (and similarly for non-relevant documents). Hence, the model is referred to as the “binary independent” [BI] model.
We saw above, in the preceding section, that this model actually should be called the “linked dependence” model because it actually depends on the weaker assumption that these probabilities are not independent, but rather that the same degree of dependence holds among relevant documents as holds among non-relevant documents, and that this degree of dependence can be captured by a proportionality constant. [Robertson et al., JASIS, 1976] [van Rijsbergen, 1979]. In this model, also known as the “relevance weighting theory” [Efthimiadis, IP&M, 1995], we start with $p_k$, the probability that term $t_k$ appears in a document given that the document is relevant and $u_k$, the probability that $t_k$ appears in a document given that the document is non-relevant. (As always, relevance is defined relative to a given query or “information need.”) Using Bayes’ rule of inference and the BI (more precisely, linked dependence) assumption, one can derive a function for ranking documents by probability of relevance. [van Rijsbergen, 1979] In this function, each term $t_k$ receives a weight $w_k$ given by:

$$w_k = \log \frac{p_k (1-u_k)}{u_k (1-p_k)}$$

The “odds” of $t_k$ appearing in a relevant document is $p_k/(1-p_k)$. Similarly, the “odds” of $t_k$ appearing in a non-relevant document is $u_k/(1-u_k)$. Hence, $w_k$ measures the odds of $t_k$ appearing in a relevant document divided by the odds of $t_k$ appearing in a non-relevant document, i.e., the odds ratio. Taking the log makes this function symmetric: $w_k = 0$ if $p_k = u_k$, is positive if $p_k > u_k$, and is negative if $p_k < u_k$. Hence, $w_k$ is a good measure of how well a term can distinguish relevant from non-relevant documents. The function $w_k$ is called the “term relevance” weight or “term relevance function,” or “logodds,” i.e., the log of the odds ratio for $t_k$. Plainly, $w_k$ will be a very large positive number for a term that has a high probability of appearing in a relevant document and a very low probability of appearing in a non-relevant document (and a very large negative number if the probabilities are reversed). Moreover, if the set of index terms in a document collection satisfies the BI or “linked dependence” condition, the odds ratio (odds of relevance divided by odds of non-relevance) for a given document $D$ is merely the product of the odds ratios of all the $t_k$ appearing in $D$, i.e., the product of all the corresponding $w_k$. Hence, the log of the odds ratio for $D$ can be computed as the sum of the $w_k$. In other words, the logodds of relevance of $D$ is computed as the sum of the $w_k$ for index terms appearing in $D$. 
The trick is to find a way of computing $p_k$ and $u_k$. If relevance data is available, e.g. from manually evaluating a previous run with the same query against the same collection or against a training set, then $p_k$ and $u_k$ can be estimated from a 2x2 “contingency” table summarizing the relevance judgments, as given below:

**Table 2: Contingency Table of Relevance Judgments**

<table>
<thead>
<tr>
<th>No. of documents including term $t_k$</th>
<th>No. of relevant Documents</th>
<th>No. of non-relevant Documents</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of documents excluding term $t_k$</td>
<td>$R-r$</td>
<td>$(N-R) - (n-r)$</td>
<td>$N-n$</td>
</tr>
<tr>
<td>Total</td>
<td>$R$</td>
<td>$N-R$</td>
<td>$N$</td>
</tr>
</tbody>
</table>

Here $N$ is the total number of documents in the collection, $n$ is the total number of documents that contain term $t_k$, $R$ is the total number of relevant documents retrieved, and $r$ is the total number of relevant documents retrieved that contain term $t_k$. From this table, we can estimate $p_k$ as $r/R$ (the proportion of relevant documents containing $t_k$), and $u_k$ as $(n-r)/(N-R)$ (the proportion of non-relevant documents containing $t_k$). Obviously, this assumes that “the term distribution in the relevant items previously retrieved [or in the training set] is the same as the distribution for the complete set of relevant items, and that all non-retrieved items [$N-R$] can be treated as non-relevant.” [Salton *et al*, JASIS, 1990]. The latter assumption is necessary to allow us to assume that $N-R$ (all retrieved non-relevant documents plus all non-retrieved documents) equals the total number of non-relevant documents. The former assumption allows us to treat the proportion of relevant documents containing $t_k$ in the retrieved sample as characteristic of the proportion in the complete collection, for all $t_k$.

Equivalently, the odds of $t_k$ appearing in a relevant document is $(r/R)/(1-r/R) = r/(R-r)$, and the odds of $t_k$ appearing in a non-relevant document is $(n-r)/(N-R)/(1-(n-r)/(N-R)) = (n-r)/(N-R-n+r)$. Inserting these values into the formula for $w_k$, we obtain:
This formula for $w_k$ obviously breaks down if $p_k$ equals one ($r = R$) or zero ($r = 0$). Similarly, $w_k$ breaks down if $u_k$ equals one ($n-r = N-R$) or zero ($n = r$). Statistical theory has been used to justify modifying the formulas for $p_k$ and $u_k$ to avoid these singularities by adding a constant $c$ to the numerator and one to the denominator, where $c = 0.5$ or $c = n/N$. [Robertson et al., 1986] However, there are always cases where these constants dominate the computation and distort the results. [Shaw, IP&M, 1995] Shaw proposes to avoid these problems by using the unmodified formulas everywhere but at the singularities, and specifying alternative formulas at the singularities.

If the constant 0.5 is inserted in the formula for $w_k$, the result is:

$$w_k = \frac{(r + 0.5)(N - R - n + r + 0.5)}{(R - r + 0.5)(n - r + 0.5)}$$

It should be noted that in the term relevance model described above, the probabilities of relevance and non-relevance, given $t_k$, and the corresponding logodds function $w_k$, are based entirely on the presence or absence of each term $t_k$ in relevant and non-relevant documents. A term $t_k$ is favored, i.e., given a high $w_k$, if it appears much more frequently in relevant documents than in non-relevant documents. It receives no “extra credit” for appearing more frequently than another term $t_j$ in relevant documents.

The simple term relevance weight given above is based on the contingency table, which is constructed on the basis of a training sample. Hence, the term relevance weight function is based entirely on the assumption that the user possesses a training sample that is adequate in size and representative of the collection(s) to which the function is to be applied. What if one has no training sample? Put another way, what should the “prior” term weight be, before any data from the collection is sampled? Robertson et al. [SIGIR ‘97], argue that this prior weight should reflect the fact that terms occurring in a large proportion of documents have little value for predicting relevance. Hence, they advocate using the idf as the prior weight, or (equivalently) the weight to be used in the absence of relevance information. On the other hand, if ample term relevance information
is available, the term relevance function above applies. Hence, they propose a term weight function that varies smoothly from \( idf \) (for zero relevance data) to the above term relevance function, \( w_k \), for ample relevance data.

Moreover, ample relevance data means not only an adequate sample of relevant documents, but an adequate sample of non-relevant documents. Hence, Robertson et al. define \( S \), the number of known non-relevant documents (analogous to \( R \) for relevant documents), and \( s \), the number of known non-relevant documents containing a given term, \( t_k \) (analogous to \( r \)). Note that \( S \) is not the same as \( N-R \), and \( s \) is not the same as \( n-r \).

Given these six variables, \( R, r, S, s, N, \) and \( n, \) Robertson et al. define a function that varies from pure \( idf \) (for \( R=0, S=0 \), i.e., no data available about relevance and non-relevance) to the above \( w_k \) term relevance function for large \( R \) and \( S \). They begin by observing that the logodds formula above, the log of the ratio of the odds of relevance to the odds of non-relevance, stated in terms of \( p_k \) and \( u_k \), can be rewritten as:

\[
w_k = \log \left( \frac{p_k}{1-p_k} \right) - \log \left( \frac{u_k}{1-u_k} \right)
\]

The first term above is the logodds of relevance given the presence of term \( t_k \). The second term is the logodds of non-relevance, given the presence of term \( t_k \). Robertson et al. express each of these terms as a linear sum of an \( idf \) term and a term relevance term. The resulting weight function is:

\[
W^{(1)} = \frac{k_4}{k_5 + \sqrt{R}} \left( k_4 + \log \frac{N-n}{N} \right) + \frac{\sqrt{R}}{k_5 + \sqrt{R}} \log \frac{r+0.5N}{R-r+0.5}
\]

\[
- \frac{k_6}{k_6 + \sqrt{S}} \log \frac{n}{N-n} - \frac{\sqrt{S}}{k_6 + \sqrt{S}} \log \frac{s+0.5}{S-s+0.5}
\]

Here \( k_4, k_5, \) and \( k_6 \) are “tuning constants” that can be adjusted to tune the weighting function. The function is based on the “assumption... that the effect should be linear in the square root of \( R \), on the grounds that the standard error of an estimate based on a sample is proportional to the square root of the sample size.” It can be readily seen that if there is no relevance data, i.e., \( R, S = 0 \) (which implies \( r, s = 0 \) too), the weighting function reduces to:
\[ w^{(1)} = k_d + \log \frac{N}{N-n} - \log \frac{n}{N-n} \]

\[ = k_d + \log \frac{N}{n} \]

which is the traditional idf function plus a tuning constant. If \( R \) and \( S \) are large, the 2nd and 4th terms (the two term relevance function terms) dominate, and the weight function reduces to a pure “evidence based,” or “training set based” weight function:

\[ w^{(1)} = \log \frac{r + 0.5}{R - r + 0.5} \log \frac{s + 0.5}{S - s + 0.5} \]

\[ = \log \frac{(r + 0.5)(S - s + 0.5)}{(R - r + 0.5)(s + 0.5)} \]

### 2.4.4.2 Bayesian Inference Network Model

An alternative approach to applying conditional (Bayesian) probability in IF, an approach that “depends less upon Bayesian inversion,” is the inference network filtering model. [Turtle & Croft, ACM Trans IS, 1991] “Inference networks can be used to simulate both probabilistic and Boolean queries and can be used to combine results from multiple queries.” They can also be used to combine multiple sources of evidence regarding the relevance of a document to a user query, e.g., a document may be represented by both terms extracted automatically from the document itself, and terms or concepts assigned manually as index terms. The inference network provides a natural way to combine these sources of evidence to determine the probability (in this context, often called the belief) that the given document satisfies a given user query or information need. The Bayesian inference network approach has been implemented in the INQUERY system. [Callan et al., IP&M, 1995] [Callan et al., DB&ExSysApp, 1992] INQUERY also employs some semantic features, e.g., concept recognizers, which are discussed in a later section. [Callan et al., DB & ExSys, 1992]

An inference network is a probabilistic filtering model, but it differs from typical filtering models. [Croft et al., SIGIR ‘91] A typical model computes “\( P(\text{Relevance} | \text{Document, Query}) \), the probability that a user decides a document is relevant given a particular
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

document and query.” The inference net model computes “P(I|Document), the probability that a user’s information need is satisfied given a particular document.” This probability can be computed separately for each document in a collection. The documents can then be ranked by probability, from highest probability of satisfying the user’s need to lowest. The result is a ranked list of retrieved documents, as with more traditional filtering models. Moreover, the list of retrieved documents can be cut off at a probability threshold, e.g., only retrieve documents with a probability greater than 70% of satisfying the user’s need. Such a probability threshold is likely to be more meaningful to the user than a cosine similarity threshold drawn from the vector space model.

Pearl [Prob Reas, 1988] has an excellent discussion of inference networks in general, and Bayesian networks in particular. He points out that these networks solve an important problem with probabilistic models (and more generally, with “extensional” models):

Interpreting rules as conditional probability statements, \( P(B|A) = p \), does not give us a license to do anything. Even if we are fortunate enough to find \( A \) true in the database, we still cannot assert a thing about \( B \) or \( P(B) \), because the meaning of the statement is “If \( A \) is true and \( A \) is the only thing that you know, then you can attach to \( B \) a probability \( p \).” As soon as other facts \( K \) appear in the database, the license to assert \( P(B) = p \) is automatically revoked, and we need to look up \( P(B|A, K) \) instead.

The point is that \( K \) may cause us to revise or retract conclusion \( B \). The ability to retract a previous conclusion, called “non-monotonic” reasoning, is forbidden in classical logic but is essential to the kind of plausible reasoning under uncertainty that human beings actually do, and which pervades information filtering. The great virtue of inference nets is that they organize our knowledge so that all the propositions, \( A, K \), etc., on which a given conclusion \( B \) depends are immediately accessible, i.e., they are the parents (directly or indirectly) of \( B \). Moreover, a rule for computing the probability of \( B \) from the probabilities of its parents can be attached to the node for \( B \).

The inference network is a graph and consists of nodes connected by directed line segments (“edges”). The nodes are true/false propositions. An edge is drawn from node \( p \) to node \( q \) if \( p \) “causes” or “implies” \( q \). We can then call \( p \) a “parent” of \( q \). As
the network is applied to IF, the root nodes are documents, i.e., propositions of the form, “Document \( D_i \) is observed.” There is a document node for each \( D_i \) in the given collection. These document nodes are parents of “text” nodes. (See Figure 1.) The \( j \)th text node for document \( D_i \) is a physical representation of the text of \( D_i \), i.e., the proposition that, “Text representation \( T_{i,j} \) of document \( D_i \) has been observed.” For most purposes and in most IF systems, no distinction is drawn between the document and its text representation. But drawing this distinction allows for the possibility that a text representation might be shared by several documents, e.g., in a hypertext system several documents might have links to the same shared text. It also allows for the possibility that the children of a document node might include not only text nodes but also audio, video, figure, etc., nodes. (Here we are evidently talking about both multiple representations of a document, and multiple components of a multimedia document.) The distinction between document nodes and text nodes will be ignored in the discussion that follows.

The text nodes in turn are parents of “content representation” nodes (“representation” nodes for short). (See Figure 1.) These are the “descriptors” or “index terms.” As noted above, the text of a document might be indexed both by terms automatically extracted from the document (as discussed in connection with vector space and Boolean models), and manually assigned descriptor terms or concepts. (For example, as Turtle and Croft point out, there may be two representation nodes associated with the phrase “information filtering”, one corresponding to the actual occurrence of the phrase in one or more documents, the other corresponding to the manual assignment of the concept expressed by the phrase to one or more documents, i.e., the human indexer’s judgment that the given documents are “about” information filtering.) Hence, there may be two or more subsets of representation nodes, each corresponding to a different method of describing or representing documents. A given representation node for term \( t_k \) may correspond to the proposition, “\( t_k \) is a good document descriptor.” If \( D_j \) is a parent of \( t_k \), then the link from \( D_j \) to \( t_k \) specifies the probability or “belief” in the proposition that \( t_k \) is a good descriptor of the document, given that the document is \( D_j \). A term is a “good descriptor” of a document if its presence in the given document serves to characterize the document and distinguish it, e.g., statistically, from other documents that are about other topics. Note (see Figure 1 below) that if \( t_k \) occurs in both document \( D_2 \) and document \( D_1 \),
then nodes $D_2$ and $D_i$ are both parents of node $t_k$. And if in addition, concept $c_j$ is assigned manually as a descriptor of $D_2$, then $D_2$ is a parent of both $t_k$ and $c_j$, i.e., document $D_2$ is described (with some probability or to some extent) by concept $c_j$ and contains term $t_k$ with some probability that $t_k$ is a good descriptor of $D_2$.

Given a document $D_1$, one wishes to compute the probabilities of the propositions represented by the child nodes of $D_1$, the children of those nodes, and so on. Hence, an inference net must provide some method of specifying the conditional probability of the proposition at node $q$, given the probabilities of the propositions at its parents, $p_1, p_2$, etc. The mechanism chosen by Turtle et al. is called a “link matrix.” Each node is assigned such a matrix. The link matrix for node $q$ contains two rows; the first row corresponds to proposition $q$ being false, the second row corresponds to $q$ being true. The link matrix for $q$ has a column for each logical combination of parent truth values. For example, if $q$ has three parents, $p_1$, $p_2$, and $p_3$, each of which can be either true or false, then there are eight columns: a column for all three parents being false, a column for $p_1$ being true and $p_2$ and $p_3$ being false, and so on, up to all three parents being true. Each of these eight combinations is a proposition that may influence our belief that $q$ is true or false. Each entry in the matrix contains a weight corresponding to how strongly we believe the truth or falsity of the corresponding proposition should influence our belief in the truth of $q$.

For example, the cell corresponding to column (i.e., proposition) “$p_1$ and $p_2$ true, $p_3$ false,” and row “$q$ true,” contains our estimate of how much the truth of the given proposition should influence our belief in the truth of $q$.

Given the link matrix for node $q$, $P(q)$ can be computed by summing all the possible prior probabilities. Each prior probability is the probability of some combination of true and false for the parent propositions. Hence, each prior probability corresponds to a column of the link matrix. For example, consider the column “$p_1$ and $p_2$ true, $p_3$ false.”

Let $P_1$, $P_2$, and $P_3$ be the probabilities that the propositions $p_1$, $p_2$, and $p_3$ are true. Then the prior probability of the given column (assuming the three parent propositions are independent of each other) is $P_1*P_2*(1-P_3)$. The prior probability is computed similarly for each of the eight columns, e.g., the prior probability for column “$p_1$ false, $p_2$ false,
$p_3 \text{ true}$ is $(1-P_1)^*(1-P_2)^*P_3$, and so on. These eight prior probabilities are summed to compute the total prior probability for $q$. As noted above, if some of the parents are more important than others, a weight for each prior probability appears in the link matrix. Each prior probability is multiplied by its weight from the link matrix before summing. Typically, the weight for each prior probability is a function of the weights assigned to the true parents in the given combination. For example, let $w_1, w_2,$ and $w_3$ be the weights assigned to parent propositions $p_1, p_2,$ and $p_3$ respectively. Then, each of the eight prior probabilities can be weighted by the normalized sum of the weights for those propositions that are true. Hence, the prior probability for column “$p_1$ and $p_2$ true, $p_3$ false” would be multiplied by $(w_1+w_2)/(w_1+w_2+w_3)$, yielding a weighted prior probability of $P_1^*P_2^*(1-P_3)^*(w_1+w_2)/(w_1+w_2+w_3)$. Then, the total prior probability (sum of the prior probabilities for each column of the link matrix) is multiplied by the probability of $q$ given the prior probabilities, $P(q|p_1, p_2, p_3)$.

Note that the link matrix is a conceptual representation. A pure link matrix contains a column for each logical combination of parent nodes. Hence, if the number of parents is large, the number of columns is very large; the number of columns grows exponentially with the number of parents. Specifically, if node $q$ has $n$ parents, its link matrix must (in general) have $2^n$ columns. Correspondingly, an operator represented by such a matrix runs in $O(2^n)$ time; it “requires $O(2^n)$ floating point operations.” Clearly, for large $n$, a more efficient implementation or an operator with a simpler interpretation, must be employed. An example, discussed below in connection with extended boolean operators, is link matrices that satisfy the Parent Indifference Criterion, i.e., matrices in which the value of the child node is determined wholly by the number of parents that are true, not at all by which ones are true. For such matrices, the number of columns clearly grows linearly, not exponentially, with the number of parents. Such operators run in $O(n^2)$ time. In certain important cases (see below), the run in linear time.
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

Document Nodes       Text Representation Nodes       Content Representation Node       Query Concept Node       Information Need, i.e., “Query”

d_1       t_1       c_1       c_2       q_2

d_2       t_2

d_3

...      ...

1

Document Network       Query Network

Figure 1: Example of Document Inference Network.
As a further refinement of the inference network structure, term dependencies, e.g., the tendency of two or more terms to co occur, can be represented by links from one term to another. For example, a link from \( t_1 \) to \( t_2 \) may represent the probability that term \( t_2 \) will occur in a given document, given that term \( t_1 \) occurs in the given document. However, the network specifier must be careful to avoid cycles (see discussion below), e.g., a link from \( t_1 \) to \( t_2 \), a link from \( t_2 \) to \( t_3 \), and then a link from \( t_3 \) to \( t_1 \), would be forbidden! Document dependencies can also be represented, e.g., a citation in document \( D_1 \) to document \( D_2 \). If clustering techniques (see below) are used to establish that a set of documents \( D_1, D_2, \ldots, D_c \) are more similar to each other than to any document not in the cluster by some appropriate criterion, then this can be represented by a cluster node which “represents” the cluster. Cluster nodes can themselves have representation nodes. Thus if document \( D_k \) belongs to cluster \( C_i \), and \( C_i \) has a representation node \( t_j \), then the presence of \( t_j \) in the user’s query will strengthen belief that \( D_k \) satisfies the user’s information need even if \( t_j \) does not appear in \( D_k \) itself.

Once the document network is built, capturing all the dependencies among documents and their representations, we assign probabilities to the nodes. Each document node receives a “prior probability,” generally equal to \( 1/(\text{collection size}) \). This is the probability that a given document will be observed, given that a document is selected at random. “Each representation node contains a specification of the conditional probabilities associated with the node given its set of parent text [document] nodes.” For example, the conditional probability of term node \( t_i \) given parent \( D_j \) is the probability that \( t_i \) is a good descriptor of \( D_j \), i.e., is effective for the purpose of characterizing \( D_j \) and distinguishing \( D_j \) from other documents in the collection. This is specified symbolically as \( P(t_j|D_j=\text{true}) \). Here, “\( t_j \)” is the proposition that “\( t_j \) is a good descriptor of the observed document,” and “\( D_j=\text{true} \)” is the proposition that “\( D_j \) has been observed.” As described above, the probability of the proposition associated with a given term or manually assigned descriptor, can be specified by a link matrix or its equivalent. However, for the special case where the parent events are observations of documents, the general scheme described above for specifying and evaluating a link matrix is modified in one essential respect: Although many documents may contain a given term, \( t_j \), and hence may be parents of \( t_j \)’s representation node, each document \( D_i \) is observed
separately (the observed document is said to be “instantiated”), and hence its probability of satisfying the user’s information need, \( I \), is computed separately. The documents can then be ranked by the probability that they satisfy \( I \). Hence, the link matrix of a representation node \( t_j \) corresponds to observation of a single document containing \( t_i \) and observation of no other documents. There are no columns in \( t_j \)’s link matrix corresponding to several documents observed at the same time. Hence, the link matrix for \( t_j \) contains only two columns: “Document \( D_i \) observed,” i.e., \( P(t_j|D_i=true \text{ and all other parents}=false) \) and “No document observed,” i.e., \( P(t_j|\text{all parents}=false) \). For convenience, these conditional probabilities are usually expressed for short as: \( P(t_j|D_i=true) \) and \( P(t_j|D_i=false) \). This two-column matrix can readily be represented in closed form.

The conditional probability of the proposition “\( t_j \) is a good descriptor” given the event “Document \( D_i \) is observed” can be based on any available evidence. However, as observed in the section on Building Term Vectors, “\( tf*idf \)” is a good statistical measure of the ability of a given term to distinguish a given document. Turtle et al. employ a variant of this measure. However, their variant, \( 0.4 + 0.6*tf*idf \), departs from conventional vector term weighting schemes in one striking respect. The probability (or “belief” as the inference net folks like to call it) is non-zero even when the term frequency for the given document is zero, i.e., even when the term doesn’t occur in the document! The constant component, 0.4 in the above function, is called the “default probability.” It corresponds to the view that a given term may have some probability of being a valid descriptor of the document, even if it is not observed in the document. In other words, \( P(t_j|D_i=false) = 0.4 \). For example, the given term itself may not be present, but a synonym may occur in the document. Or, terms that frequently co-occur with the given term may be present. Or observation may be restricted to the title or abstract, or a summary of the document; the given term may be absent from these surrogates, yet present in the full text of the document. The constant 0.4 was arrived at, not by any deep theory, but purely by experiment. Turtle et al. tried a wide variety of linear functions of the form \( A + B*tf + C*idf + D*tf*idf \), where \( tf \) was normalized by maximum within-document term frequency, and \( idf \) was normalized by collection size. They found that the variant above gave the best results.
It might have been expected that the default probability would be dependent on \textit{idf}, but experiment showed that a constant default performed as well. To complicate matters further, Greiff \textit{et al.} [SIGIR ‘97] developed a computationally tractable, probabilistically motivated soft [extended] boolean operator based on a link matrix. (See below.) They found that to achieve performance comparable to the \textit{p}-norm model (see section on Extended Boolean Approach), they had to set the default probability to zero!

So far, we have discussed that part of the inference network that describes a collection of documents. Not surprisingly, it is called the “document network’ and it is calculated once for a static collection. Naturally, it is updated if the collection it describes is updated. The document nodes are the top or “root” nodes of the document network; the representation nodes are the bottom or “leaf” nodes. (The document network is not a tree since it has multiple roots and a text or representation node can have multiple parents. But it is directed and acyclic, i.e., no “loops,” so it is possible to use graph tree terminology and talk about root, parent, and leaf nodes. For short, a directed acyclic graph is called a “DAG.”) Note: There is a very good reason why loops (cycles) must be avoided in Bayesian inference networks. As Turtle and Croft point out, “evidence attached to any node in the cycle would continually propagate through the network and repeatedly reinforce the original node.”

The other part of the inference network is the “query network.” (See Figure 1.) It too is a DAG. Its multiple roots are the concepts that express the user’s information need. These concept nodes may be the parents of multiple intermediate “query” nodes. Each query node is a parent of the single leaf node representing the user’s information need. The use of those intermediate query nodes allows an information need to be expressed as multiple queries. Naturally, a separate query network is constructed for each user information need. “The single leaf representing the information need [of a given query network] corresponds to the event that an information need is met.”

To apply a given information need to a given document collection, the query network corresponding to the given need must be attached to the document network corresponding to the given collection. This is accomplished (see Figure 1) by specifying parent-child links from the content representation nodes of the document network to the concept nodes of the query network, i.e., the leaf nodes (or at least some
of them) of the document network become parents of the root nodes (at least some of them) of the query network. In the simplest case, a representation node from the document network and a concept node from the query network may be identical, e.g., the former may be a term found in certain documents, and the latter may be a term specified in a user query. (In Figure 1, \( t_m \) is a term in document \( D_1 \) and also a term in the query \( I \). Similarly, term \( t_k \) is a term in both document \( D_2 \) and document \( D_i \), and also a term in the query \( I \).) The link from the one to the other then expresses the relationship that the observation of the given term in a given document contributes evidence to the belief that the document satisfies the given information need. In a more complex case, multiple representation nodes may be parents of a single query concept node which is not identical to any of its parents. This occurs for example, when documents are represented in multiple ways, and queries use “concepts that do not explicitly appear in any document representation.” For example in Figure 1, concept \( c_n \) is a (perhaps manually assigned) concept descriptor of document \( D_i \), and \( t_k \) is a term or phrase occurring in \( D_i \). Both \( c_n \) and \( t_k \) are parents of query concept \( c_v \). Document descriptor concept \( c_n \) might be the concept “information filtering,” query concept \( c_v \) might be the concept “textual filtering,” and document term descriptor \( t_k \) might be the phrase “information filtering” actually occurring in \( D_i \) (and also \( D_2 \)). So, the given structure tells us that both the presence of the phrase “information filtering” in a document (as a string in its text representation) and the presence of the concept “information filtering” (as a manually assigned concept descriptor in its semantic representation), contribute to the belief that the document is about textual filtering, which contributes, in turn, to a belief that the user’s information need \( I \) is satisfied.

Once a query network for a given information need, \( I \), has been attached to a document network for a given collection \( C \), it becomes possible to compute the “belief” (the probability) that the information need has been satisfied by a given document or subset of documents. We must specify an operator or estimation rule, e.g., using a link matrix or its equivalent, for every non-root node of the total filtering inference network, specifying how that node’s probabilities are to be estimated given the probabilities of its parents. Thus we specify how the probability of \( I \) being satisfied depends on its parent query nodes, how the probability of each query node being true depends on the probabilities of its parent concept nodes, how the probability of each query concept
node being true depends on the probabilities of its parent document representation nodes, and so on. If we select some particular document $D_i$, we set $D_i$’s node to true and all other document nodes to false. This “evidence” percolates down through the network as we calculate the probability of each representation node given the evidence of its parent document nodes, the probability of each query concept node given the probabilities of its parent representation nodes, the probability of each query node given the probabilities of its parent concept nodes, and finally the probability (the “belief”) that the user’s information need, $I$, is satisfied, given the probabilities of its parent query nodes. Hence, by selecting some particular document $D_i$, we can compute the probability (the “belief”) that the user’s information need, $I$, is satisfied by $D_i$. We can repeat this process for each document in the collection, thus computing a probabilistic ranking for the documents.

Note that the component terms or concepts that comprise a query can be combined probabilistically, as described above. On the other hand, link matrices can also be used to specify non-probabilistic, e.g., boolean, operators.

For example, a strict boolean AND means that the belief in the truth of node $q$ depends on the truth of all its parents. If, as above, $q$ has three parents, $p_1$, $p_2$ and $p_3$, then the link matrix row for $q = true$ will have a zero for every column except the last one; the last column, “$p_1$ and $p_2$ and $p_3$ true” will contain a one. The row for AND false is, of course, the complement: a one in every column except the last. Hence, the meaning of the link matrix is that $q$ is true if and only if $p_1$, $p_2$, and $p_3$ are all true. Similarly, the $q = true$ row of the link matrix for boolean OR will contain a one for every column except the first column, “$p_1$, $p_2$, and $p_3$ all false,” a zero in that first column. Note: the presence of all ones and zeros in the link matrix for a strict logical OR or AND is a way of saying that the parents are unweighted, i.e., each term in a strict boolean AND or OR is just as important as any other.

Link matrices can even be specified efficiently for soft (also called extended) boolean operators. (See section on Extended Boolean Approach.) [Greiff et al., SIGIR ’97] Instead of specifying a column for each logical combination of parent truth values, Greiff et al. specify a column for each number of true parents, independently of which parents are true. So, if $q$ has four parents, there will be five columns, corresponding to no parents
true, one parent true, two parents true, three parents true, and all four parents true. The weight in the true row for, e.g., three parents true, is an estimate of the probability that \( q \) is true, given that exactly three of its parents are true, any three of its parents. In other words, the weight for the true row of column \( i \) is the conditional probability of \( q \) given \( i \) parents true, \( P(q | i \text{ parents true}) \). Any link matrix of this type is said to satisfy the Parent Indifference Criterion (PIC). Understandably, the cases they explore are those in which the conditional probability is non-decreasing as the number of true parents increases, since the more parents are true, the more support there is for belief in \( q \). Hence, the weights in the true row for \( q \) either remain the same or increase as the number of parents increases, i.e., if \( j > i \) and row 0 = true in link matrix \( m \), then \( m(0, j) = m(0, i) \). As before, the unconditioned probability of \( q \), \( P(q) \), is computed by multiplying the prior probability of each logical combination of parents by its corresponding weight from the link matrix, and summing these products. The difference is that for PIC operators, there is a single weight (and hence a single column in the link matrix), for all logical combinations in which the same number of parents are true. This fact not only reduces the matrix to \( n+1 \) columns; it also makes possible a much more efficient algorithm for operator evaluation. [Greiff, SIGIR '97] The curve of \( P(q) = true \) vs. \# of true parents can be linear or non-linear. Separate matrices must be specified for the soft boolean OR and the soft boolean AND respectively. Commonly, but not necessarily, the parents are terms in the document being evaluated, that are also in the query. The probabilities in the link matrix for the soft boolean OR operator are set so that the curve rises rapidly for a small number of parents, and then increases more slowly; this corresponds to the idea that for a generalization of OR, a small number of query terms present in a given document count a lot toward the total probability that the document is relevant, but additional terms present add a little more. Similarly, for a soft boolean AND operator, the probabilities in the link matrix are set so that the curve increases slowly until most of the parents are true, then increases rapidly; this corresponds to the idea that for a generalization of AND, a small number of query terms present in a given document count a little, but presence of most or all of the query terms add a lot more to the total probability that the document is relevant. If the parents are terms in a given document, then truth of a parent is presence of the term in the given document. The parents can be weighted as before, e.g., using some variant of \( tf*idf \). The child \( q \) can be the query (Information Need in the Turtle et al. terminology), or a concept within the query (see below).
Greiff \textit{et al.} show that PIC operators can be implemented to run in $O(n^2)$ time. Moreover, if the probability vs. number of parents curve is piecewise linear, and “all but one of the pieces of the function is of constant width,” the operator can be evaluated in $O(n)$ time. (Note, by the way, that the strict boolean operators are also defined by PIC matrices. However, for the strict booleans, the link matrix can be represented by a closed form operator, so the PIC algorithm is not necessary.)

The operator at any given non-root node, $N_i$, computes a belief (probability) in $N_i$ in terms of operands which are the beliefs in $N_i$’s parent nodes. A set of operators have been implemented in the INQUERY system that can be specified at the level of the query network, i.e., that are effectively part of the INQUERY query language. They include strict boolean AND, OR, and NOT, extended (soft) boolean operators such as those discussed above, weighted sum (similar to those used for computing the cosine similarity of a document in document vector space), unweighted sum (i.e., mean), and maximum (maximum of operand values). There are also proximity operators which return not a belief but “true” if the proximity condition is satisfied or “false” if it is not satisfied. Proximity operators include unordered text window proximity operators (operands are terms that must occur in any order in a text window of size $\leq n$), and ordered interword proximity operators (operands are terms that must occur in a specified order with interword separation $\leq n$). A proximity value can be converted into a belief value by an operator such as “PHRASE” (i.e., if the operands satisfy the ordered proximity condition with $n = 3$, calculate the unweighted sum of their beliefs).

For example, suppose that the user’s query includes an OR condition (strict or extended), e.g., she is looking for documents that contain $t_1$ or $t_2$ or $t_3$. This is represented, in the query network, by a node $Q$ with three inputs, corresponding to $t_1$, $t_2$, and $t_3$. The $t_1$ input may represent the probability that $t_1$ is a good content descriptor, i.e., a good descriptor for purposes of distinguishing relevance, given that some document $D$ has been observed
in the current collection, and similarly for \( t_2 \), and \( t_3 \). These probabilities may be computed by a widely used ad hoc, statistical measure like \( tf\times idf \) (see section on Building Term Vectors in Document Space), or by a measure based on some theoretical model, e.g., the BI model (see section on Binary Independence Model). Then, the output of the OR node will represent the joint probability that \( t_1 \) OR \( t_2 \) OR \( t_3 \) is a good content descriptor, given the probabilities associated with \( t_1 \), \( t_2 \), and \( t_3 \) separately. If the OR node is a concept node, \( c_1 \), then its output is the probability that the \( c_1 \) is present, given \( D \), i.e., that \( D \) is “about” \( c_1 \). The separate boolean probabilities associated with the ANDs, ORs, NOTs, etc. of which the query is composed are then combined into one total probability for the user’s query, the probability that the user’s information need, expressed as a logical combination of concepts, has been satisfied by the observed document \( D \). This same process can be applied to each document in the collection, thus generating a probabilistic ranking of the documents.

As a simple special case, the OR node or AND node of the preceding paragraph could be the user’s actual query, i.e., there may be no intermediate concept layer. In that case, evaluation of the boolean node, given document \( D \), would give the probability that \( D \) was “about” the user’s query, i.e., satisfied the user’s information need.

On the other hand, the user’s query could be a term vector (either supplied directly by the user) or extracted from a free text specification of the user’s need. If the vector consists of the three terms \( t_1 \), \( t_2 \), and \( t_3 \), then these three terms are parents of the user’s query node. The link matrix for the query, \( Q \), would contain (as before) a column for each of the eight logical combinations of truth value for the three parents. Note that “truth” of \( t_1 \) given \( D \) does not mean that \( t_1 \) is observed in \( D \). If \( t_1 \) is present in \( D \), truth of \( t_1 \) is the proposition that \( t_1 \) is a good descriptor of \( D \) (or in other words, contributes evidence that \( D \) is relevant to queries containing \( t_1 \)). The value of \( t_1 \) is a measure of the probability that \( t_1 \) is indeed a good descriptor, i.e., some measure of its “goodness.”

The corresponding value of the proposition that \( t_1 \) is false is then \( 1-P(t_1 = true|D) \). On the other hand, if \( t_1 \) is not present in \( D \) at all, the value of the proposition “\( t_1 \) is a good descriptor” is zero, because \( D \) offers no support for the proposition. Correspondingly, the value of the proposition “\( t_1 \) is not a good descriptor is 1-0 = 1. (But note that if the concept of a default probability, discussed above, is employed, then the probability that \( t_1 \)
is a good descriptor, given no document support, is not zero; using Turtle’s function, given above, its value = 0.4.)

One essential virtue of the inference network approach is that it allows one to represent a complex set of dependencies, dependencies in a document collection, dependencies in a complex information need specification, and dependencies between the concepts that represent the document collection and the concepts used to express the information need. The belief that a given information need is satisfied by a given document or set of documents in the given collection can then be estimated by evaluating the operator at each non root node of the network. Different sources of evidence (automatically extracted terms, phrases, paragraphs, and manually assigned descriptors), can be combined. Different query types, e.g., natural language and Boolean, “can be combined in a consistent probabilistic framework. This type of ‘data fusion’ has been known to be effective in the information filtering context for a number of years.” [Callan et al., IP&M, 1995] (Note: This “data fusion” is fusion of document representations and evidence, and fusion of queries. The next section deals with fusion of results, i.e., fusion of retrieved documents from different sources or different queries.) Last but not least, inference net evaluation does not require a complex closed form expression that captures all the dependencies. Instead, the logic of evaluation is spread over the network. However, the one problem that neither inference networks nor any other probabilistic representation can solve is the difficulty of ever knowing/estimating the dependencies and prior probabilities in a complex relationship among documents and queries.

2.4.4.3 Logical Imaging

Another approach to computing posterior probabilities, based on “non-classical logics,” is described by Crestani and van Rijsbergen [SIGIR ’95]. Given a document $D_i$, the key to “Logical Imaging” is to determine for every term $t_k$ that indexes the given document collection and is not in $D_i$, the term in $D_i$ that is “most similar” to $t_k$. Then posterior term probabilities for $D_i$ are computed by transferring the prior probability of each term not in $D_i$ to its most similar term in $D_i$. The transfer is additive, i.e., if terms $t_j, t_k,$ and $t_l$ are terms not in $D_i$, and all three are “most similar” to $D_i$ term $t_{ij}$, then we add the prior probabilities of $t_j, t_k,$ and $t_l$ to the prior probability of $t_{ij}$ to obtain its posterior probability. Probability is neither “created” nor “destroyed” in this process, merely
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

transferred from non document terms to document terms. Hence, if the prior probabilities for the entire term space sum to one, the posterior probabilities for a given document must also sum to one. The sum of posterior probabilities of document terms that are also in a given query then becomes the probability of relevance for the given document relative to the given query.

The obvious question is how to compute term “similarity” in order to compute for each term $t_i$ the degree of similarity of every other term. (Given this ranking, it is then straightforward to determine for any term $t_k$ that is not in a given document $D_i$, the term in $D_i$ that is “most similar” to $t_k$.) The measure chosen by Crestani and van Rijsbergen is the “Expected Mutual Information Measure” (EMIM) between terms. As applied to IF, “[t]he EMIM between two terms is often interpreted as a measure of the statistical information contained in the first term about the other one (or vice versa, being a symmetric measure.” [Crestani & van Rijsbergen, SIGIR ‘95]. Let $T_i$ be a variable that takes on two values: $T_i=1$ means that term $t_i$ occurs in a randomly chosen document and $T_i=0$ means that it does not. Similarly $T_j$ is a variable for the occurrence or non occurrence of another term $t_j$. $P(T_i)$ is the probability of event $T_i$, $P(T_j)$ is the probability of event $T_j$ and $P(T_i, T_j)$ is the joint probability of events $T_i$ and $T_j$. Then the EMIM is:

$$I(t_i, t_j) = \sum_{T_i, T_j} P(T_i, T_j) \log \frac{P(T_i, T_j)}{P(T_i)P(T_j)}$$

where the summation is over the four possible values of $T_i$ and $T_j$ together, i.e., $T_i$ occurs and $T_j$ does not, $T_j$ occurs and $T_i$ does not, they both occur, or neither occurs, in a given document. Note that if $T_i$ and $T_j$ occur independently, $P(T_i, T_j) = P(T_i)P(T_j)$ and $I(T_i, T_j) = 0$, so the EMIM is a measure of how much the two events, in this case co occurrence of $T_i$ and $T_j$ in a document, departs from stochastic independence. [van Rijsbergen, 1979]

Crestani and van Rijsbergen also describe an extension of Logical Imaging called “General Logical Imaging.” The difference is that instead of identifying the term $t_l$ in $D_i$ most similar to a given term $t_k$ not in $D_i$, one identifies the set of terms, e.g., $t_p, t_m, t_n$, in $D_i$ most similar to $t_k$. Then one transfers the prior probability of $t_k$ to the set $t_p, t_m, t_n$ according to a transfer function (called an “opinionated probability function”) which
prescribes how much of \( t_k \)'s prior probability is transferred to the most similar term \( t_l \), how much to the next most similar term, \( t_m \), and so on.

2.4.4.4 Logistic Regression

Cooper et al. [TREC, 1994] [SIGIR '92], use “a probabilistic model … to deduce the general form that the document-ranking equation should take, after which regression analysis is applied to obtain empirically-based values for the constants that appear in the equation … The probabilistic model is derived from a statistical assumption of ‘linked dependence.’” (See discussion of Linked Dependence in earlier section.) Logistic regression is the regression analysis method employed. That is, the “probability of relevance vs. document evidence” curve is fitted to a logistic regression function. The values of the “empirically-based” constants are derived from a training set.

Cooper et al. explain the reasons why logistic regression is more appropriate than standard (non-logistic) linear regression for predicting the probability of relevance. The most important reason is that the probability of relevance to be modeled is two-valued, i.e., every document in the training set is either known to be relevant (probability of relevance equals one), or known to be non-relevant (probability of relevance equals zero). Hence, the desired probability curve must fit a set of data points all of which reside either on the horizontal \( p=0 \) line, or the horizontal \( p=1 \) line. Clearly, the sloping straight line generated by linear regression will fit these points very poorly for any possible slope. On the other hand, logistic regression generates an “S-shaped” curve. With appropriate parameters, the lower arm of the “S” can be made to approximate \( p=0 \), while the upper arm can be made to approximate \( p=1 \).

To obtain this “S-shaped” curve, we express the probability of relevance as a logistic regression (“logit”) function, as follows:

\[
P(R \mid X_1, \ldots, X_M) = \frac{e^{c_1 \cdot X_1 + \ldots + c_M \cdot X_M}}{1 + e^{c_1 \cdot X_1 + \ldots + c_M \cdot X_M}}
\]

where the \( X_j \) are individual facts about a given document. Cooper et al. group the facts (the “evidence”) from a given query/document pair into “composite” clues, \( A_i \) (\( i = 1 \) to \( N \)), where each composite clue is composed of a set of facts \( X_j \) (\( j = 1 \) to \( M \)). For example, “if … \( A_i \) is a word stem, then \( X_j \) might be (say) the relative frequency of the stem in the
query, $X_2$ its relative frequency in the document, and $X_3$ its inverse document frequency in the collection.” Hence, in contrast to vector space methods, and most other probabilistic approaches (except inference networks), which describe each document with a single level of features, a “feature vector,” Cooper et al. model each document and query more accurately as two levels of feature: At the outer level, each document and query is described by a conventional feature vector of “composite” features, $A_i$. At the second (inner) level, each composite feature is expanded into a set of elementary features, $X_i$. Each composite feature, $A_i$, is related to its elementary features by logistic regression. The probability of relevance of the document as a whole is related to its composite features, the $A_i$, by the traditional linked dependence assumption.

The calculation turns out to be easier in terms of the log of the “odds” of relevance. The “odds” of relevance is defined as the ratio of the probability of relevance to the probability of non-relevance. Hence, the odds that a document is relevant, given a single composite feature, $A_i$, can be expressed in terms of its corresponding elementary features, as:

$$O(R | A_i) = \frac{P(R | X_1, \ldots, X_M)}{P(\neg R | X_1, \ldots, X_M)}$$

If one replaces the probability by the corresponding logistic regression function in the above identity, and takes the natural log of both sides, one obtains:

$$\log O(R | A_i) = \log \frac{P(R | X_1, \ldots, X_M)}{P(\neg R | X_1, \ldots, X_M)} = c_0 + c_1 X_1 + \ldots + c_M X_M$$

This equation gives the logodds of relevance given a single composite clue, $A_i$. To extend it to a set of $N$ clues, $A_1, \ldots, A_N$, one uses the linked dependence assumption, discussed above in an earlier section. This assumption is that the same degree of dependence holds for both the relevant document set and the non-relevant document set. This degree of dependence is expressed (crudely) as a common proportionality constant. Hence in symbols, we have:
CHAPTER 2: APPROACHES TO INFORMATION FILTERING

\[ P(A \cap B | R) = K \ P(A | R) \cdot P(B | R) \]

and

\[ P(A \cup B | \neg R) = K \ P(A | \neg R) \cdot P(B | \neg R) \]

Note that linked dependence, like pure binary independence, breaks a complex joint probability into a product of simpler separate probabilities for the individual “composite” clues. Cooper et al. point out that since each clue, each piece of evidence, is a separate factor in the linked dependence formulation, and hence makes a separate contribution to the total probability of relevance, the effect is that the probability of relevance computed for high-ranking documents, documents containing many clues matching the given query, will tend to be too high. They offer a couple of relatively crude methods of compensating for this effect, discussed below.

Dividing the second linked dependence equation by the first (which causes the proportionality constant \( K \) to cancel out), and using the identity,

\[
\frac{P(A | R)}{P(A | \neg R)} = \frac{O(R | A)}{O(A)}
\]

we have:

\[
\frac{O(R | (A, B))}{O(R)} = \frac{O(R | A)}{O(A)} \cdot \frac{O(R | B)}{O(B)}
\]

or, generalizing to \( N \) composite clues, multiplying by \( O(R) \), and taking the log of both sides, we have:

\[
\log O(R | A_1, A_2, \ldots, A_N) = \log O(R) + \sum_{i=1}^{N} \left[ \log O(R | A_i) - \log O(R) \right]
\]

where the \( A_i \) are the “composite features” to be used to characterize any randomly chosen query-document pair, \( O(R | A_i) \) is the odds that the document is relevant to the query given the composite feature \( A_i \), \( O(R) \) is the odds that the document is relevant to the query in the absence of any evidence, and \( O(R | A_1, A_2, \ldots, A_n) \) is the odds that a document is relevant given all the composite features, \( A_1, A_2, \ldots, A_n \). The sum in the above equation is taken over all the clues, i.e., from \( i = 1 \) to \( i = N \). Note that the composite
features employed are those that match in at least one query-document pair in the training set. This is the “fundamental equation” employed by Cooper et al. for document ranking by logistic regression. Note that the regression is “logistic” in the sense that we express the probability of relevance as a logit function. The actual regression performed (see below) is linear regression.

If all the composite features, $A_i$, for a query/document pair are of the same type, i.e., defined in terms of the same independent variables, $X_j$, then the set of composite features forms a matrix with $M + 1$ columns. For any query-document pair in the training set, there will be a set of rows in the matrix with one row for each composite feature, e.g., each word stem, that occurs in both query and document. (These are called “match terms.”) Hence, each row in the training set corresponds to a query-document-term “triple.” The row for a given composite feature of a given query-document pair will contain the $M$ values of the elementary features, $X_j$, comprising that composite feature, e.g., the relative (normalized) frequency of the term in the given document, the relative frequency of the term in the query, etc. The given row will also contain one additional value: the logodds of relevance given the presence of $A_i$, \( \log O(R/A_i) \). This value is computed from the training set, by observing the proportion of documents relevant to the given query (as judged by the humans who constructed the training set) that contain $A_i$, the proportion of documents not relevant to the given query that contain $A_i$, dividing the former by the latter, and taking the log of this quotient. This value is the \((M+1)\)th value in the row for the given query-document-composite clue triple. The entire set of such rows in the training set forms a matrix. If this training set matrix, or a matrix composed of a representative sample of rows drawn from the training set, is submitted to a statistical program package capable of performing ordinary linear regression, the package will compute values for the coefficients $c_0, c_1, \ldots, c_m$. Cooper et al. argue that the resulting linear function of the $X_j$ with the $c_j$ computed by the regression program will be a better predictor of \( \log O(R/A_i) \) than direct computation from the training set. Note that each coefficient, $c_j$, is the coefficient applied to any value of elementary clue type, $X_j$, e.g., if $X_j$ is relative frequency of a given match term in a given document, then $c_j$ is the coefficient applied to the relative frequency of any term in any document for purposes of computing its logodds of relevance to any query containing the same term.
At filtering time, the system is given a new query $Q$ against an operational document collection it is hoped will have similar statistical properties to the training set. Given any document $D$ from this operational collection, the filtering system can estimate $\log O(R|A_i)$ of $D$ for each of the $A_i$ using the coefficients computed from the training set (as described above), and the values of the $X_i$ obtained from the given document and query. The $\log O(R)$ can be estimated straightforwardly from the proportion of relevant documents in the training set. Summing the $\log O(R|A_i)$-$\log O(R)$ contributions, the system obtains from the above equation an estimate for $\log O(R|A_1, A_2, \ldots, A_n)$ for the given document. These logodds estimates can be used to relevance rank the retrieved documents relative to the given query. Moreover, a logodds estimate can be converted into a probability of relevance for the benefit of the end user.

In a subsequent TREC experiment [TREC-2], Cooper et al. employ a variant of the method described above. They call the earlier method the “triples-then-pairs” approach. The term “triples” refers to the fact that coefficients are computed separately for each of the composite clues, the $A_i$; each of the rows in the training set used to compute the coefficients for a given $A_i$ corresponds to a query-document-composite clue triple, i.e., to the values of the elementary clues, the $X_i$, for a given $A_i$, obtained from a given query and document. By contrast, Cooper et al. call their TREC version the “pairs only” approach. A single linear regression is applied to a set of rows derived from the entire training set. Each row corresponds to a single query-document pair; hence, each row contains the $M$ elementary clues, $X_i$, for each of the $N$ composite clues, $A_i$. Therefore, the elementary clues have a double index, $X_{n,m}$, where $n$ indexes the composite clue, and $m$ indexes the elementary clues for a given composite clue. Hence, instead of obtaining coefficients for computing the logodds of relevance given a single $A_i$ (as in the earlier “triples” approach), Cooper et al. obtain at once the coefficients for computing the logodds of relevance of any given document to any given query given all the $A_i$. In other words, they apply regression to obtain the coefficients for:

$$\log O(R|A_1, A_2, \ldots, A_n) = c_0 + c_1 \sum_{n=1}^{N} X_{n,1} + \ldots + c_m \sum_{n=1}^{N} X_{n,M}$$

As noted earlier, the effect of the “linked dependence” assumption (or the even stronger binary independence assumption) is to overstate the probability of relevance as
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

\(N\), the number of composite clues in the query that match the document being ranked, increases. In other words, the effect is to overstate the probability of relevance for high-ranking documents. In their TREC-2 work, Cooper et al. compensate for this by multiplying each of the \(X_{n,m}\) terms by a “function \(f(N)\) that drops gently with increasing \(N\), say \(1/\sqrt{N}\) or \(1/(1 + \log N)\).”

For the TREC2 experiment, three “composite” clues were employed: (1) normalized word stem frequency in query, (2) normalized word stem frequency in document, and (3) normalized word stem frequency in collection. Note that, whereas for the “triples-then-pairs” approach, each composite clue might be, e.g., a stemmed match word, and the facts comprising the composite might be its various (in this case, three) frequencies, in the “pairs only” approach, each composite clue is a type of frequency, and the facts comprising a given composite are the values of the given frequency for each of the \(M\) match word stems. For word stem \(m\), these were defined in TREC2 as follows:

1. \(X_{m,1} = \frac{\text{number of times the } m\text{th word stem occurs in query}}{\text{(total number of all stem occurrences in query + 35)}}\);
2. \(X_{m,2} = \frac{\text{number of times the } m\text{th word stem occurs in the document}}{\text{(total number of all stem occurrences in document + 80)}}\);
3. \(X_{m,3} = \log(\text{number of times the } m\text{th word stem occurs in the collection})\), divided by the total number of all stem occurrences in the collection).

Note that Cooper et al. assume above (in both the “triples” and “pairs only” methods) that a training set is available for a sample of “typical” queries, but not for the actual future queries for which filtering is to be performed. Hence, regression coefficients, the \(c_i\) above, are computed for the elementary clues associated with each composite match term \(A_i\), e.g., each word stem that matches in at least one query-document pair of the training set. When a document \(D\) is to be ranked against a new query, \(Q\), the \(A_i\) employed are those which occur in both \(Q\) and \(D\), i.e., they are match terms for the particular query-document pair being evaluated, not the complete set of match terms for the training set as a whole.

(On the other hand, \(Q\) and \(D\) might contain a match term that never occurred in the training set. Such a match term could not be used in computing the logodds of relevance of \(D\) to \(Q\), because the coefficients \(c_j\) would not be known for this “new” \(A_j\).) That is, the
predictors whose values are used to compute the logodds of relevance of a document $D$ to a new query $Q$ include the elementary clues, the $X_{n,m}$, for every term, e.g., every word stem, that occurs in both $Q$ and $D$, and is also a match term of the training set. In the “triples” method, the value of each of those elementary clues, e.g., the relative frequency of a given word stem in $D$ (that also occurs in $Q$), is multiplied by the coefficient for that type of elementary clue, as computed from the training set. Then these products are summed to compute the logodds of relevance for the given $A_i$, e.g., the given word stem. These logodds values are then summed to compute the value of the logodds of relevance of $D$ to $Q$, given all the matching terms. In the “pairs only” method, the values of all the elementary clues of a given type, e.g., the relative frequencies of all match terms for $Q$ and $D$ are summed first, and then multiplied by the coefficient for that clue type.

By contrast, if training data is available for the actual queries to be evaluated operationally (which is commonly the case in routing applications), then, given one of those actual queries, $Q$, Cooper et al. [TREC-2] develop an equation that predicts the logodds of relevance of any given document $D$ to $Q$. This equation is derived, as before, from their “fundamental equation.” However now, each of the $A_i$ corresponds to presence or absence of the $i$th filtering clue, e.g., the $i$th term of $Q$, in $D$. Note that the odds of relevance for each term $A_i$ in $Q$ can now be estimated directly from the training set. If $A_i$ is in $D$, one computes the ratio of relevant to non-relevant documents containing $A_i$. Similarly, If $A_i$ is not in $D$, one computes the ratio of relevant to non-relevant documents not containing $A_i$. Hence, the composite clues $A_i$ need not be expressed in terms of the elementary clues, as in the earlier cases. However, it may be that, in addition to the query-specific training set, there is also a larger non-specific training set for “typical” queries as before. In that case, the logodds of relevance of $Q$ to $D$ can be expressed as a linear combination of a query-specific function of all the $A_i$ in $Q$, and a non-specific predictive function of the $A_i$ in $Q$ that are also in $D$, obtained by the earlier method where the queries to be evaluated operationally are not known. The non-specific component will be expressed in terms of elementary clues, as before. How should the query-specific and non-specific components be weighted? According to Cooper et al. [TREC-2], this remains a subject for research.
2.4.4.5 Okapi (Term Weighting Based on Two-Poisson Model)

Robertson et al. [SIGIR ’94] has developed a term weighting scheme based on the Poisson distribution. This scheme was first presented in the City University of London Okapi system. As it has proved to be one of the most successful weighting schemes in TREC competitions, it has been adopted by other TREC participants, and is generally identified by the system in which it was introduced, as Okapi weighting.

The Okapi approach starts with the view of a document as a random stream of term occurrences. Each term occurrence is a binary event with respect to a given term $t$. That is, there is a (typically small) probability $p$ that the event will be an occurrence of $t$, and a probability $q = 1-p$ that the event is not an occurrence of $t$. Then, the probability of $x$ occurrences (commonly called “successes”) of $t$ in $n$ terms is given by the binomial distribution. [Hoel, 1971] For very small $p$ and very large $n$, the binomial distribution is well approximated by the Poisson distribution:

$$p(x) = e^{-m} \frac{m^x}{x!}$$

where $m$ is the mean of the distribution. To incorporate within-document term frequency $tf$, Robertson makes the fundamental assumption that the term frequency of a given term $t$ is also given by a Poisson distribution, but that the mean of this distribution is different depending on whether the document is “about” $t$ or not. It is assumed that each term $t$ represents some “concept,” and that any document in which $t$ occurs can be said to be either about or not about the given concept. Documents that are about $t$ are said to be “elite” for $t$. Hence, Robertson assumes that there are two Poisson distributions for a given term $t$, one for the set of documents that are elite for $t$, the other for documents that are not elite for $t$. (This is why the Okapi weighting is said to be a 2-Poisson model.) The Poisson distribution for a given term $t$ becomes:

$$p(tf) = e^{-m} \frac{m^{tf}}{(tf)!}$$

where $m$, the mean of the distribution, is either $\mu$ or $\lambda$ depending on whether the distribution is for documents elite for $t$ (mean = $\mu$), or documents that are not elite for $t$ (mean = $\lambda$). Note that these two Poisson distributions give the probability of a given term frequency for a given term $t$ in terms of document eliteness to $t$, not in terms of relevance.
to a given query. A query can contain multiple terms. A document contains many
terms, and may be about multiple concepts. The usual assumptions about term
independence, or Cooper’s “linked dependence,” are extended to eliteness; that is the
eliteness properties of any term $t_i$ are assumed to be independent of those for any other
term $t_j$.

Robertson defines the weight $w$ for a given term $t$ in terms of a logodds function:

$$w = \log \frac{p_{tf}q_0}{q_{tf}p_0}$$

where $p_{tf}$ is the probability of $t$ being present with frequency $tf$ given that the document is
relevant to a given query, and $q_{tf}$ is the probability of $t$ being present with frequency $tf$
given that the document is non-relevant to the given query. The $p_0$ and $q_0$ are the
corresponding probabilities with $t$ absent. Hence, $P_{tf}/P_0$ is not the odds of $t$ being
present in a relevant document as before, but the odds of $t$ being present with a given $tf$
as compared to not being present in a relevant document at all. (And similarly, for $q_{tf}/q_0$
with respect to non-relevant documents.) When the Poisson distributions of $t$ relative to
document eliteness/non-eliteness given above are incorporated into this logodds
function of $t$ relative to document relevance/non-relevance, the result is a rather complex
function in terms of four difficult-to-estimate variables: $p'$, $q'$, $\mu$ and $\lambda$. Here, $p'$ is the
probability that a given document is elite for $t$, given that it is relevant, i.e., $P$ (document
elite for $t \mid R$). Similarly, $q' = P$ (document elite for $t \mid \text{not } R$).

Robertson converts this difficult-to-compute term weight function into a more practical
function. His basic strategy is to replace complex functions by much simpler functions
of term frequency that have approximately the same shape, e.g., the same behavior at
$tf=0$, the same behavior as $tf$ increases, and grows large, etc. His approximation starts
with the traditional logodds function for presence/absence of $t$, as derived from the
relevance/non-relevance contingency table in 2.4.4.1 (Binary independence). This is
multiplied (in effect, “corrected” or “improved”) by a simple approximation function
for term weight in a document as a function of $tf$, a function that approximates the shape
of the true 2-Poisson function. The approximation contains a “tuning constant,” $k_1$, in
the denominator, whose value (determined by experimentation) influences the shape of
the curve. Then, the weight function is multiplied by a similar approximation function for the query, i.e., a function of within-query term frequency, $qtf$. This function also contains a tuning constant, $k_3$.

To improve the approximation further, Robertson takes document length into account. He offers two broad hypotheses to account for variation in document length: The “Verbosity hypothesis” is the hypothesis that longer documents simply cover the same material as corresponding shorter documents but with more words, or (more fairly) covers the same topic in more detail. (This is the hypothesis that underlies most document vector normalization schemes discussed above.) The “Scope hypothesis” is the hypothesis that longer documents deal with more topics than shorter documents. (This is the hypothesis that underlies most work with document “passages.”)

Obviously, each hypothesis can be correct in some cases, and indeed, in other cases, both hypotheses may be correct, i.e., a document may be longer than another both because it uses more words to discuss a given topic, and because it discusses a greater number of topics. Hence, Robertson refines his approximation to allow the user to take either or both hypotheses into account, as appropriate. First, on the basis of the Verbosity hypothesis, he wants the weight function to be independent of document length. On the simple common assumption that term frequency is proportional to document length, he multiplies $k_1$ by $dl_i$, the length of the $i$th document, the document $D_i$ under consideration, so that all terms will increase proportionally with document length, and the weight function will remain unchanged. Then, on the assumption that the value of $k_1$ has been chosen for the average document, he further normalizes $k_1$, dividing it by $dl_{avg}$ the average document length for the collection under consideration. Then, he modifies this normalization factor with another tuning constant, $b$, into a composite constant $K = k_1((1-b) + b(dl_i/dl_{avg}))$. The constant, $b$, also determined by experiment, controls the extent to which Verbosity hypothesis applies ($b=1$) or does not apply ($b=0$).

To compute document-query similarity for a given document, $D_i$, the term weights determined by the above approximation function are added together for all query terms that match terms in $D_i$. Finally, to this sum Robertson adds a “global correction term” that depends only on the terms in the query, and not at all on whether they match terms in $D_i$. This correction term reflects the influence of document length variation, departure
from the average length, with respect to the weight of each query term. The correction term contain yet another tuning constant, $k_2$.

The final result, first used in TREC3 [Robertson et al., 1995] and TREC4 [Robertson et al., 1996], is called BM25; the BM stands for “Best Match” and the 25 is the version number, reflecting the evolution of this term weighting scheme. The BM25 function for computing the similarity between a query $Q$, and a document $D_i$ is:

$$S(Q, D) = \sum_{t \in Q} \left( \log \frac{(r + 0.5)/(R - r + 0.5)}{(n - r + 0.5)/(N - n - R + r + 0.5)} \right) \cdot \frac{(k_1 + 1) qtf}{K + tf} \cdot \frac{(k_3 + 1) qtf}{k_3 + qtf} + k_2 \frac{|Q|}{avdl - dl} \frac{avdl - dl}{avdl + dl}$$

where

Summation is over all terms $t$ in query $Q$

$r =$ number of documents relevant to $Q$ containing term $t$

$R =$ number of documents relevant to $Q$

$n =$ number of documents containing $t$

$N =$ number of documents in the given collection

$tf =$ frequency (number of occurrences) of $t$ in $D_i$

$qtf =$ frequency of $t$ in $Q$

$avdl =$ average document length in the given collection

$dl =$ length of $D_i$, e.g., the number of terms, or the number of indexed terms, in $D_i$

$|Q| =$ number of terms in $Q$

$k_1, k_2, k_3,$ and $K$ are tuning constants as described above.

$K = k_f ((1-b) + b(dl/dl_{avg}))$ where $b$ is another tuning parameter.

Varieties of Okapi BM25 have continued to be used down through TREC-9, both by its originators [Robertson et al., 2000] and others, due to its effectiveness. According to Robertson et al., “$k_f$ and $b$ default to 1.2 and 0.75 respectively, but smaller values of $b$ are sometimes advantageous; in long queries $k_3$ is often set to 7 or 1000 (effectively infinite).” $k_2$ has often been set to 0, e.g., in TREC-4 and TREC-9.
2.5 Routing/Classification Approaches

In theory, the “routing” or “classification” problem is identical to the information filtering problem: to identify documents that match, i.e., are relevant to, a specified query or information need. Hence, in principle the same methods are applicable to both problems. [Belkin & Croft, CACM, 1992] However, the practical differences between the two problems affect which methods are practical for each.

In information filtering, the user has at any given time one or more relatively static collections. The collections may be updated, but not so rapidly as to change their basic, e.g., statistical or semantic, properties overnight. New collections may come online, but they will have the same relatively static characteristics as the existing collections. The user generates many queries against these collections. In other words, the collections are relatively static; the queries are not.

In the classification environment, there is no fixed collection. Instead, there is a steady (perhaps high volume) stream of incoming documents. There is a well-defined set of topics of interest, or a well-defined set of users, each with his own well-defined set of interests and concerns. The problem is to classify each document according to which topic(s) it is “about” or which user(s) the document would interest, and then route the document to the appropriate “bin(s).” Documents that are not about any topic of interest are thrown away. The set of documents to be classified and routed is not static at all; rather it is constantly changing. Moreover, these documents are not available initially for the purposes of studying their statistical or other properties. Rather, they will arrive over a (perhaps long) period of time. On the other hand, the set of user needs are presumed to be (relatively) stable [Belkin & Croft, CACM, 1992], although new needs and users will arrive over time, and old needs will become obsolete. Hence, the queries/information needs are relatively static; the document population is not.

The term “routing” is often applied to a classification system in which there is only a single topic of interest. Hence, the objective is to distinguish and pass on all documents relevant to the given topic, and discard all other documents. In that case, the router is often called a “filter”. Filtering can be “negative” as well as positive, i.e., the purpose of a user’s “profile” may be to specify “junk” that he wants to throw away. [Belkin & Croft, CACM, 1992]
Of course, this distinction between routing and information filtering is an idealization. In practice, the distinction is not necessarily so clear cut. The collections to which information filtering is applied may not be as static as one would wish. The information needs in the routing application may also change rapidly. Routing and information filtering may be viewed as opposite ends of a spectrum with many actual applications in between.

In the routing application, one does not start with a large static collection of actual documents to which classification and routing are to be applied. Hence, it is common to employ a “training set” of documents which are (it is hoped) statistically typical of the documents to be encountered in practice. The routing system is trained against these documents. Training a system against a “training set” is analogous to expanding or refining a query with relevance feedback. In the latter case, the retrieved documents judged relevant are, in effect, the “training set”. In the former case, the user supplies relevance judgments for the documents in the training set, e.g., document $D_1$ is relevant to class $C_1$, document $D_2$ is relevant to classes $C_1$ and $C_2$, $D_3$ is non-relevant, etc. The effect of training is to build a query or set of queries that classify the incoming documents correctly. Hence, the desired effect is that the query for class $C$, will match or rank documents according to their degree of relevance to class $C$.

The biggest practical difference between routing and information filtering is that in routing, the training, i.e., the query expansion and refinement, are performed in advance, i.e., before the system goes “operational.” Hence, computationally expensive and time consuming training methods that would not be acceptable in real time become practical. (In short, routing permits preprocessing, perhaps slow, of the relatively static queries. In contrast, ad hoc querying permits preprocessing, perhaps slow, of the relatively static document collections.) The essential requirement is that once the system has been trained, the amount of time required to perform the classification at run time is moderate. Moreover, even at run time, the user is not sitting at his screen waiting for a response to a query he has just issued. Hence, response time may not be as critical an issue as in information filtering. However, the volume of documents that a routing application must classify may be very large in some applications. In such cases, it may be impossible for the system to “keep up” with the traffic volume unless the run-time classification algorithm is very fast.
Even during a pre-operational training phase, a very large “feature space” can present problems. As noted above, classification or learning algorithms break down if the number of features required for classification, the number of dimensions of the feature space, is very large. Hull [SIGIR ‘94] notes that “[i]f there are too many predictor variables [i.e., features used to classify the documents], the classifier will overfit the training data … there must be significantly fewer predictor variables than relevant documents before it is possible to obtain good estimates of the parameters in the classification model.” Similarly, Schutze et al. [SIGIR ‘95] consider “classification techniques which have decision rules that are derived via explicit error minimization … Error minimization is difficult in high-dimensional feature spaces because the convergence process is slow and the models are prone to overfitting.” In particular, if the feature space consists of all significant terms in a given collection, or even in the relevant documents of a collection, the number of features will certainly be far too large. Hence, even though the training is performed “offline,” i.e., before classifying the actual traffic, methods to reduce the number of features are essential. Note that overfitting is the same problem that Buckley and Salton [SIGIR ‘95] encountered when they used relevance feedback to expand queries, thereby increasing greatly the number of query terms, i.e., “features.” They developed their technique of Dynamic Feedback Optimization to avoid that problem (see above).

Two methods [Schutze et al., SIGIR ‘95] have been used to reduce the dimensionality of the feature space: reparameterization (which replaces the original feature space by a lower dimension feature space derived from the original features), and feature selection (which selects from the complete set of features a small subset of the “best” features, i.e., the ones most likely to distinguish relevant from non-relevant documents). A popular method of reparameterization (LSI) is discussed above. Methods of feature selection (more specifically, term selection but the methods are generalizable to other statistical features) are discussed above in the section on query expansion.

Another problem with large feature spaces or computationally expensive classification algorithms is that the “real world” that generates the documents to be classified may change rapidly, resulting in a document population with rapidly changing statistical characteristics. Hence, it may be necessary to retrain the system frequently, which becomes unacceptably expensive if the training algorithm is very slow or requires
enormous computational resources. This issue seems to have been largely ignored in the literature, which generally assumes that user needs and the statistical characteristics of the document population to be classified are stable.

If the population of incoming documents is completely stable, then the training set (if it is a representative sample of that future population) is sufficient to train the classifier. If the training set is imperfect (or non-existent), relevance feedback can be applied to the accumulating collection of documents. [Buckley et al., SIGIR ‘94] Documents that have satisfied a user can become, over time, a very large and effective training set for that user or for a given topic. [Belkin & Croft, CACM, 1992] “Over the life of the query [i.e., a longstanding information need], thousands of documents could conceivably be returned to the user for relevance judgments.” [Buckley et al., SIGIR ‘94]

On the other hand, Lewis [SIGIR ‘95] is one of the few to consider the case where “the classifier is applied to time-varying data such as news feeds or electronic mail.” In such a case, relevance feedback (i.e., query refinement and expansion based on user identification of those documents retrieved by the original query which are relevant to her need see below for a more detailed discussion) can be applied to the incoming documents to update the query, enabling it to track a population of incoming documents whose statistical characteristics are (slowly) changing. (Buckley et al. consider the application of relevance feedback to a routing application, but they do not specifically address the issue of changing statistical properties; they assume that the purpose of continuing feedback is to enable the classifier to approach more closely a fixed target.) Lewis does consider time-varying data but he explicitly does not consider relevance feedback. Instead, he considers the case where the classifiers are “autonomous” systems, i.e., there is little feedback from end users, and hence the systems must estimate their own effectiveness and re optimize themselves as the incoming data changes. He assumes probabilistic classifiers, i.e., classifiers that, given a document, output both a classification and a probability that the document belongs in that classification, given the set of features being used for classification. His method is to specify an “effectiveness” measure for the classifier as a function of the classifications and associated probabilities assigned to $N$ previous documents. The classifier then retunes itself to maximize the effectiveness measure. Lewis studies three possible effectiveness measures.
Yu et al. [CIKM 98] address the issue of routing a stream of (possibly) time-varying data, in an interactive environment where each incoming document judged relevant by the system is presented to the user for a possible relevance judgment. Their “adaptive text filtering” algorithm maintains a pool of terms, $Pool_R$, that have occurred in documents judged relevant by the user, and another pool of terms, $Pool_N$, that have only occurred in documents judged non-relevant. (In other words, a term that has occurred in relevant documents will be in $Pool_R$, regardless of whether it has also appeared in non-relevant documents. By contrast, a term will be in $Pool_N$ only if it has appeared in non-relevant documents, and has never appeared in relevant documents.) An incoming document is retrieved and presented to the user as possibly relevant under two conditions: (1) The document is retrieved if the sum of the weights of terms in the document that are also in $Pool_R$ exceeds a specified threshold. The weight of a term in this calculation is actually the sum of two weights, its feature weight, based on all the documents in which it has occurred so far that have been presented to the user and judged relevant, and its document weight, its weight in the current document computed by the traditional $tf*idf$ function. (2) The document is also retrieved if the proportion of new terms in its document vector plus the proportion of terms in the document vector that are also in $Pool_R$ exceeds a specified threshold. A new term is a term that is neither in $Pool_R$ or $Pool_N$. In other words, a document is presented to the user as a relevance candidate according to the 2nd criterion if some proportion of its terms have already been seen in documents he has judged relevant, and a proportion have not been seen at all. A weighting factor is used to determine the relative importance of presence in $Pool_R$ versus novelty. $Pool_R$ and $Pool_N$ are updated based on the user’s relevance judgment. The feature weights of terms in the document are also updated based on the user’s relevance judgment. (Note that the presence of many novel terms can cause a document to be presented to the user, but only his relevance judgment can cause the feature weights and pool contents to be updated.) The weight of a given term is increased if the document was retrieved on the basis of the 2nd (novelty) criterion, and the user judges it relevant. The weight of a term is decreased if the document was retrieved on the basis of the first (relevance) criterion, and the user judges it non-relevant.

Leaving the realm of adaptive classification methods applied to time-varying data, let us consider methods of classification where the training set is static. As noted above, the
first consideration is how to limit the feature space. Once the feature space has been reduced sufficiently, a variety of classification training or learning methods are available. Let’s consider some of these methods briefly.

Relevance feedback weighting (e.g., the Rocchio formula discussed in a later section), can be applied to the training set. If nothing is known about the classification weights initially, then the weight of the original “query” to be refined or expanded by feedback in the can be set to zero. In other words, we are using relevance feedback here to generate the classification “query” rather than to refine it as in an ad hoc query application. Schutze et al.[SIGIR’95] tried Linear Discriminant Analysis (LDA), Logistic Regression, and Neural Networks. In contrast to Rocchio, all three of these methods “have decision rules that are derived via explicit error minimization.” Note that logistic regression and Rocchio formulas both have constant parameters that need to be determined but with logistic regression, the parameters are computed directly to optimize the formula whereas in the Rocchio approach, the parameters (A, B, and C) can only be determined by systematic trial and error. LDA classifies the population into two (or more) distinct groups. The separation between the groups is maximized by maximizing an appropriate criterion, e.g., the “separation of the vector means” of the groups. [Hull, SIGIR ‘94] For the present case, LDA computes a linear function $z$ of the document descriptors that distinguishes the two groups of interest, relevant documents and non-relevant documents, “as widely as possible relative to the variation of values of $z$ within [each of] the two groups.” [Hoel, 1971] Again, this is a direct rather than a trial and error procedure. Schutze et al. found in their experiments that all three of these “classifiers perform 10-15% better than relevance feedback via Rocchio expansion for the TREC-2 and TREC-3 routing tasks.” Hull [SIGIR ‘94] also employed discriminant analysis. Hull applied this technique to a feature space whose dimensionality was reduced using LSI. As a further refinement, LSI was applied not to the entire collection used for training but to the set of relevant documents for a given query. Hence, LSI must be applied separately for each query but the document term matrix to which it is applied in each case is much smaller than the matrix for the entire collection.

Dumais et al. {CIKM ‘98] compared five classification methods that work by learning from a training set: Rocchio relevance feedback, decision trees, Naive Bayes, Bayes Nets, and Support Vector Machines (SVM). They trained on the “so-called Reuters-
21578 collection,” a collection of news stories. They “used 12,902 stories that had been classified into 118 categories (e.g., corporate acquisitions, earnings, money market, grain, and interest).” (Note that news stories, though an important, and widely used, type of text corpus, are in some respects particularly easy, because of their relatively standard organization, conventions, and vocabulary.)

Dumais et al. represented the documents in the traditional way as vectors of words. Then, they performed feature selection to reduce the dimension of the vectors. They used mutual information, $MI(x_i, c)$ as the feature selection measure, where $x_i$ is the $i$-th feature, and $c$ is the category for which the various classifiers are being trained. (See the section on Logical Imaging for the definition of “mutual information.”) They selected the 300 “best” features (highest MI value) for SVM and decision trees, and 50 features for the other three classification methods.

Since Rocchio’s method is here being used for classification rather than query refinement, there is no “initial query” term, as noted above. Dumais et al. also elected to discard the negative examples, i.e., the documents that were not relevant to the given category for which the classifiers were being trained. Since the training set starts with relevance judgments for each category, there are no interactive relevance judgments. Hence, the Rocchio formula for a given category was reduced to computing the centroid (the average) of the documents labeled relevant to the given category. At test time, a new document was judged relevant to a given category if its similarity to the category’s centroid (as measured by the Jaccard similarity measure) exceeded a specified threshold.

Lewis and Gale [SIGIR ‘94] use a variation on traditional relevance feedback which they call “uncertainty sampling.” In any situation where the volume of training data is too large for the user to rate all the documents, some sampling method is required. In traditional relevance feedback, the sample the user is asked to classify consists of those documents that the current classifier considers most relevant. Hence, Lewis and Gale call this approach “relevance sampling”. It has the notable virtue, especially if the relevance feedback is taking place while the system is operational, that the documents the user is asked to classify are the ones that (as far as the classifier can tell) he wants to see anyway. However, if the training is taking place before the system is operational (or in a very early stage of operation) and the primary objective is to perfect the classifier, then uncertainty sampling (derived from “results in computational learning theory”) may
work better. The method assumes a “classifier that both predicts a class and provides a measurement of how certain that prediction is. Probabilistic, fuzzy, nearest neighbor, and neural classifiers, along with many others, satisfy this criterion or can be easily modified to do so.” The sample documents chosen for the user to rate are those about which the classifier is most uncertain, e.g., most uncertain whether to classify them as relevant or non-relevant. For a probabilistic classifier (such as the one they actually describe and test in their paper), the most uncertain documents are those that are classified with a probability of correct classification close to 0.5. Lewis and Gale obtained substantially better classification for a given sample size when the classifier was trained by uncertainty sampling of the training set than when it was trained by relevance sampling (and far better than with training on a random sample).

Yang [SIGIR ’94] addresses the classification problem that there is often a wide gap between the vocabulary used in documents to be classified and the terms used in the class or topic (here called “category”) descriptions, i.e., the “queries.” The training set consists of documents to which users have manually assigned category descriptions. (Yang is using a medical application so examples of category descriptions are “acquired immunodeficiency syndrome” and “nervous system diseases”.) A given category may be assigned to many documents. More surprisingly, the same document, e.g., a common diagnosis, may occur multiple times in the training set. Even more surprisingly, the same category may be assigned multiple times to the “same” document; the way this comes about is that the category is assigned to two distinct documents which become identical as the result of aggressive application of a stoplist. The problem is to classify, i.e., assign appropriate category descriptions to, a new document. Yang’s approach consists of two stages. In the first stage, she computes the conventional cosine similarity between the new document to be classified and the documents in the training set, e.g., the similarity \( \text{sim}(X, D_j) \) between the new document \( X \) and a training document \( D_j \). In the second stage (the novel part of the method), she estimates the conditional probability \( \Pr(c_k|D_j) \) that a given category \( c_k \) is relevant to a training document \( D_j \). \( \Pr(c_k|D_j) \) is estimated as the “number of times category \( c_k \) is assigned to document \( D_j \)” (see above) divided by the “number of times document \( D_j \) occurs in the training sample”. Then \( \text{sim}(X, D_j) \) is multiplied by \( \Pr(c_k|D_j) \) for each \( D_j \) and this product is summed over the \( N \) top-ranking \( D_j \)’s, i.e., the ones most similar to \( X \). Experimentally,
Yang found that the optimum value of $N$ for her collection was $N = 30$. The result is $\text{rel}(c_k|X)$, a relevance score for $c_k$. These scores are not probabilities but they provide a ranking of categories for the given document $X$ similar to what would be obtained with probabilities. This ranking can then be used to assign the highest ranking categories to document $X$.

DR-LINK [Liddy et al., ACMIS, ‘94] deals with an issue that arises whenever a collection or stream of documents must be categorized and routed according to multiple topics. The distribution of relevant documents with respect to computed topic similarity scores will tend to vary widely from one topic to another. For example, suppose that a categorization engine ranks a document population with respect to two topics, $T_1$ and $T_2$, producing two separate document rankings, one with respect to $T_1$ and the other with respect to $T_2$. In the $T_1$ ranking, 95% of the documents actually relevant to topic $T_1$ (as judged by human users) may be found in the top ranking 5%. On the other hand, in the $T_2$ ranking, it may be necessary to traverse the top ranking 35% to find 95% of the documents actually relevant to topic $T_2$. Hence, quite different topic similarity thresholds are required for $T_1$ and $T_2$ respectively. DR-LINK deals with this problem by developing a multiple regression formula on document training sets for each of the topics to be categorized. (Clearly, this approach only applies to an application such as routing, where a fixed number of topics will be applied to a changing population of documents, and a training set of typical documents is available for each topic.) The regression formula for each topic has two independent variables, (1) the desired recall level, e.g., 95% in the example above, and (2) the top-ranked document’s similarity. The dependent variable returned by the regression formula is the estimated topic document similarity score threshold needed to achieve the desired level of recall. Hence, a different threshold can be computed for each topic. In tests involving TREC-2 topics and data (173,255 Wall Street Journal articles from 1986-1992), this formula proved quite effective at computing a threshold appropriate for a given topic and desired recall level. Moreover, it was found that because a few relevant documents ("stragglers") tend to be low-ranking, the number of documents that need to be examined can be drastically reduced by lowering the recall level from an unrealistic 100% to a more realistic 80%.
2.6. Natural Language Processing (NLP) Approaches

The phrase “Natural Language Processing” (NLP) approaches” to IF refers here to all methods based on knowledge of the syntax and/or semantics of the natural language in which document text is written, or knowledge of the world, e.g., the application domains, to which the documents refer. Hence such approaches may also be broadly characterized as *semantic* approaches, in the sense that they attempt to address the structure and meaning of textual documents directly, instead of merely using statistical measures as surrogates. However, as discussed below, there are three sources of terminological confusion. First, the term “semantic” is sometimes used to refer to one particular level of NLP, although in reality it is applies to (at least) five different levels. Secondly, many NLP techniques, especially in the realm of “shallow” (and correspondingly, computationally efficient) NLP methods employ statistical techniques, e.g., to determine the most likely sense or part of speech of a given word in a given context. Third, NLP techniques are rarely used by themselves in IF. More commonly, they are used to supplement statistical techniques.

Human beings find it amazingly easy to assess the relevance of a given document based on syntax and semantics. They find statistical and probabilistic methods much more difficult, tedious and error prone. For automated systems, the situation is the reverse. They can perform statistical calculations easily. Developing automated systems that can understand documents in the syntactic/semantic sense is much more difficult. As a result, most IF systems to date have been based on statistical methods. Increasingly however, syntactic and semantic methods are being used to supplement statistical methods. The reason is plain. Even the best statistical or probabilistic methods will miss some relevant documents and retrieve some (often quite a bit of) junk. The hope is that an appropriate combination of traditional statistical/probabilistic methods and syntactic/semantic methods will perform better than the statistical methods alone. Ideally, the combination would approach human performance. This ideal is a long way from realization. [Faloutsos & Oard, UmD-CS, 1995].

Note, by the way, that a technique like Latent *Semantic* Indexing (LSI) (discussed above) is not a semantic method in the sense used here despite the presence of the word “semantic” in its name. Rather, it is a statistical method for capturing term dependencies that it is hoped have semantic significance.
Liddy [BASIS, ‘98] classifies NLP techniques according to the level of linguistic unit processed, and (correspondingly) the level and complexity of the processing required. She identifies the following levels: phonological, morphological, lexical, syntactic, semantic, discourse, and pragmatic. The phonological level is the level of interpreting speech sounds, e.g., phonemes. It is mainly of interest in speech to text processing, rather than textual IF.

Several traditional IF techniques do use NLP techniques, almost entirely at the morphological and lexical levels. The morphological level is concerned with analysis of the variant forms of a given word in terms of its components, e.g. prefixes, roots, and suffixes. Hence, traditional stemming techniques that reduce variants of a word to a common root form for query-document term matching, exemplify morphological IF processing. The lexical level is concerned with analysis of structure and meaning at the purely word level. For example, traditional lexical IF processing includes construction of stop lists of words believed to have low semantic content. [Faloutsos & Oard, UMdCS, 1995] (But see below!) Similarly, generation and use of thesauri for query expansion, and of controlled vocabulary lists for indexing and query formulation, are other traditional examples of lexical IF processing. Proper noun identification is another, somewhat newer, form of IF lexical processing. Tagging words with their parts of speech is also a kind of lexical processing, common and well established in NLP, but rare in traditional IF.

The syntactic level is the level at which the syntactic structure of sentences is determined, in terms of the parts of speech of the individual words. In practice, a single sentence can have many possible structures. Determining the correct structure from these alternatives requires knowledge at the higher levels (or statistics based on a training set). For this reason, and more generally because it is relatively expensive computationally, syntactic level processing has been little used in traditional IF. Some use of syntax has been made to identify units larger than single words, e.g., phrases, but even here, statistical co-occurrence and proximity rather than NLP, have been the preferred methods in IF.

The semantic level is the level at which one tries to interpret meaning at the level of clauses, sentences, rather than just individual words. Note that the disambiguation of words having multiple senses is a semantic-level task, because a word can only be disambiguated in the context of the phrase, sentence, or sometimes larger text unit in
which it occurs (and also because disambiguation may require real-world knowledge, generic, user-specific, or domain-specific). Because of the difficulty and NLP sophistication required (and the need for the aid of the higher levels), traditional IF has avoided semantic-level processing in favor of statistical keyword matching, as discussed in earlier sections.

The *discourse* level is the level at which one tries to interpret the structure and meaning of larger units, e.g., paragraphs, whole documents, etc., in terms of words, phrases, clauses, and sentences.

The *pragmatic* level is the level at which one applies external knowledge (that is, external to the document and original query). The knowledge employed at this level may include general knowledge of the world, knowledge specific to a given application domain, and knowledge about the user's needs, preferences, and goals in submitting a given query.

The most important source of semantic content in traditional IF is relevance feedback, the refinement and expansion of a query based on human judgments of which of the documents retrieved by the query are relevant to the given query. Naturally, these human judgments are based on the user's understanding of the semantic content (in the broadest sense) of the retrieved documents, and her understanding of her actual needs. Hence, the feedback is implicitly at the higher NLP levels. However in traditional IF, this feedback is typically used merely to improve statistically the set of term descriptors and the weights assigned to those descriptors. Only the human user “understands” or “processes” the documents at any semantic or natural language level. However, the methods discussed below can be extended to relevance feedback in various ways, e.g., the descriptors extracted from documents identified as relevant can be higher-level linguistic units such as phrases, or even concepts that do not actually appear in the documents themselves.

This section discusses research into the application of the higher levels of NLP, i.e., syntactic, semantic, discourse, and pragmatic, to the classic problems of IF. It also discusses advances at the lexical levels, e.g., improved proper noun recognition and classification.
It should be stressed that, almost without exception, the NLP methods discussed below are used in conjunction with, not in place of, traditional boolean, vector, and statistical term weighting techniques for document-topic matching and document categorization. [Lewis et al, CACM] Semantic methods can be used to extend the terms to which matching is applied from keywords to key phrases. They can be used to disambiguate terms that have multiple meanings, or fit multiple parts of speech. They can be used to map keywords, phrases and proper nouns into conceptual terms, e.g., subject category terms, that express more naturally the user’s interests, but that will not necessarily co-occur in both the user’s topic statement and the relevant documents. They can be used to supplement the terms in a user’s query with candidate synonyms. They can be used to identify semantic relationships among the keywords or phrases occurring in a topic, or in a candidate document. Hence, topics and documents can be matched not only on whether the specified keywords occur in both, but on whether they occur in the same (or similar) relationship in both topic and document. Semantic methods permit the identification of relationships other than the purely boolean, e.g., given the keywords “company” and “investigation,” semantic methods can distinguish a query about a company performing an investigation from a document about a company being investigated. Note that statistical or user-specified weights can be applied to all of these semantically derived terms, i.e., to phrases, conceptual terms, synonyms, relationships, etc.

Semantic methods can significantly enhance term normalization techniques. Term normalization reduces query and document descriptor terms to a common form for matching purposes. Stemming is the most common type of normalization in traditional IF systems. Another traditional technique is the manual assignment of index terms to documents from a controlled vocabulary. Semantic methods permit more sophisticated forms of automated normalization. Varied syntactic forms can be mapped to a standard syntax, e.g., “investigation by the company,” and “the company is investigating” can be mapped into the common noun phrase “company investigation.” Related words, e.g., house, apartment, and hut, can be mapped into a common subject category, “dwelling.” Varied forms of a proper noun can be mapped into a standard form. And so on.
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

2.6.1 Phrase Identification and Analysis

A common use of syntactic (and to some degree semantic) methods is phrase identification. [Riloff, SIGIR’95] [Jacqemin & Royaute, SIGIR ‘94] [Kupiec, SIGIR ‘93] [Anick & Flynn, SIGIR’93] [Strzalkowski & Carballo, TREC-2] [Evans & Lefferts, TREC-2] Phrases are typically identified in IF so that they can be used as descriptor terms, i.e., so the descriptors of a document are not limited to single words. Traditional methods identify phrases by statistical co-occurrence, e.g., co-occurrence of pairs of terms in documents at a rate greater than would be expected by random chance. Co-occurrence can be combined with adjacency, e.g., if two (or more) terms co-occur within a few words of each other at a rate greater than chance, the probability that they are related semantically certainly increases. If the terms in question are also related syntactically, the chance that they form a phrase is still greater. Syntactic analysis can identify phrases even when the terms of which they are composed are not adjacent, or do not co-occur with greater than chance frequency. However, extraction of phrases by purely syntactic means alone is seldom effective since it is likely to extract many phrases of little value for characterizing the topic(s) of a given document or query. [Croft et al., SIGIR ‘91] A combination of syntactic and statistical methods is more effective. [Lewis, SIGIR ‘92] [Lewis et al., SIGIR 96] Lewis et al. suggest that statistical weighting techniques should be applied to phrase descriptors, even if they are generated by NLP or combined NLP/statistical techniques. However, they suggest that “[w]eighting for phrases may differ from weighting for single-word terms to allow for their lower frequency and different distribution characteristics.”

Lewis et al. urge that phrase descriptors should be “linguistically solid compounds,” e.g., noun phrases, etc. However, they also stress that phrase matching should reflect the variety of forms a phrase can assume, and the varying degrees of evidence provided by each form. For example, if the noun phrase “prefabricated units” is extracted from a document, it is likely that it signifies the corresponding concept. The presence in a given document of the verb phrase “[they] prefabricated units” would provide weaker evidence for the presence of the concept. The co-occurrence in a given document of the two words “prefabricated” and “units” in close proximity, e.g., in the same sentence or paragraph, but not in the same syntactic phrase, would provide still weaker (but not non-zero) evidence that the concept was present. The co-occurrence of the two words in
different paragraphs of the given document would provide much weaker evidence still, and so on.

The above example also illustrates that NLP identification and extraction of phrases can have an important effect on the traditional approach of word stemming. A stemmer would reduce “prefabricated” and “prefabricate” to the root “prefabricat.” But, as we see above, the distinction between “prefabricated” and “prefabricate” (or prefabricating, for that matter) may be the difference between a noun phrase and a verb phrase, and hence a corresponding difference in the evidence that the respective phrases contribute about the topic(s) discussed by a document that contains them.

Lewis et al. also note that the degree of sophisticated NLP and statistical processing applied to extraction of phrases and other compound terms might be considerably greater for user queries than for documents. There are several reasons for this: First, the number of queries is normally very much less than the number of documents, so that an IF system can afford to process the queries more carefully. Second, it is very important to understand what the user’s requirements are; this is complicated by the fact that queries are generally much shorter than documents, and often more carelessly formulated by users who are not professional IF searchers. (For the same reasons, it is very desirable to support interactive query refinement, using thesauruses, NLP analysis of user queries, relevance feedback with regard to both good terms and good documents, as assessed by the user, etc.) Third, any error in extracting phrases and compound terms from documents can be corrected (or at least compensated for) during the query document matching process, since the matching process will take into account not merely a single phrase that may have been extracted from a given document incorrectly, but the context of other words and phrases that have been extracted from the same document.

The burden of applying expensive NLP to large document collections can be further eased by a two-step process: First, coarse ranked filtering of candidate documents using statistical and shallow NLP techniques. Second, more sophisticated NLP applied only to the much smaller list of highly ranked documents retrieved by the first stage.

Shallow or coarse NLP refers to techniques for extraction, based on local contexts, of noun and verb phrases, compound proper nouns (discussed later), simple basic propositions, e.g., produce (factory, house), complex nominals, e.g., “debt reduction,”
“health hazards,” and simple concept-relation-concept (CRC) structures, e.g., the
sentence fragment “the company’s investigation of the incident...” generates two CRC
triples, each relating a noun to the verb: [investigate] -> (AGENT) -> [company], and
[investigate] -> (PATIENT) -> [incident], which convey the information that the
company acts in the investigation, and the incident is the object of the investigation.
CRC triples and structures constructed from them are discussed further below. Here,
it should be stressed that shallow, localized extraction of such “solid” linguistic
components is contrasted with complete parsing of sentences and paragraphs, a very
much more difficult and expensive process, requiring the support of elaborate knowledge
bases (KB’s) and typically generating multiple legal parses.

Lewis et al. further argue that, although NLP may be used to supplement word
descriptors with phrase and compound term descriptors, these compound term descriptors
should not be combined into higher level structured descriptors, e.g., frames, templates.
Note that they do not object to such higher-level descriptors for knowledge
representation and filtering. Rather, they object to them as index descriptors for
document filtering. Their argument is that such higher-level structures are labor
intensive to produce, and that an exact query-document match at the level of such
elaborate structures is very unlikely. In subsequent sections, there is a discussion of
certain higher-level document structures. In particular, there is a discussion of exciting
work in the area of document discourse structure, the structure of clauses, sentences, and
paragraphs that determines the narrative or expository flow of a document. It is this
discourse structure that enables a reader to follow the flow of the writer’s argument,
and understand what the writer is saying. Some research indicates that combining term
descriptors with this discourse structure can enhance document filtering. It can also be
used for forms of knowledge filtering, e.g., document summarization. But note that the
discourse structure is not a structure composed of conventional document descriptors
such as words and phrases. Rather, it is a wholly separate, higher-level structure of the
document as a whole. The words and phrases are related to the discourse structure either
by being located in identifiable components of the discourse structure, or by serving as
linguistic clues to identify the components of the discourse structure.

Strzalkowski and Carballo [TREC-2] extract phrases syntactically from a large
collection, but then apply a variety of statistical techniques to these phrases before
formulating queries.
The extracted phrases are statistically analyzed in syntactic contexts in order to discover a variety of similarity links between smaller subphrases and words occurring in them. A further filtering process maps these similarity links onto semantic relations (generalization, specialization, synonymy, etc.) after which they are used to transform a user's request into a search query.

As an example of “statistical analyzing in syntactic contexts,” they filter out poor terms with a statistical measure called Global Specificity Measure (GTS) “similar to the standard inverted [sic] document frequency (idf) measure except that term frequency is measured over syntactic units [e.g., phrases such as verb-object, noun-adjunct, etc.] rather than document size units.” (italics mine). (See below for further discussion of GTS.) As an example of using a derived semantic relation “to transform a user’s request into a search query,” they offer the example of adding the compound term “unlawful activity” to “a query … containing the compound term ‘illegal activity’ via a synonymy link between ‘illegal’ and ‘unlawful.’”

Note that the above example illustrates another use of combining syntactic with statistical techniques: the automated creation of thesauri. Traditionally, thesauri have been constructed either manually, e.g., WordNet, or statistically. (Because of the many semantic relations it supports, WordNet, described below, is more properly described as a semantic network.) Stzalowski and Carballo illustrate that thesaurus type information (not only synonymy but other semantic relations like specialization and generalization) can be derived by applying statistical techniques, e.g., co-occurrence, similarity, etc.) to syntactic units derived by automated parsing (and not only to the syntactic units themselves but to their internal structure). Note that even when the thesaurus generation, e.g., determination of synonymy, is wholly statistical [Evans & Lefferts, TREC-2] [Milic-Frayling et al. TREC-4], the extraction of candidate phrases syntactically allows these phrases to be units to which the statistical methodology can be applied.

Phrases are commonly extracted to serve as document (query) descriptors, i.e., as index terms. However, they are also used in a variety of other, related ways. For example, phrase extraction is used as a key feature of a system, MURAX [Kupiec, SIGIR ’93], that answers natural language questions that require a noun phrase as answer. The answers are obtained from a large, general purpose, on-line encyclopedia.
The method does not depend on domain-specific knowledge; the questions can be on any topic whatever, as long as they require noun phrase answers. But the method does make use of other simple semantic and syntactic heuristics, e.g., the question can begin with “who”, “what”, “where”, or “when” because such questions usually require persons, things, places, or times (which can be expressed as noun phrases) as answers; questions beginning with “why” or “how” (which usually require procedural answers) are forbidden. Phrases are extracted from the question, and the question is reformulated as a structured boolean query with proximity constraints. The query returns a number of encyclopedia articles (possibly none). The number of hits (retrieved articles) can be increased by broadening the query, e.g., relaxing proximity constraints, dropping phrases, or reducing a phrase to a sub-phrase. The number of hits can be decreased by narrowing the query, e.g., adding a term such as the main verb. Retrieved articles are ranked by number of matches with the query. Phrases or words that match phrases or words in the query are then extracted from the retrieved articles. An answer to the original question may require information from multiple retrieved articles. A variety of term matching rules are used; the phrase that appears in an article need not have exactly the same form as the corresponding phrase in the question, e.g., “was succeeded by” can match “who succeeded.” This system demonstrates that “shallow syntactic analysis can be used to advantage in broad domains.”

Syntactic phrase extraction is performed and used somewhat differently by Riloff [SIGIR ‘95]. The goal is one of those specified for the Message Understanding Conference (MUC) series rather than the Text Filtering Conference (TREC) series. In this case, the goal is extraction of data about a particular topic, e.g., terrorist events or joint ventures, from each relevant document, and the filling out of a topic-specific template for each such document. For example, given a document reporting a terrorist event, the goal might be to extract such key elements as the name of the perpetrator, the name of the victim, the location at which the event took place, etc. Since the objective is not merely to identify relevant documents but to “understand” each document well enough to extract key elements, Natural Language Processing (NLP) naturally assumes greater importance. So does knowledge of the domain to which the extraction task applies, e.g., terrorism. Hence, Riloff extracts instances of phrase patterns called “concept nodes.” A concept node is a specific linguistic pattern containing a specific word. For example, each occurrence of the concept node called “$murder-passive-victim$” is a
string of the form “<X> was murdered” (or “<X> were murdered” for multiple victims).

An important result demonstrated by Riloff (also noted by others) is that such common IF techniques as stop lists and stemming, which are intended only to remove noise words and noise variation of a given term, can in fact remove clues crucial for judging the relevance of a document or the semantics of its content. For example, the presence of the term “dead” in a document was not a reliable guarantee that the document described a murder, but the phrase “was found dead” proved to be an extremely reliable descriptor for that purpose. It evidently “has an implicit connotation of foul play.” Similarly, the presence of the term “assassination” (singular) in a document proved a much more reliable indicator that the document described a specific assassination event than the presence of the term “assassinations” (plural). The plural often referred to assassinations in general. Prepositions, passive vs. active verb form, positive vs. negative assertion, also proved significant to determining the significance of a phrase as a descriptor, at least within a specific domain. For example, the term “venture” by itself in a document was not a good indicator that a document described a joint venture. But the phrases “venture with” and “venture by” proved very good descriptors indeed, e.g., 95% precision for the test collection.

2.6.2 Sense Disambiguation of Terms and Noun Phrases

Another area (besides phrase extraction and phrase analysis) where researchers have tried to use syntactic/semantic techniques to improve IF performance is “word sense disambiguation.” [Chakravarthy & Haase, SIGIR ’95] [Sanderson, SIGIR ’95] [Voorhees, SIGIR ’93] It is common place that natural language words have multiple meanings. More precisely, a string of characters representing a word (a “lexical token” in the jargon of the field) can represent multiple words, each with a different meaning (a different “sense”). (The technical term for such a word is polysemous.) Sometimes, the meanings are closely related meanings of what we tend to consider the “same” word. Often, they are two (or more) completely different words that happen to be spelled the same, e.g., “bank” can mean either a financial institution or the border of a river, “bat” can mean either a flying mammal or a sporting implement, etc. It is plausible to suggest that if one can use syntactic/semantic methods to determine which “sense” of a word is intended by each occurrence of the word in any given document or query, one will be
better able to retrieve the documents relevant to a given query (better recall) and to reject non-relevant documents (better precision). In particular, one would be able to avoid matching one sense of a word in a query with a completely different sense of the word in a document.

One crucial issue that arises in sense disambiguation is how fine-grained the sense distinctions are to be. This may depend on the resources available, e.g., an on-line Machine Readable Dictionary (MRD) such as the Longman Dictionary of Contemporary English (LDOCE) [Procter, 1978] may provide a relatively large number of senses relative to what may be generated by hand from a training corpus. On the other hand, the number of senses to distinguish depends heavily on the task for which the sense disambiguation is being performed. The number of senses that need to be distinguished for IF, may be considerably less than the number that need to be distinguished, e.g., for polished machine translation.

One reason is that senses of a single word that are closely related in one language may (and often will) correspond to different words in another language.

Moreover, for IF purposes, it may often be sufficient to distinguish fairly broad subject categories. For example, the 1978 LDOCE distinguishes 13 senses of the word bank. [Guthrie, 1993] Yet, these thirteen senses may be grouped into several broader categories. One coarse sense of bank is a repository to which one can make deposits and from which one can make withdrawals. At a finer grain, one can distinguish a financial bank from a blood bank or a leave bank. Similarly, another quite different coarse sense of bank is a heap of material. At a finer grain, one can distinguish a snow bank, a sand bank, a cloud bank, and the bank of a river. For many IF purposes, it may be sufficient to distinguish the senses of bank at the coarser level, or to distinguish a financial bank from other repositories.

Guthrie et al. [1991] disambiguates words like bank by using word co-occurrence statistics derived from an electronic reference source, in this case the LDOCE MRD. The problem is to disambiguate polysemous words like bank, occurring in a test corpus. Given an occurrence in the test corpus of bank, they extract the local context of the given occurrence. The context may be the sentence in which the given word occurs, e.g., “We got a bank loan to buy a house,” or some specified number of words on either side of the given word. This context is then matched against each neighborhood of the given word, derived from the LDOCE.
The neighborhood of a given word is the set of words (excluding stop words) that co-occur in the definition of one sense or group of related senses of the given word. The electronic version of LDOCE specifies a set of relatively broad subject categories, each category designated by a Subject Field Code (SFC). (The use of SFC’s is discussed in greater detail below, in the section on Concept Identification.) SFC’s are assigned to some of the sense definitions of a given word. For example, the sense of bank as “a place in which money is kept and paid out on demand” is assigned an SFC of “EC” which stands for the subject category “Economics.” (The verb sense of bank, “to put or keep (money) in a bank” is also assigned the code “EC.”) At the simplest level, the neighborhood of the Economics sense of bank is the set of words (excluding stop words) in all the definitions of bank that are assigned an SFC of “EC.” Since the definitions are relatively short, a co-occurrence neighborhood is computed for each non-stop word in each “EC” definition of bank. In other words, since the word “money” occurs in one or more EC definitions of bank, a neighborhood is computed for money. The union of these neighborhoods becomes the neighborhood for the EC sense of bank. Guthrie et al. then compute the overlap (number of words in common) of this neighborhood with the context of the occurrence of bank to be disambiguated. In the same way, the overlap is computed between the context and the neighborhood of every other sense (or subject category) of bank. The sense of the neighborhood with the greatest amount of overlap with the context is chosen to be the sense of bank in the occurrence being disambiguated. The Guthrie procedure is automatic, but the best success rate achieved in this and similar approaches is only about 70% accuracy.

From the sentence used in the above example, i.e., “We got a bank loan to buy a house,” it is easy to see why sense disambiguation using MRD definitions might be relatively ineffective. A key word in the above sentence is “loan.” Although a great many loans involve money, it is entirely possible that none of the definitions of money in the MRD involve the use of the word “loan.” The word “loan” may co-occur with the word “money” with significant frequency in a news or business corpus, yet not occur in definitions of “money.”

Voorhees [SIGIR ‘93] uses the large general purpose semantic network, WordNet [Miller, Jlex, 1990]. WordNet groups words into strict synonym sets called “synsets.” The synsets are divided into four main categories: nouns, verbs, adjectives, and adverbs.
Within each category, synsets are linked together by a variety of semantic relations appropriate to the category, e.g., for nouns the relations include hypernymy/hyponymy (the “is-a” or class/subclass relation, e.g., a car is a vehicle), antonymy (word A means the opposite of word B), and several “part-of” relations. Voorhees, in the reported research, disambiguates nouns only. The approach is to group noun synsets in WordNet so that the synsets in a given group all contain words with closely related senses. The groups are called “hoods”. A hood is effectively similar to a class in Roget’s thesaurus. Given a document, she can then count the number of nouns in the document that are found in a given hood, i.e., the number of nouns that can have the sense (or related senses) of a given hood. Of course, a given noun can belong to multiple hoods, corresponding to multiple senses. In such cases, the noun is assumed to have the sense it possesses in the hood which contains the largest number of words (or word occurrences) from the document. In other words, it is assumed that in a given document, a given noun will tend to co-occur most with other nouns having related senses. (The word “bat” is ambiguous, but if it co-occurs in a document with the also ambiguous words, “base,” “glove,” and “hit,” it is very likely that the word is being used in its baseball sense.)

Leacock et al. [Corpus Proc, 1996] have investigated sense disambiguation of words in a large text corpus by statistical classification based on term co-occurrence in the contexts in which the given words occur. In this study, a context was defined to be a sentence in which a given word occurred, and the preceding sentence. The preceding sentence was included in the context because a given word is often used anaphorically. If the preceding sentence also contained the given word in the same sense, then the sentence preceding that sentence was also included in the context. A single polysemous word, “line,” was studied. A training set was constructed consisting of contexts for each of six different senses of “line.” These included: line of text (“a line from Hamlet”), a formation (“a line at the box office”), an abstract division (“the narrow line between tact and lying”), a telephone connection (“the line went dead”), a thin, flexible cord (“A fishing line”), and a class of product (“a product line”). Elements varied included: type of classifier (Bayesian, vector space, and neural network), number of senses (two, three and six), and number of contexts in training set (50, 100, and 200). The machine classifiers were also compared to human classifiers. All of the machine classifiers performed best with 200 contexts (71% to 76% correct answers). Performance of the
three classifiers tended to converge as the number of contexts went up. With only two senses to distinguish (between product line and formation), the accuracy was over 90%. However, with three senses (product line, line of text, formation), the classifiers did only a little better than with six (mean accuracy of 76%). In general, some senses were harder than others for all three classifiers to identify; the hardest were: line of text, formation, division, in that order. The three easiest for all three classifiers were product, phone, and cord, although the order of ease for these three varied with the classifier. Moreover, for any given sense, some contexts made sense identification easier than others. Interestingly, the humans, when given the same information as the machine classifiers, e.g., a vector of stemmed substantive words for the vector space classifier, found the same senses (and context within a sense) easy to distinguish, and the same senses difficult. In general, the humans did better than the machine classifiers. The exception was the “easy” contexts, chosen because the machine classifiers made no errors on them; the humans had a 15% error rate on these contexts. Most significant, when the humans were given the original sentences comprising each context, their performance was nearly perfect. On the other hand, when they were given the same input as the Bayesian classifier, i.e., the complete set of words in the context, with no stemming or stop-word removal, but with the words in reverse alphabetical order (because none of the classifiers used word order), their performance was much the same as when stop words (also called function words) were removed. The stop words were of no value to the human classifiers except when they were presented in the proper syntactic order. Plainly, the machine classifiers would do much better if they could make effective use of order and proximity as the human classifiers do.

Evaluation of sense disambiguation can be very tedious since it is necessary not only to evaluate the retrieved output for relevance, but to examine the filtering process at the level of individual words to determine how it was affected by the disambiguation. Moreover, it is sometimes difficult to determine what the intended sense of a word, e.g., in a short query, actually is. Sanderson [SIGIR ‘94] reports on a novel method of evaluation, attributed to Yarowsky [Hum Lang, 1993]. The approach is to create ambiguous pseudo-words artificially by replacing each occurrence of a pair of distinct words in the document (any pair, not just adjacent words) by a pseudo-word formed by concatenating the two actual words. The effect is to create a new document with half as many word occurrences, each having a known in advance ambiguity. Still greater
ambiguity can be created by generating pseudo-words composed of $N$ real words where $N$ may be, 3, 4, ..., 10, etc. Of course, indexes and queries have to be modified correspondingly. “The disambiguator is then applied to each occurrence of [each] new word. Evaluation of the disambiguator’s output is a trivial matter as we know beforehand the correct sense of each occurrence of the word.”

The surprising result of such research into disambiguation is that it seems to improve IF performance very little. Indeed, in most cases, it actually degrades performance. Sanderson found that adding quite large amounts of controlled ambiguity, e.g., size ten pseudo-words, had little effect on IF filtering. Removing a controlled amount of ambiguity seemed to make IF performance worse. Voorhees too found that in most cases, performance was degraded by disambiguation. The following reasons were identified: If the number of terms in the query is not very small, the term combination itself appears to have the effect of disambiguating documents and queries (e.g., as in the “base, bat, glove, hit” example above). This phenomenon is also noted by Lewis et al. [CACM’96] On the other hand, if the query is very small, then it provides very little context for the disambiguator to use. Hence, the disambiguator is likely to introduce errors, resulting in missed query document matches. According to Voorhees, “[t]hese results demonstrate that missing matches between the documents and query degrades performance more than eliminating spurious matches [by disambiguation] helps filtering for small homogeneous collections.” Sanderson used a less homogeneous collection with similar results. He found that the disambiguator had to be at least 90% accurate to avoid performance degradation. Supplying a domain-specific context for short queries (as humans do) and selecting the most frequent sense for an ambiguous term may alleviate the problem. However, there are some queries for which Voorhees found that disambiguation was very helpful.

2.6.3 Concept Identification and Matching

In all of the previous discussion of methods for matching documents against queries or topics, the query and document terms being matched are words or phrases extracted from the given queries and documents, or (in the case of techniques like LSI) factors derived statistically from the words. The words may be subject to pre-processing, e.g., stemming to reduce variants to a common form, but in essence query words are being matched against document words. However, the words are essentially being used as
surrogates for the concepts they express. Boolean expressions, vectors, SVD factors and the like may be used to capture the intended concept more precisely, by specifying the words that co-occur in a given semantic context. However, what the user really wants is to retrieve documents that are about certain concepts. Of course, in many cases the user is looking for documents about a specific named entity, e.g., a given person, or a given book. Even then, certain concepts are usually implicit, e.g., if she is looking for documents about a person named “Baker,” she wants to specify that the documents must be about a person named “Baker” and not about the profession, “baker.” Moreover, she may want to see documents about an author named “Baker,” or a CEO named “Baker,” or a Baker who participated in a certain criminal action, etc.

Liddy et al., [ACM Trans IS, 1994] have developed a technique for matching topics (or topic profiles) against documents at the concept or subject level. They have implemented a text categorization module based on this technique; the module provides a front-end filtering function for the larger Document Filtering through LINguistic Knowledge (DR-LINK) text filtering system, although it can also serve as a stand-alone routing or categorization system and has been tested in this mode, e.g., using TREC 2 data. The text categorizer described in TREC-2 used a machine-readable dictionary (MRD), the “Longman’s Dictionary of Contemporary English” (LDOCE). The second edition (1987) of LDOCE contains 35,899 “headwords”, i.e., words for which the LDOCE contains an entry. Each headword may contain multiple definitions, corresponding to multiple senses of the word. The LDOCE contains 53,838 senses of the 35,899 headwords. For each word, the LDOCE specifies one or more “parts of speech.” For each part of speech associated with a given word, the LDOCE specifies all its senses, and assigns a “Subject Field Code” (SFC) to each sense. For example, “earth” can be both a noun and a verb (the latter chiefly in British terminology). As a noun, it can refer to the planet on which we live (SFC=ASTRONOMY), or to the soil in which we plant crops (SFC=AGRICULTURE), or to a class of chemicals (SFC=SCIENCE), etc. In all, the LDOCE assigns six SFC’s to “earth” as a noun. (Current versions of DR-LINK use a proprietary MRD, containing a proprietary, and larger, set of SFC’s. [Liddy, PC] Since this MRD is specifically designed to serve the needs of DR-LINK, it omits many features of stand-alone, commercial MRD’s such as the LDOCE.)
The categorizer uses the SFC’s to construct SFC vectors, instead of term vectors. First, it assigns one or more parts of speech to each word in the document or topic statement, using a probabilistic part-of-speech tagger, POST. (As in most vector space approaches, the topic or query is treated as just another document; a vector is developed for the topic in exactly the same way as for the documents to be categorized.) Then, it looks up each part-of-speech tagged word in the LDOCE, and attaches to the word all SFC’s associated with it. For example, if a particular occurrence of “earth” in a given document is tagged as a noun, it would have six SFC’s attached to it, including ASTRONOMY, AGRICULTURE, SCIENCE, etc. If the word is not found in the LDOCE, stemming is applied, and a second LDOCE lookup is performed. Note that some words may not be assigned any SFC at all, either because the word is not in the LDOCE, or because it does not appear in the LDOCE as the part-of-speech with which it has been tagged. Some words may require a larger, or a more specialized dictionary resource than the LDOCE.

Since a given word, even when tagged with a given part of speech, may still have multiple SFC’s assigned to it (as in the example of “earth” above”), the word sense must be disambiguated. However, in contrast to the word sense disambiguation discussed in the preceding section, here disambiguation is conducted entirely at the SFC, i.e., concept, level. Disambiguation is first performed at the level of the local context, here interpreted as the sentence level. SFC frequencies are computed for the given sentence. There are two reasons why a given SFC may be assigned more than once in a given sentence. First, two or more words in the given sentence may each be assigned the same SFC as one of their respective senses. Second, two or more senses of the same word occurrence, even tagged with the same part of speech, may be assigned the same SFC. Liddy et al. give the example of the sentence, “State companies employ about one billion people.” The word “state” is assigned the SFC “POLITICAL SCIENCE” four times, corresponding to four different senses of the word, e.g., “state” as nation, “state” as subdivision of a nation, “state” as in “separation of church and state,” “state” as in “secretary of state,” etc. The word “people” is assigned the SFC “POLITICAL SCIENCE” twice, e.g., “people” as in “the people of New York,” and “people” as in “the people have chosen a president,” etc. Two important cases of SFC frequency are identified for purposes of sense disambiguation: If a given word within the given sentence is only assigned a single SFC, this is called a “unique SFC.” Obviously, for such a word, no sense disambiguation is
required. On the other hand, an SFC is considered “highly frequent” if it is assigned more than three times over all the words in the given sentence. If a word is assigned multiple SFC’s and one of these SFC’s is highly frequent, that SFC will be assigned to the given word. In the illustrative sentence above, “Political Science” is a highly frequent SFC since it is assigned six times (four for “state” and two for “people”). Hence, it will be assigned as the SFC for both “state” and “people” in that sentence. Note that a word like “people” has other senses, e.g., the LDOCE also assigns the SFC’s “SOCIOLOGY” and “ANTHROPOLOGY” to the word “people.” But in the given sentence, “POLITICAL SCIENCE” is a much more frequent SFC than either “SOCIOLOGY” or “ANTHROPOLOGY,” so it is chosen in that sentence as the preferred SFC to disambiguate “state” and “people.”

If a word in the given sentence remains ambiguous, i.e., the word is assigned multiple SFC’s but none of the SFC’s assigned to the given word is highly frequent, then an SFC correlation matrix is used. This matrix is built from a training corpus; in the research described here, the corpus consisted of 977 newspaper articles. The matrix is 124 X 124 because there were 124 SFC’s in the LDOCE edition used in the research described here. The given word is disambiguated by choosing that one of its assigned SFC’s that has the highest correlation (tendency to co-occur in the same document) with the unique and highly frequent SFC’s in the same sentence. Note that the correlation matrix measures correlations among SFC’s at the document level. Hence, at this stage, document as well as sentence level knowledge is being used for disambiguation.

After every ambiguous word in every sentence in the given document has been disambiguated, a document-level SFC vector is created by summing the single SFC’s assigned to each word. For example, if the SFC “POLITICAL SCIENCE” is assigned as the SFC of one or more words in one or more sentences of the given document, then it will become one of the terms of the SFC vector for the given document, with a term weight determined by its normalized term frequency, or some other appropriate term weighting formula. For example, the reported research uses Sager’s term weighting formula, $f_{in}/k_n$, where $f_{in}$ is the SFC frequency of SFC $i$ in document $n$ and $k_n$ is the number of tokens (SFC occurrences) in document $n$.

Once an SFC vector has been created for every document in the collection being categorized, and every topic for which documents are to be categorized, the similarity
of each document to each topic can be computed, and all documents above a specified threshold for a given topic can be assigned to its category. However, in DR-LINK, the SFC vectors are employed in a more sophisticated manner, to take advantage of the discourse structure of the document. This is described in the next section. Moreover, similarity at the SFC vector level, though it can be used in a stand-alone mode, can also (and in DR-LINK is) combined with similarity at other levels, e.g., matching at the level of proper nouns, relationships, etc. These issues are discussed in sections that follow.

The Liddy topic matching scheme described above is a novel form of “controlled vocabulary” methodology. It is customary to distinguish IF systems as “controlled vocabulary” or “free text vocabulary.” In the former, documents are indexed manually by their authors or by professional indexers from a preset vocabulary of index terms. In the latter (as in most of the research described in this report), the index terms for a given document are generated automatically from the content of the document. The great advantage of the former is that expert human judgment is applied to choose appropriate document descriptors. The great disadvantages are that it is extremely labor-intensive, and that users must know and use the controlled vocabulary in formulating their queries and topic profiles. The Liddy scheme offers “the best of both worlds.” On the one hand, documents are indexed automatically, and the user can formulate queries and topics in terms of her own vocabulary. On the other hand, document and topic terms are mapped from a large dictionary lexicon into a controlled vocabulary of concept terms, the SFC’s. This controlled vocabulary has been developed and refined over a considerable period of time by professional lexicographers. The only drawback is that machine-readable resources such as the LDOCE need to be extended (along with the set of SFC’s) to deal with documents in specialized domains.

Note that relevance feedback (see below) can readily be applied to these SFC vectors [Liddy et al, Online, 1995]. The user’s original query is converted into an SFC vector and matched against SFC vectors representing each document in the given collection, as described above. A list of documents, ranked by SFC vector similarity to the query and above a specified similarity threshold, is returned to the user. The user then identifies those documents on the list (or paragraphs or sentences within those documents) that she considers relevant to the given query. DR-LINK can generate a Relevance Feedback
(RF) SFC vector from each of the user selected documents, and aggregate these vectors into one RF-SFC vector. Since the documents (or paragraphs or sentences) selected by the user should be good examples of the kind of document for which she is looking, the resulting RF-SFC vector should exemplify (better than the original query) the user’s requirements at the conceptual (subject-category) level. Hence, the RF-SFC vector can be used in place of the SFC vector generated from the original query, to calculate similarity at the conceptual level, and return a new ranked list of documents. As with other forms of relevance feedback, this process can be repeated to refine the query further, until no more improvement is observed. Of course as with all relevance feedback, improvement depends upon the existence within the collection of documents that are truly relevant to the user’s requirements.

The advantages of an NLP-based, subject-based system such as DR-LINK, over the far more common keyword-based systems is well-illustrated by the comparison of three IF systems reported by Feldman. [ONLINE, 96] For example, she cites a common experience of IF searchers, expressed by the rule, “Search for wars, and you will also retrieve sports.” It is not hard to see why keyword-based systems, whether boolean or vector space, are vulnerable to such errors. The vocabularies of sports and war overlap to a significant degree, e.g., terms like “battle,” “conflict,” “win,” “loss,” “victory,” “defeat,” and many more. If the user employs some of these terms in formulating her query, a keyword based system will have difficulty distinguishing sports stories from war stories. Feldman encountered this difficulty when she asked for “information about African countries which had civil wars, insurrections, coups, or rebellions.” She encountered this difficulty with both the boolean DIALOG system, and the relevance ranking TARGET system. She encountered this difficulty even though the terms she used (or stemmed), terms like “insurrections,” “coup,” “rebellions, “wars,” are not the most common terms to appear in sports stories. Presumably, some of these terms, or synonyms for them, e.g., synonyms for “war,” do appear in sports stories. The only way to exclude such stories from a boolean query without excluding legitimate stories too, would be to add “AND NOT....,” where the “NOT” is followed by a set of sports-specific terms, e.g., “NOT rugby,” “NOT soccer,” etc. A relevance ranking system might be expected to do somewhat better, if the specified terms occur more frequently in war stories than in sports stories. However, TARGET did not do better on this query than DIALOG in Feldman’s test. DR-LINK did much better
on this query, retrieving 50 relevant war stories, and excluding the “false drop” sports stories altogether.

It is easy to see why DR-LINK was much more successful on this query. The SFC vector approach provides several advantages in such an example. For instance, a sports story would almost always contain some sports-specific terms. Hence, the corresponding SFC vector for such a sports story would contain sports-specific SFC’s. The presence of such SFC’s would help DR-LINK to disambiguate terms that belong to both the sports and war domains correctly, choosing the sports sense of such terms. On the other hand, the fact that all the terms in the user’s query had a war sense and none had a pure sports sense would favor disambiguating the query terms as having their war senses at the SFC level.

It should be noted that DR-LINK is not the only system that indexes a large document collection automatically using a controlled vocabulary. For example, CONSTRUETIS [Hayes et al., 1990] assigns subject category terms automatically from a controlled vocabulary to a large collection of news stories maintained by Reuters. The subject categories include 135 economic categories, e.g., mergers and acquisitions, corporate earnings, interest rates, various currencies, etc. 539 proper name categories (people, countries, companies, etc.) are also supported. Rapid automatic indexing enables the system to keep up with the rapid addition of new stories. Indexing is based on shallow knowledge base techniques. Each subject category is recognized by if-then rules. The “if” conditions are specified in terms of “concepts.” The concepts, in turn, are defined in terms of a pattern language. The pattern language specifies keyword patterns. The patterns may include such features as keyword order, boolean combinations of keywords, gaps for a specified number of arbitrary words, etc. Hence, CONSTRUETIS conceptual indexing by subject category employs much more than simple boolean conditions, but much less than true natural language processing. However, the subject category terms that are assigned automatically to each story (the terms that comprise the controlled vocabulary) are the same terms that users must employ to retrieve the story. By contrast, the user of DR-LINK never sees or needs to know its controlled vocabulary. Documents and natural language topic requests are both mapped automatically into the common, controlled vocabulary of SFC’s.
2.6.3.1 Formal Concepts

The preceding section dealt with concepts assigned to word senses by human beings, e.g.,
domain experts or lexicographers. In this section, we describe how “formal concepts” can
be defined for a given domain, or a given body of text, using a mathematical method
for knowledge representation, exploration, and processing called Formal Context
Analysis (FCA).

FCA is an unsupervised learning technique that “allows implications between
attributes to be determined and visualized.” [Cole et al., Comp Int, 1999]

FCA models a domain as being composed of individual objects (called formal objects)
and their attributes (called formal attributes). The set of objects and attributes chosen to
model a domain is called a formal context. Formal contexts relate objects to their
attributes. A context may be represented as a matrix in which the rows are objects, the
columns are attributes, and each cell specifies whether a given object has (yes) or doesn’t
have (no) a given attribute. If a formal context represents a textual document (or
collection of documents), the objects may be the individuals mentioned in the text, e.g.,
specific persons, companies, locations, buildings, etc., whether named or unnamed.

Correspondingly, the attributes of a given individual object will be those mentioned
explicitly or implicitly in the text as characterizing the objects, e.g., a building may be
small, tall, or ornate, etc. The object type is also considered an attribute of the object,
e.g., the attributes of the Empire State Building include both the descriptive adjective
“tall,” and the object type “building.”

Semantic relations can also be formalized by a context matrix. Each row of a relation
matrix is an ordered pair of objects (classes or instances) that the text specifies
(explicitly or implicitly) as in some relation to each other. The columns are the relations,
and the attributes of the objects participating in those relations. Note that a pair of
objects may be in more than one relation to each other. If row \( i \) represents the ordered
object pair \( \{O1, O2\} \), and column \( j \) represents the relation \( R \), then a “yes” in cell \( \{i, j\} \)
means that \( O1 \) is in the directed relation \( R \) to \( O2 \).

The formal contexts of FCA can be translated into Sowa’s conceptual graphs (CG’s)
[1984] and vice versa.[Wille, 1997] Both CG’s and FCA have been used to represent and
process knowledge. Both are methods of formalizing logic, and relating logical concepts
to real-world concepts. Since CG’s were developed to represent the syntax and semantics of natural language, FCA can be used for the same purpose. The objects of FCA map into Sowa’s concept nodes. The relations map into the relations that connect Sowa’s concepts. These relations are generic rather than application-domain-specific, e.g., Sowa defines such relations as “agent” (AGNT), “patient” (PTNT), “location” (LOC), etc. Agents act upon objects, patients (not necessarily the medical kind!) are objects that are acted upon, and so on. Hence, in the above example, if John sued Mary, O1 could be John, O2 could be Mary, O1 would be the AGNT with respect to suing, the one who is doing the suing, and Mary would be the PTNT with respect to the suing, the one who is being sued. The context would show “yes” for the cell {John, AGNT}; it would also show “yes” for the cell {Mary, PTNT}. The context would also contain cells for attributes of John and Mary individually, e.g., John may be tall and antagonistic, Mary may be blonde, reasonable, etc.

An FCA formal concept is a pair \((A, B)\) where A is a subset of the objects in a context, and B is a corresponding subset of attributes applicable to all the objects in the concept. Eklund et al. [proposal, 1999] offer a very simple example of a context that describes the solar system. The objects, \(A\), are the planets, the attributes, \(B\), are size (small, medium, large), distance (near, far), and moon (yes, no). One of the concepts that FCA can discover is \({\{\text{Earth}, \text{Mars}\}, \{\text{small, near, yes}\}}\). A defining characteristic of the formal concept \((A, B)\) is that “A must be the largest set of objects for which each object in the set possesses all the attributes of B. The reverse must be true also of B.” [Cole et al., 1999] In other words, the set of attributes, \(B\), must be the largest set of attribute values that characterize the set of objects, \(A\).

The relationships among concepts and individuals that are captured in a formal context or a conceptual graph, can also be represented as a “concept lattice.” Each node of a concept lattice corresponds to the maximum set of objects that possesses a given attribute. This set of objects is called the extent; the number of objects comprising this extent may be attached to the node. (This set may also be viewed as the maximal concept associated with the given attribute.) If a parent node representing objects with attribute \(m_1\) branches to two child nodes representing objects with attributes \(m_2\) and \(m_3\) respectively, then the extent of the \(m_2\) node represents objects having both attribute \(m_1\) and attribute \(m_2\). Similarly, the extent of the \(m_3\) node represents objects having both
attribute $m_1$ and attribute $m_3$. Now, since a lattice is a more general graph than a tree, it is quite permissible for two nodes, e.g., the $m_2$ and $m_3$ nodes of the above example, to meet in a common node below. This common meet node represents the set of objects that possess attributes $m_1$, $m_2$ and (not or!) $m_3$. If a node having attribute $m_4$ is immediately below this meet node, various implications involving $m_2$, $m_3$ and $m_4$ (given $m_1$) can be represented. If the extent of $m_4$ is greater than or equal to the extent of the meet of $m_2$ and $m_3$, this represents “$m_2$ and $m_3$ implies $m_4$” (in other words, every object having both attribute $m_2$ and attribute $m_3$ is one of the objects having $m_4$). Similarly, If the extent of $m_4$ is less than or equal to the extent of the meet of $m_2$ and $m_3$, this represents “$m_4$ implies $m_2$ and $m_3$” (in other words, every object having attribute $m_4$ is one of the objects having both attribute $m_2$ and attribute $m_3$). If the extent of $m_4$ is exactly equal to the extent of the meet of $m_2$ and $m_3$, then there is equivalence between $m_4$ and the meet (intersection) of $m_2$ and $m_3$.

Partial implications (conditional probabilities) can also be represented. The probability of $m_1$ given $m_0$, denoted $Pr(m_1 \mid m_0)$, is $\text{extent}(m_1)/\text{extent}(m_0)$. If the $m_0$ node is the root of the lattice, representing the entire population of objects under consideration, then the implications described in the preceding paragraph are unconditional. However, one can still compute $Pr(m_1 \mid m_2)$ as $\text{extent}(\text{meet}(m_1 \text{ and } m_2))/\text{extent } (m_2)$. That is, the probability of an object having attribute $m_1$ given that it has attribute $m_2$, is computed as the quotient of the number of objects having both $m_1$ and $m_2$ (objects having $m_1$ given that they also have $m_2$) divided by the total number of objects having $m_2$.

Given that a rich context may involve a large number of objects and attributes, the user will normally want to focus on a more manageable subset. Cole et al. [Comp Int 1999] provide this capability. That is, the user can focus on certain specific objects and attributes of interest; the system will generate the concept lattice for the selected entities. Cole calls this a “scale.” As a further refinement, their system allows the user to generate a lattice involving one set of concepts within a lattice involving another set of concepts, i.e., to nest one scale within the nodes of the lattice of another scale.

For example, Cole et al. have applied FCA to the analysis of medical discharge summaries. In both cases, the objectives are to represent the relationships and
implications among the concepts. “The relationships will often be mundane, but occasionally surprising and new.” For example, they generate a concept lattice exhibiting relationships among classes and subclasses of disease, e.g., “respiratory tract diseases,” is a broad disease class, “asthma” is a disease subclass within that broader class. Another broad class may contain such subclasses as “carcinoma.” The lattice may indicate (via “meet” nodes) the co-occurrence of diseases, e.g., how many patients suffer from both asthma and carcinoma. Within each node of this “disease” lattice, another lattice may be nested, exhibiting, e.g., patient behaviors such as smoking, and drug therapies that have been applied. For example, the “smoking” node of the behavior lattice can be found in each node, e.g., the “carcinoma” node, of the disease lattice. The extent of the “smoking” node within the “carcinoma” node gives the number of patients diagnosed with carcinoma who smoke. By studying these lattices, relationships among diseases, patient behaviors, and therapies may be displayed and analyzed.

Recently, Cole et al. [KDD99] have focused on semi-structured text (XML-based and HTML-based). The FCA objects studied are text documents, e.g., email messages. The attributes are keywords and patterns found in the documents. The patterns are specified as regular expressions. For example, email attributes may include the originator and addressees (from and to lines), and a date condition, e.g., all dates in the range from September to November 1994. The attributes may also include keywords, e.g., names, mentioned in the textual body of the email document. Analysis of the conceptual lattice associated with an email collection leads to the discovery of patterns, e.g., that a high proportion of the emails addressed to a given addressee mention a given person or combination of persons by name.

2.6.3.2 Concepts and Discourse Structure

Many researchers have recognized that a document, especially a large document, may not be the ideal unit for matching against queries or topics. A document may deal with multiple topics. The matters of concern to a given user, or the key words that identify her interests, may be localized to a small portion of a document. Hence, a variety of research efforts, some of them described in this report, attempt to break documents into segments, often called “passages.” Sometimes, the boundaries of these segments are determined orthographically, e.g., on the basis of paragraph or section or sentence boundaries. In other cases, documents are segmented arbitrarily, e.g., by overlapping windows $N$
characters long. The former approach takes semantics into account, but only indirectly, by assuming that sentence, paragraph, or section boundaries specified by the author accurately reflect her intended semantic structure. The latter approach ignores semantics in favor of locality. Of course, it is likely that the words or sentences that occupy a local passage have some semantic relationship, but it is impossible to say a priori what that relationship will be.

Liddy et al. [Proc RIAO Conf., 1994] have taken a more principled approach by studying the discourse structure (based on “discourse linguistic theory”) of various types of documents, e.g., newspaper articles [TREC-2, 1994], or abstracts of empirical technical documents [Liddy, ASIS ‘87, Liddy, 1988]. A coherent, well-written document has a semantic structure that represents the way the author has organized the ideas or story she wants to tell. Moreover, textual documents of a particular type will have a predictable, standard structure. The elements of this structure are called “discourse components.”

Liddy has extended an earlier model due to van Dijk [Hillsdale, 1988] for the text type “newspaper article.” She has identified 38 discourse components in her extended model. Each clause or sentence in a given article can be tagged as one of these components. These tags “instantiate” the model. Assigning a tag to a clause says that the given clause belongs in the corresponding component of the model. Each component will contain certain kinds of information relative to the story told by the entire article. Examples of component tags for the newspaper article model are: MAIN EVENT, VERBAL REACTION, EVALUATION, FUTURE CONSEQUENCE, and PREVIOUS EVENT. Components may be nested, corresponding to nesting in the sentence structure. Linguistic clues are used to identify the components. For example, Liddy [TREC-2] offers the following example of nested, tagged discourse components.

\[
<\text{LEAD-FUT}> \text{South Korea’s trade surplus, } <\text{LEAD-HIST}> \text{ which more than doubled in 1987 to } $6.55 \text{ billion, } <\text{LEAD-HIST}> \text{ is expected to narrow this year to above } $4 \text{ billion, } <\text{LEAD-HIST}> \text{ is expected to narrow this year to above } $4 \text{ billion. } <\text{LEAD-FUT}>
\]

Plainly, this is a LEAD-FUTURE component about the expected future trade surplus of South Korea, as indicated by linguistic clues such as the phrase “is expected to,” containing a nested LEAD-HISTORY component about South Korea’s past trade
surplus, as indicated by linguistic clues such as the past tense “doubled” and the “1987” date.

Tagging the clauses and sentences of a document by discourse component allows Liddy to generate multiple SFC vectors, one for each component. This means that one can not only match the subjects found in a topic against the subjects found in a document; one can also determine whether they are in the correct discourse component. For example, if the topic required that a document discuss future trade surpluses in South Korea, it would be important not only that the subject appear in a given document, but that it appear in a FUTURE EVENT or LEAD-FUTURE discourse component. A document that has the right subject in the right discourse component should receive a higher relevance ranking score than a document that has the right subject in the wrong component. Liddy has identified 38 discourse components for the newspaper article text type. However, she has found that topic requests usually do not have so fine a discourse grain. Hence, she has improved the performance of DR-LINK by mapping the 38 components into seven meta-components for the purpose of topic-document matching and ranking: LEADMAIN, HISTORY, FUTURE, CONSEQUENCE, EVALUATION, ONGOING, and OTHERS. These seven meta-components yield eight SFC vectors, one for each component and one for the combination of all seven together. The resulting module that matches topics against documents using these eight SFC vectors is called the “V-8 SFC Matcher.”

Mann et al. [Text, 1988] have developed an alternative method of discourse analysis called Rhetorical Structure Theory (RST) [Mann et al., Text]. RST can be used for the automated markup and parsing of natural language texts [Marcu, AAAI, 96] [Marcu, PC, 1999]. Both Marcu and Eklund et al. [proposal, 1999] are exploring possible applications of RST to automated textual information extraction. Marcu is studying the application of RST to document summarization and machine translation of natural languages. Eklund et al. have considered its application to text data mining, knowledge base construction, and knowledge fusion across documents.

Mann et al. claim that RST “provides a general way to describe the relations [called rhetorical relations] among clauses in a text, whether or not they are grammatically or lexically signalled.”
Of course, automatic parsing as developed by Marcu, depends on recognizing just such grammatical or lexical signals, and using them to drive the actions of the parser. Marcu has taken two approaches to automated RST parsing. First, he has developed a set of manual rules. Second, he has applied a machine learning tool to a large text corpus to “learn” a set of parsing rules. Clearly, the success of such automated parsing depends on the text possessing a certain coherence and clarity typical of news article text, and well-written scientific and legal papers. The techniques might be much less successful if applied to informal text, e.g., e-mail; they have never been applied to such informal texts. [Marcu, PC, 1999]

A rhetorical relation links two clauses (non-overlapping spans), called the nucleus and the satellite. The significance of these terms is that most (though not all) of the relations are asymmetric. One clause, the nucleus, is usually more essential than the other. The less essential clause, the satellite, is sometimes incomprehensible without the nucleus to which it is related. Even where the satellite is comprehensible by itself, the nucleus is generally more essential for the writer’s purposes. Marcu capitalizes on this fact to generate automatic document summaries. The nuclear clauses, taken by themselves, form a coherent summary of the document’s essence. The satellites do not. Another indication that the satellites are less essential is that they are often substitutable, i.e., in a given relation one satellite can often be substituted for another while retaining the same nucleus. Mann and Thompson define 24 rhetorical relations, but stress that the set is open-ended. Marcu has discovered a considerably larger set of relations, but anticipates that other researchers may discover yet more relations, as they explore other classes of text. [Marcu, PC, 1999]

Marcu is also considering the possible application of RST parsing to machine translation (MT). One of the difficulties with existing MT is that even when words and phrases and even clauses are properly translated, the overall translation of the text may be awkward or incorrect. This is because the discourse structure in the RST sense may vary from one language to another, especially at the lower levels of the RST trees. Translating this structure may substantially improve the quality of the translation.

The parsing of a text according to RST identifies and marks up all the rhetorical relations, and the clauses participating in those relations. The rhetorical structure is defined recursively, i.e., one of the spans participating in a rhetorical relation can itself be
composed of rhetorical relations. Hence, the rhetorical structure of a text is a tree. Only the nodes of the tree are necessarily “simple” clauses. A given text can be parsed in multiple ways. Hence, a given text may be represented by multiple rhetorical parse trees. However, Marcu has defined principled rules for “legal” parse trees. If one adheres to Marcu’s rules, one can still generate multiple trees for a given text document, but it becomes possible to reject some candidate parses as ill-formed while accepting others as well-formed.

The following passage illustrates two asymmetric rhetorical relations: “concession” and “elaboration.”

> Although discourse markers are ambiguous,\(^1\) one can use them to build discourse trees for unrestricted texts;\(^2\) this may lead to many new applications in text data mining.\(^3\)

A *concession* relation exists between the nucleus, either 2 or 3, and the satellite, 1. The nucleus is asserted to be true despite the contradictory “concession” of the satellite. An *elaboration* relation exists between the satellite 3 and the nucleus, either 1 or 2. The satellite elaborates on the assertion made by the nucleus. Note that the same clause, e.g., 3, can be a nucleus in one instance of one relation, and a satellite in one instance of another relation.

The word “although” is a lexical marker for the *concession* relation in the example given above. Similarly, the semicolon is a marker for the *elaboration* relation in this example. However, Mann *et al.* stress that “the definitions [of the rhetorical relations] do not depend on morphological or syntactic signals … We have found no reliable, unambiguous signals for any of the relations.” Marcu’s relative success indicates that for well-structured text, reliable markers can be found for a fairly high proportion of instances of the relations. But Marcu’s studies have been limited to well-structured text types, e.g., Scientific American articles. Mann *et al.*, on the other hand, have studied a wide variety of types including “administrative memos, magazine articles, advertisements, personal letters, political essays, scientific abstracts, and more.” They claim that an RST analysis is possible for all of these diverse types. However, not surprisingly, they have found no simple linguistic markers that work for recognizing or delimiting these relations across all of the diverse text types. They also find that certain text types do not have RST analyses, including “laws, contracts, reports ‘for the record’
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

and various kinds of language as art, including some poetry.” A common characteristic of
the text types studied with some success by Marcu is that they are types of *expository*
writing. Hence, Marcu’s approach might work well with legal (judicial) opinions which
are typically expository, although according to Mann et al. as quoted above, it would
probably not work for laws and contracts, which are typically *not* expository.

Marcu measured the success of his automated parsers in terms of the classical precision
and recall measures. Recall measured the proportion of the rhetorical relations in a text
that were identified, i.e., the “coverage.” Precision measured the proportion of identified
relations that were identified correctly. These values were computed by comparing the
results of the automated parses against parses performed manually by human judges. It
was found that human judges could achieve a high degree of agreement in their
respective parses, thereby justifying the claims of RST for the text types studied.
However, it was also found that the human judges required a considerable amount of
training and practice before they could achieve this consistency. Parsing according to
the rules of RST is far from trivial.

Eklund et al. have proposed combining RST with FCA by generating formal contexts
whose objects are larger entities like clauses, sentences, and even documents.
Correspondingly, the relations would be rhetorical relations. If a simple formal context
based on RST is converted to a simple conceptual graph (CG), its nucleus (in the RST
sense) would become the “head” of the CG. Other text spans having RST relations to
the nucleus could then be used to build more complete CG’s. In this way, a CG
knowledge base could be constructed, extending over whole documents and fusing the
knowledge of multiple documents.

2.6.3.3 Proper Nouns, Complex Nominals and Discourse Structure

It is not sufficient to match topics against documents at the level of subject categories,
i.e., at the level of SFC’s. Much of the essential content of a topic request and the
corresponding content of documents, is found in proper names (PN’s), e.g., names of
persons, countries, companies, government agencies, etc. Much of the remaining
content is found in “complex nominals” (CN’S), e.g., noun phrases formed by adjacent
nouns, adjective-noun combinations, etc. In the passage about South Korea quoted near
the beginning of the preceding section, “South Korea” is an example of an essential
proper name (PN); “trade surplus” is an example of an essential complex nominal (CN)
noun phrase. Systems that recognize and extract CN’s and PN’s for use as document and topic descriptors include DR-LINK [Liddy et al., TREC-2], and Strzalkowski’s system [TREC-3] developed at NYU. The latter system augments a traditional statistical backbone “with various natural language processing components.”

Recognition and extraction of complex nominals (CN’s) involves several problems. [Strzalkowski et al., TREC-3] First, it is necessary to recognize CN’s in any of various syntactic structure, e.g., to recognize “information filtering” in “information filtering system,” “filtering of information from databases,” and “information that can be retrieved by...” It is necessary to distinguish cases where two-word CN’s are satisfactory, from cases where longer CN’s are necessary, e.g., in the phrase “former Soviet president,” “former president” and “Soviet president” have quite different meanings so the longer three-word phrase should be preserved. It is also important to resolve the ambiguity in parsing CN’s, e.g., to recognize that in the phrase “insider trading case,” the key head word is “trading” and its modifier “insider.” Phrases like “insider case” and “trading case” would be much less significant. Here, statistics must supplement pure syntactic analysis to ensure that the extracted phrase is semantically significant as well as syntactically correct. A statistical analysis of a corpus of business and economics stories will reveal that “insider trading” occurs far more often than the other pairs.

DR-LINK extracts CN’s with its “Complex Nominal Phraser.” Complex nominals are recognized “as adjacent noun pairs or non-predicating adjective + noun pairs in the output of the part-of-speech tagger.” DR-LINK’s PN recognition and categorization capability is described below in a subsequent section.

The problems of recognizing, extracting, and categorizing proper nouns, and the approaches to these problems taken by several research teams, is discussed in a later section.

DR-LINK not only extracts proper nouns (PN’s) and complex nominals (CN’s) from the topic statement, but also identifies the discourse component in which each should preferably be found. A discourse component in a given document is weighted on the basis of how many complex nouns and complex nominals from the topic statement it contains. This weight is then multiplied by another weight factor that reduces the total weight if the discourse component is not the one required by the topic statement, e.g.,
if the topic statement requires a noun phrase to be in a FUTURE component, but the phrase only occurs in the LEADMAIN component of a given document, the weight of that component will be reduced. If the FUTURE component also contains the given noun phrase, the FUTURE component will receive a higher weight than the LEADMAIN component. A CONSEQUENCE component that doesn’t contain the desired noun phrase at all will receive a lower weight than the LEADMAIN component. Hence, the PN/CN similarity score depends not only on the presence of specified proper nouns and complex nominals in a given document (conventional keyword matching), but also on the discourse components in which they occur (discourse text structure matching). In fact, it is properly called a PN/CN/TS score where the third acronym stands for Text Structure. Hence, both the SFC similarity score and the PN/CN/TS similarity score for a given document reflect the discourse structure of the document, and the discourse requirements of the topic statement.

Observe too that proper nouns are assigned semantic categories in DR-LINK [Paik, ARPA Workshop]; this categorization is discussed below. Hence, the PN similarity score of a document relative to a given topic may depend on a category match as well as a match on the actual proper noun itself.

2.6.3.4 Integrated SFC/PN/CN Matching

Plainly, a document that matches a topic statement not only at the level of subject categories (SFC’s) and the discourse components in which they occur, but also matches on proper nouns and noun phrases, should be ranked higher relative to the given topic statement than a document that matches only on the one or the other. DR-LINK’s “integrated matcher” combines these two kinds of matching. It takes as input two similarity scores: one based on SFC vector similarity, and one based on Proper Noun (PN), Complex Nominal (CN) similarity. An SFC cutoff or threshold score is computed. Documents are then ranked by a combined similarity score, with documents having a non-zero PN/CN similarity being ranked above those with a zero PN/CN score. Generally, documents below the SFC threshold are not returned, but a fraction of documents (depending on the recall level) with high PN/CN scores and documents below the SFC threshold are inserted above the zero PN/CN documents.
2.6.3.5 Relations and Conceptual Graph Matching

If a topic and a document match on the presence of two entities, there is a chance that the document is about the specified topic. If the two entities occur in close proximity in the document, the chance is better. If the topic requires that the entities occur in a certain discourse component, and the entities occur in the required component within the document, the chance of document relevance to the given topic is better still. But the assurance of relevance will be even stronger if the entities are related in the same way in both topic and document. Hence, DR-LINK also has the capability to identify relationships in a number of syntactic cases, e.g., noun phrases (NP’s), nominalized verbs (NV’s), prepositional phrases (PP’s), and the complex nominals (CN’s) already mentioned. In all cases where DR-LINK can identify a relation, it generates a concept-relation-concept (CRC) triple. This process not only identifies relations. It also converts varied syntactic forms into a single canonical form to simplify topic-document matching.

Once CRC triples have been generated, they are linked together to form larger units, e.g., clauses. Linking can occur because the same proper noun, e.g., a company name, occurs in two CRC’s. Or it can occur because of coreference resolution, e.g., a pronoun in one CRC is identified as referring to a given proper noun in another. The structures that DR-LINK forms by linking CRC’s are conceptual graphs (CG’s). CG’s [Sowa, 1984] are a graphical notation for representing the syntax and semantics of natural language. CG’s are also available in a linear textual form. The graphical nodes represent entities, and relationships among the entities. However, CG’s are more powerful than simple entity-relationship diagrams. They can represent first-order predicate logic, e.g., they can express quantification. CG’s have been discussed earlier with respect to Wille’s work on their equivalence to formal contexts, as defined in Formal Context Analysis.

Liddy offers an example of converting an NV into canonical CRC’s: The phrase “the company’s investigation of the incident” is converted into:

[investigate] -> (AGENT) -> [company]
[investigate] -> (PATIENT) -> [incident]

Note that the “nominalized” form “investigation” has been converted into the standard verb form “investigate.” This simplifies matching with another phrase in which the verb form occurs. The relations “AGENT” and “PATIENT” are standard CG relations. The
AGENT is the entity (also called the “actor”) who performs the action, in this case investigates. The PATIENT is the passive entity that is the object of the action, in this case the entity that is being investigated.

Once CG’s have been generated, they are made more “conceptual,” by replacing entity nodes by codes representing the concepts of which the entities are instances. In TREC-2, DR-LINK used the Roget International Thesaurus (RIT) codes. (Current versions of DR-LINK, use WordNet Synsets. [Liddy, PC]) Finally, CG’s in topics can be matched against CG’s in documents. [Myaeng et al., JITAE, 1992]

2.6.3.6 Recognition of Semantic Similarity in CN’s

Strzalkowski [IP&M, 1994] [TREC-2] [TREC-3] uses statistics not only to choose semantically significant head-modifier pairs from ambiguous CN’s. He also use it to identify clusters of words that are semantically related, and hence are candidates for use in query expansion. Two terms are candidates if they occur in the same contexts, e.g., as head nouns with a number of common modifiers, or as modifiers of a number of common head nouns. For example, “man” and “boy” are modified by a number of the same modifiers (appear in a number of the same “contexts”) in the corpus studied, including “bad,” “young,” and “older.” That is, the corpus contains a number of references to “young men” and “young boys,” “older men” and “older boys,” etc. The same two words, “man” and “boy,” also serve as modifiers of a number of the same head nouns, including “age” (references to “man’s age” and “boy’s age), “mother” (references to “a man’s mother” and “a boy’s mother”), and so on. Hence, “man” and “boy” appear in the same contexts both as head nouns and as modifiers.

Several additional factors must be considered to identify two terms as semantically similar. First, the terms should appear in few contexts other than the ones they share. Second, the shared contexts must not be too common, e.g., “natural” is too common a term to justify predicting similarity of “logarithm” and “language” on the basis of the shared contexts, “natural language” and “natural logarithm.” Third, the number of distinct shared contexts must exceed some threshold; this threshold depends on how narrow or broad the training corpus is, e.g., a Wall Street Journal corpus is broader than a Communications of the ACM (CACM) corpus. Hence, “banana” and “Baltic” may share the context “republic” a number of times, but their similarity is rejected because they share no other context. Similarly, “Dominican” and “banana” share two contexts,
“republic” and “plant,” but in a broad corpus, this is still not enough. A still more striking example is “pharmaceutical” and “aerospace” which are not semantically similar despite being found to share more than six common contexts: “firm,” “industry,” “sector,” “concern,” etc. Here the commonness, and hence relative unimportance, of the shared contexts must outweigh the mere number of shared contexts. Strzalkowski et al. also observe that modifiers are more reliable contexts than head terms; hence, in totalling the number of shared contexts for a term, they count a head context as only 0.6 of a context.

A final problem with Strzalkowski’s statistical clustering of terms with shared contexts is that it does not distinguish similarities that indicate synonyms (“merge,” “buyout,” and “acquisition”), and specialization, which generate terms suitable for query expansion, from complements (“Australian” and “Canadian”) and antonyms (“accept” and “reject”) which are generally not suitable for query expansion. Strzalkowski addresses this by defining a “Global Term Specificity” (GTS) measure. The GTS is roughly analogous to the \( idf \) but is measured over syntactic contexts rather than documents. Moreover, it is only useful for comparing terms that are already known to be similar in terms of context co-occurrence. For such contextually similar terms \( w_1 \) and \( w_2 \), the assumption is that \( w_1 \) is more specific than \( w_2 \) if it occurs in fewer distinct contexts; if \( w_1 \) and \( w_2 \) occur in the same number of distinct contexts, but \( w_1 \) occurs in many more instances of those contexts than \( w_2 \), it may be more specific. GTS is given by:

\[
GTS = \begin{cases} 
IC_L(w) \cdot IC_R(w) & \text{if both exist} \\
IC_R(w) & \text{if only it exits} \\
IC_L(w) & \text{otherwise}
\end{cases}
\]

where (with \( n_w, d_w > 0 \)):

\[
IC_L(w) = \frac{n_w}{d_w( n_w + d_w + 1 )} \quad \text{head is context}
\]

\[
IC_L(w) = \frac{n_w}{d_m( n_m + d_m + 1 )} \quad \text{modifier is context}
\]
Here, $d_w$ is the number of distinct contexts in which $w$ occurs (as a modifier for $IC_L$, as a head for $IC_R$), and $n_w$ is the number of actual occurrences of $w$ in these contexts. If $GTS(w_1)$ is greater than or equal by some appropriate factor than $GTS(w_2)$, then $w_1$ is assumed to be more specific than $w_2$. If $GTS(w_1)$ is less than or equal to $GTS(w_2)$ by some appropriate factor and vice versa, then the terms are assumed to be synonymous. Hence, this process leads to clusters of terms that are predicted to be either synonyms, or in the relation that one term is a specialization of the other. These clusters can be used either for automatic query expansion or interactively, to suggest candidate expansion terms to a human user.

DR-LINK also uses statistical techniques to identify terms that are likely to be interchangeable in certain CN contexts, specifically terms that are premodified by the same set of terms in a given corpus. If two terms $a$ and $b$ are premodified by the same set of terms, there is said to be a “second order association” between $a$ and $b$. Hence, this process identifies phrases that are substitutable for each other for purposes of document topic matching.

### 2.6.4 Proper Noun Recognition, Categorization, Normalization, and Matching

The presence of specified proper nouns is often a necessary, though not necessarily sufficient, condition for a document to be relevant to a specified topic. If the topic is the Japanese stock market, then some form of the proper noun “Japan” is clearly essential, although by itself hardly sufficient, since documents might deal with many other aspects of Japan. Names of persons, companies, government agencies, religions, chemicals, and many other entities may be essential to the specification of a topic, and the recognition of documents relevant to the given topic.

Recognition, extraction, and matching of proper nouns is considerably more complex than it might at first seem. A variety of factors complicate the process. Many proper nouns consist of more than one noun, e.g., “Wall Street Journal.” Many proper nouns include a preposition, e.g., “Department of Defense;” or a conjunction, e.g., “John Wiley and Sons.” Many proper nouns can be specified in multiple forms, e.g., “MCI Communications Corp.,” “MCI Communications,” and “MCI.” Many proper nouns are group nouns, which may result in references either to the group as a whole, or to the
individual entities making up the group, e.g., “European Community,” “Latin America.” Common nouns and noun phrases may also group individual entities that have proper noun names, e.g., western nations, socialist countries, third world, agricultural chemicals.

Borgman et al. [JASIS] discuss at length the particularly difficult case of names of persons. Conventions for assigning names vary with the culture and historical period. In ancient times, single names were normal. The practice of assigning multiple names, e.g., first, middle, and last names, is more recent. Some cultures use compound surnames, but the conventions vary from one culture to another, e.g., “[h]ispanic children receive a combination of their parents’ surnames, and wives acquire a combination of their maiden surnames, and their husbands’ surnames.” Order of names also varies with culture, e.g., “[a]sians traditionally place the surname first, although asians living in Western nations often report their names with surnames last.” “Personal names may be translated from one language to another, retaining meaning,... or be transliterated from one alphabet or character set to another.” Multiple transliteration schemes exist. People change their names over their lifetime, as a result of marriage, divorce, adoption, or movement from one country to another. People adopt or receive nicknames and diminutives, e.g., “Dick” for “Richard,” “Bob” for “Robert.” A person may use one form of her name on a drivers license, but another form for publication as an author. On top of all this, errors are common, not only typographical errors, which affect any typed input, but phonetic errors, e.g., a person from one cultural or linguistic background transcribing a spoken name from another cultural or linguistic background is especially likely to err.

Paik et al. [ARPA Workshop] [Corpus Proc] have developed a sophisticated series of procedures for proper noun recognition and matching in their DR-LINK (Document Filtering through LINguistic Knowledge) and KNOW-IT (KNOWledge base Information Tools) IF engines. The proper noun recognition system described here was developed through corpus analysis of newspaper texts. First they assign parts of speech to all the words in the document; then they execute a general purpose noun phrase bracketter, and a special-purpose proper noun phrase boundary identifier. Next the system categorizes all the proper nouns; this is consistent with the DR-LINK emphasis on capturing the conceptual level of a document, as well as the actual keywords and phrases. Topic requests may often be stated at the conceptual level. As Liddy et al. note, “queries about government regulations of use of agrochemicals on produce from abroad, require
presence of the following proper noun categories: government agency, chemical, and foreign country.” Note that a document that contains proper nouns in those categories may not actually contain the words “government,” “agrochemicals,” “produce,” or “abroad.” DR-LINK attempts to recognize eight categories: Geographic Entity, Affiliation, Organization, Human, Document, Equipment, Scientific, and Temporal; within each of these categories, DR-LINK recognizes two or more subcategories, for a total of 29 meaningful subcategories. (A more recent version, embodied in both DR-LINK and another commercial tool, KNOW-IT, recognizes over 60 subcategories.) “Affiliation” includes “religion” and “nationality.” “Human” includes “person” and “title.” “Scientific” includes “disease,” “drug,” and “chemical.” And so on. DR-LINK performs this categorization using such clues as known prefixes, infixes, and suffixes for each category, e.g., Dr., Mr., Ms., and Jr. for persons, Inc. and Ltd., for companies, etc. DR-LINK also uses a database of aliases for alternate names of some proper nouns, and knowledge bases such as gazetteers, the CIA World Factbase, etc. Contextual clues are also used, e.g., if the pattern proper noun, comma, proper noun is encountered, and the second noun has been identified as a state, the first noun (if not otherwise categorized) will be categorized as a city.

Since a given proper noun may take multiple forms, DR-LINK standardizes proper nouns as they are being categorized. That is, all forms of the same proper noun are mapped into a single standard form, to simplify subsequent matching. This is equivalent to stemming of ordinary words, reducing all variants to a common form. However, whereas stemming (at least in English!) largely involves processing of multiple suffixes, standardizing of proper nouns involves standardizing of prefixes, infixes, suffixes, and variant forms of proper nouns, e.g., “Dick” to “Richard.” Note that two variant forms of the same proper noun, referring to the same entity, may occur not only in two different documents, or in a document and a topic request, but also within a single document. In particular, an entity may be named in full on its first reference, and mentioned in a more abbreviated form on subsequent references. It is an important instance of reference resolution for a Natural Language Processing (NLP) based IF system to recognize that these are references to the same entity.

DR-LINK also expands group proper and common nouns, so that a topic request can match a document on either the group name or its constituents. For example [Feldman,
ONLINE], a request for documents about “African countries which have had civil wars, insurrections, coups, or rebellions” will return not only documents that contain some form of the proper noun “Africa,” but also documents containing references to countries within Africa. DR-LINK uses proper noun and common noun expansion databases.

Note that, in a system like DR-LINK, proper nouns can provide several levels of evidence for topic-document similarity computations. First, there is the obvious matching on the names themselves. Second (as noted earlier), there is matching on categories assigned to the names. This category matching is similar to, but supplements, the matching on subject categories (SFC’s) described in an earlier section. Third, expansion of group nouns can result in matching a document on proper nouns not actually mentioned in the topic statement. This can work in the other direction too, e.g., if a document mentions Montana or Atlanta, then these references may be used to match the document against a topic that only speaks about “American” companies. Fourth, proper nouns naming geographical entities can provide relationship information, e.g., they can “reveal the location of a company or the nationality of an individual.” Subject information can be combined with proper noun category information for more refined topic-document matching. A report of a merger should involve (at least) two proper nouns of category “company,” while a report of an invasion is likely to specify two geographic entities, most likely at the level of country or province.

A significantly different approach to proper noun recognition is taken by Mani et al. [Corpus Proc, 1996] Their approach differs somewhat both in goals and methods. They focus on a much smaller set of subject categories: people, products, organizations, and locations. Within large text corpora, they seek (like Paik) to categorize previously unknown names automatically. However, they attempt to go further than Paik, extracting from the text appropriate semantic attributes for each named entity, e.g., the occupation and gender of a person. A given entity may be mentioned more than once in a given document, and each mention may employ a different variation of the entity’s name, e.g., “President Clinton,” “Bill Clinton,” “Clinton,” “the president,” etc. They seek to “unify” these mentions, i.e., to recognize all mentions to the same entity, and to combine the attributes associated with these varied mentions into one common schema describing the given entity. This is called “coreference resolution” for proper nouns.
When two mentions (and their associated attributes) are successfully unified as referring to the same entity, they are said to be “coanchored.” Note that this goes considerably beyond (although it includes) the normalization of proper nouns performed by Paik.

Coreference resolution is closely tied to attribute extraction. On the one hand, attributes extracted from one mention of a given entity can be combined with attributes extracted from another mention, to fill out as many of the “slots” associated with the given type of entity as possible. For example, one mention may indicate that Clinton’s occupation or title is “president.” Another mention may indicate that his gender is “male.” On the other hand, extracted attributes can serve as evidence to determine whether two mentions refer to the same entity, or to two distinct entities. For example, if “President Clinton” has been associated with the gender attribute value “male,” and “Hilary Clinton” has been associated with the gender attribute value “female,” this is evidence that these mentions do not refer to the same entity. But, “President Clinton” and “Mr. Clinton” will match on gender, and hence will be coreference candidates unless additional evidence indicates a contradiction. Moreover, attributes may also serve to indicate whether two mentions refer to distinct but related entities. For example, “Bill Clinton,” “Hilary Clinton,” and “the Clintons,” are distinct, but related entities. As a further refinement, Mani distinguishes between “discourse pegs,” i.e., entities that are distinct in a given discourse, and entities that are distinct in the real world. For example, President Clinton, and ex-Governor Clinton may be two distinct discourse pegs for purposes of analyzing a given document, although they refer to the same real-world object in the world model or belief system of an external knowledge base.

As Mani encounters new proper noun mentions in the text of a given document, he naturally wants to limit the number of earlier mentions that must be evaluated as possible candidates for coreference. He does this by indexing each mention by normalized name (a standardized form, analogous to Paik), by name elements in its name (individual words within the name), and by its abbreviations. Only mentions that match on at least one of these indexes are coreference candidates. Abbreviations are generated by rule, or retrieved from a lexicon; hence, a full name in one mention can be matched against an abbreviated name in another.
Another difference between the Mani and Paik approaches is that Mani makes greater use of the context surrounding a proper noun, and of the discourse structure of successive mentions. In particular, Mani makes use of both honorifics and “appositive phrases,” phrases adjoining and identifying a proper noun. It is a widely used convention, especially in news stories, to attach an honorific or an appositive phrase to the first mention of a given name, e.g., “Anthony Lake, Clinton’s national security advisor,” or “Osamu Nagayama, 33, senior vice president and chief financial officer of Chugai,” or “German Chancellor Gerhard Schroeder.” Such appositives and honorifics are generally employed whenever the named entity is not a “household name,” and is not sufficiently identified by title. It is applied to entities other than persons, especially organizations and locations, e.g., “X, a small Bay Area town.” (Paik indicates that one of their intended research directions is the use of appositive phrases. However, in one knowledge base derived by KNOW-IT from New York Times articles, Anthony Lake was erroneously categorized as a body of water, presumably because the appositive phrase was ignored or misinterpreted.) Mani identifies candidate appositive phrases by pattern matching based on left and right delimiters such as commas and certain parts of speech. Syntactic analysis is then used to extract key elements, e.g., a head or premodifier, from the given phrase. In the “Nagayama” example above, “senior vice president” would be extracted, and looked up in a semantic lexicon ontology, which identifies the title as a “corporate officer.” Plainly, the value of such appositive phrases for categorization depends on the availability of lexicons that enable one to interpret their semantic content.

Another distinctive feature of the Mani methodology, closely related to the gathering of evidence over multiple mentions of an entity, is the explicit handling of uncertainty. Evidence gathered in one mention can reinforce or contradict evidence gathered in another mention. Mani employs a variety of Knowledge Sources (KS’s). KS’s are little rule-based programs that attempt to categorize (“tag”) entities. Many of the rules employed by Mani’s KS’s are similar to the rules employed by Paik’s system, e.g., one KS attempts to identify organizations by using suffixes such as “Inc.” and “Ltd.” Another tries to identify persons by looking for titles and honorifics, e.g., “Mr.,” “Lt. Col.”, “Ms.”, etc. Other KS’s use lexicons, e.g., organization lexicons, gazetteers as geographic lexicons, etc. On the basis of the evidence it collects, a KS can generate multiple hypotheses with different confidences. Mani offers the example that “General
Electric Co.” may generate one hypothesis that the entity named is a person, with “General” as a title, while other hypotheses may be that it is an organization or a county, based on the abbreviated suffix “Co.”. On the other hand, multiple KS’s may generate the same hypothesis based on different evidence, e.g., one KS may hypothesize that the given mention is an organization based on the “Co.” suffix; another KS may generate the same hypothesis based on the presence of the name in an organization lexicon. A “Combine-Confidence” function computes the confidence of a given hypothesis about a given mention as the weighted sum of the probabilities assigned to the hypothesis by each KS that contributed to it, each probability weighted by the reliability of the KS that generated it.

The confidence values associated with hypotheses play an important role in mention unification. If two person mentions have conflicting hypotheses about the occupation slot, but one hypothesis has a much lower confidence than the other, unification may succeed. On the other hand, if two mentions have conflicting gender hypotheses, and these hypotheses both have high confidence values (e.g., based on the honorifics “Mr.” and “Mrs.” respectively), the unification will fail.

2.6.5 Semantic Descriptions of Collections

In a later section, the fusion of IF results from multiple collections is discussed. However, all the cases discussed there assume that the set of collections to be accessed is known, preferably in advance. If the set of collections is very large, diverse, and dynamic, e.g., the Internet, this assumption no longer holds in general. In such cases, IF becomes a two-stage process, i.e., first find an appropriate set of collections, and then apply IF techniques such as those discussed in this paper. The process of finding the “right” collections becomes more manageable (though far from trivial) if each candidate collection includes, or is assigned, a formal machine-readable description of its contents. (The problem of searching and indexing the Internet or a large Intranet in the normal case where such standardized formal descriptions are not available, is discussed in a later section.) The first stage of the IF process then becomes the matching of a given query against a “collection” consisting of these collection descriptions. Chakravarthy and Haase [SIGIR ‘95] explore a case where structured collection descriptions (they call the collections “archives”) with semantic content are created manually using WordNet, and then natural language queries are translated automatically (using syntactic/semantic
techniques, an on-line Webster’s dictionary, and WordNet again) into a structured form that can be matched against the archive descriptions. They report that their system, NetSerf, has a “database” that currently contains descriptions (they call them “representations”) of “227 Internet archives. Most of these are from two sources, the Whole Internet Catalog [Krol, 1992] and the Internet Services List [Yanoff, 1993].”

An archive description (“representation”) consists of <relation-type, relation-word> pairs. Relation types illustrated by the authors in their examples include: TOPIC, INFO-TYPE, OBJECT, AUTHOR, PERTAINS-TO, IN, and HAS-OBJECT. “For each relation-word, NetSerf uses WordNet to identify all its synsets.” The relation word is disambiguated by the synsets containing the word that are not chosen. Chakravarthy and Haase offer the example of the World Factbook archive, described in natural language as, “World facts listed by country.” The TOPIC is “country,” and the INFO-TYPE is “facts.” Three of WordNet’s four synsets are assigned to the relation-word, “country:”

SYNSET: {nation, nationality, land, country, a_people}
SYNSET: {state, nation, country, land, commonwealth, res_publica, body_politic}
SYNSET: {country, state, land, nation}

A fourth synset of “country,” [rural area, country] is omitted since it corresponds to a sense of the word “country” that is obviously inapplicable here.

The authors plan to explore in the future the automatic construction of such descriptions from sources such as home pages and README files.

Query processing in NetSerf starts with a natural language query. “The query processor makes the assumption that the query, after preprocessing, consists of one or more topic words followed by prepositional phrases and verb clauses that modify either the topic words or preceding modifiers.” Manual rephrasing is sometimes necessary, e.g., where the original query takes the form of two sentences. The query is then “tagged,” i.e., a part of speech is assigned to each word or other lexical “token.” Common query introductions such as “What is” are deleted. Words or phrases identifying the leading information type are extracted, e.g., given the query “satellite photographs of hurricane’s progress,” the information type “satellite photographs” is extracted. Topic words and modifiers are extracted and cast into <relation-type, relation-word> form. The relation type is determined by the syntactic type of the modifier. A word sense
disambiguator based on neighboring relation words in the original textual query, WordNet hypernyms, etc., is executed. Finally, the main topic relation words are expanded “using semantic relations from the dictionary” that are “extracted using a pattern definition language.” For example, given the topic relation-word “pub” and its dictionary definition, the query processor generates the <relation-type, relation-word> pair: <PERTAINS-TO, “alcoholic beverage”>.

NetSerf queries are matched to archive representations. Query relation-words are matched against archive description relation-words. A “hit” occurs “if some valid synset of some relation word in R [the archive representation] is a hypernym of some valid synset of some relation word in Q. [the NetSerf query]” A positive weight is added for every hit where the relation types match. A negative weight is added for every hit where the relation types do not match.

Chakravarthy and Haase found that “structured representations [of archives, i.e., the archive descriptions], and semantic knowledge-based matching lead to significant improvements.” On the other hand, sense disambiguation led to a slight degradation of performance.

2.6.6 Information Extraction

One of the most important areas of IF where NLP plays a crucial role is information extraction (IE). IE is the extraction of information from a collection of documents in response to a query.

IE must be clearly distinguished from document filtering (DR), and document summarization (DS). DR, the focus of much of the research described in this report, is the filtering of documents relevant to a given query or topic. The document set retrieved may be relevance ranked or not. Either way, what the user receives is a set of documents believed to be relevant to the user’s need. DS is similar to DR except that the system generates a summary of each retrieved document. This summary may be a few sentences or paragraphs (perhaps modified syntactically achieve greater reading “smoothness” and “continuity”) believed to capture the essence of what the given document is about, or a set of key words believed to suggest the document’s essence. In either case, the filtering is document-based. Indeed, the distinction between DR and DS systems is often not clear cut. A DR system seldom returns a list of documents
directly. Rather, it typically returns a list of document identifiers, perhaps accompanied by relevance scores. These identifiers may be titles, subject lines, summaries, etc. The user can then request the actual text of a given document by selecting its identifier. On the other hand, a DS system returns a list of summaries. The DS system may allow the user to expand on the summary by requesting the document from which the summary was extracted.

By contrast, an IE system returns to the user information (not a document list) responding to the user’s information request (query). The response may be generated from multiple documents, or from a combination of a document and a database entry. Moreover, note that the term is information extraction, not text extraction. This implies that the “answer” generated by the IE system does not necessarily consists of text literally extracted from the document, with only minor syntactic tinkering if any. Instead, the answer may be (as with the systems participating in the Message Understanding Conference MUC [Darpa, 1992] competition) a template in which slots have been filled in. The template is usually generated manually in advance of the IE competition, or execution of an IE application. The names of the template fields do not necessarily appear in any document from which relevant data is extracted. The IE system must “understand” (typically using simple linguistic clues) that a term appearing in the text of a document, e.g., a human or corporate name, is an appropriate value for a given field of the template, and “fill in” the value of the field with the extracted value. Hence, the answer provided to the user is a mixture of manually generated template names, and extracted text values. Moreover, if a person is identified from information in a text document, the template may be filled in with a combination of information about the given person extracted from the given document, and additional information extracted from an entry about the person in a structured database.

Alternatively, the answer returned to the user may be textual, e.g., a sentence or paragraph. However, the sentence may be a combination of extracts from two sentences in the original document, linked by co-reference resolution. For example, the first sentence may identify a person by name and title. The second sentence may refer back to the individual by pronoun, e.g., “he announced that ….” Hence, the textual answer generated and returned to the user never actually appears explicitly in the text of the given document.
Note that, even in an IE system, e.g., KNOW-IT [Liddy, WP, 1999], the user may be given the option of going back to the original document(s) from which the answer was extracted.

Ideally, an IE system would “understand” the documents it reads (just as a human library researcher would), extract and condense all the information relevant to the user’s request, and return a single, coherent, comprehensive answer. In actual fact, such a strategy is far beyond the state of the art. The time required to compile the knowledge and logic required for such a level of processing, even for a relatively narrow subject domain, would be prohibitive. Moreover, such a level of logical analysis would be far too elaborate and slow for processing the huge document collections available today in libraries and on the Internet.

Hence, many IE systems use statistical part-of-speech tagging followed by “shallow parsing,” (also called “partial parsing”). [Cowie et al., 1996] The term “shallow parsing” refers to high speed parsing techniques in which only key fragments of a sentence are parsed. The phrase “shallow knowledge” refers to relatively simple ad hoc rules, often tailored to the needs and characteristics of a given domain, and supplemented by large lexicons, machine readable dictionaries (MRD’s) and other online reference sources.

Part-of-speech tagging is the process that assigns to each word in a sentence the grammatical role, e.g., noun, verb, adjective, determiner, etc., that the given word plays in the given sentence. This role is called its “part of speech.” “[M]ost English words have only one possible part of speech [but] many words have multiple possible parts of speech and it is the responsibility of a tagger to choose the correct one for the sentence at hand.” [Charniak, 1997] For example, the word “can” can be a modal verb, noun, or verb. Statistical parsers are typically trained on a tagged corpus. In the simplest form, the parser will simply know which part of speech is most common for the given word in the training corpus. A more sophisticated statistical tagger uses context, e.g., knows which part of speech is most probable for a given word when the word occurs following some other part of speech. For example, taken by itself, the most probable part of speech for “can” is the modal-verb. However, if the word “can” follows the determiner “the,” the part of speech “noun” becomes far more probable.
Shallow parsing allows an IE system to “skim” over a sentence, only parsing the most critical fragments, rather than generating a complete parse tree for the entire sentence. Parsers that attempt to parse a sentence fully “typically operate in polynomial time and tend to get bogged down with sentences containing more than 20 to 30 words.” [Cowie et al. 1996] Moreover, complete parsers generate a great many legal parses for a single sentence of significant length. For many IE tasks, the output of a partial parser is quite adequate, identifying critical subjects, objects, proper noun categories, etc.

Shallow knowledge is domain-specific knowledge, typically consisting of ad hoc rules that work in, but perhaps only in, the given domain. As an example of how narrowly tailored these IE rules can be, consider these examples from the U.Mass/MUC-4 [MUC-4] [Cowie et al., 1996] system for extracting events in the domain of Latin American terrorism (the rule numbering is arbitrary):

Rule 1: The direct object of “robbed” (active voice) is the victim of a robbery.
Rule 2: The subject of “disappeared” (active voice) is the victim of a kidnapping.
Rule 3: The object of “in” after traveling (active voice) is the target of an attack.
Rule 4: The subject of “hurled” passive voice is the instrument of an attack.
Rule 5: The subject of “placed” is the instrument of a bombing.

The rules were evidently derived from the text corpus. Rule 1 is fairly general, and might apply outside of the target domain. On the other hand, the other rules would obviously break down badly outside the intended domain. Consider the use of Rule 4 in a “baseball” domain! Rules 3 and 5 are even more specialized. Obviously, in most domains, traveling in a vehicle carries no implication about an attack at all, let alone who the target of the attack is. Similarly, in most domains, objects can be “placed” without any implication that they are bombing instruments. It is clear that the use of such rules requires a two-step process. First, IF techniques must be employed to locate the documents (or passages within documents) that are likely to be about terrorism. Only when the corpus has been narrowed down in this way do rules such as those above stand any chance of working. Yet within the intended domain, such naive rules have been found to work fairly well.

Note that because ad hoc shallow knowledge is only useful in the specific domain for which it was developed, it is usually the case that it cannot be re-used in another domain.
Therefore, the development of a shallow knowledge base makes sense only if it can be generated so rapidly and inexpensively that it can be treated as “a disposable artifact.” In other words, the assumption underlying the shallow knowledge approach is that it is easier and cheaper to create a shallow knowledge base for each new domain that comes along, than to create a deep knowledge base that can be re-used for many domains.

The opposite extreme is represented by the Cyc project [Lenat et al., 1989], the goal of which is to create a huge KB of commonsense knowledge of the world, the kind of knowledge that is not explicitly represented in encyclopedia and other reference books because it is knowledge that “everybody knows,” knowledge that is taken for granted, but knowledge that expert systems and IE systems tailored to a specific domain do not possess. Cyc research has demonstrated that building such a KB is a very long-term, expensive, difficult affair. It should be noted that the Cyc approach has been to enter knowledge manually, although the hope has always been that Cyc would reach a critical mass at which it could begin acquiring knowledge automatically from large textual sources.

The KNOW-IT system represents an intermediate approach to IE. It is not tailored to any particular application or knowledge domain. Like its technical and commercial relation, DR-LINK, it supports a broad hierarchy (60+) of proper noun categories, in a hierarchy eight levels deep. Similarly, it supports generic semantic relationships, such as “affiliation, agent, duration, location, or point in time,” as opposed to relations specific to a particular domain such as terrorism, e.g., relations “such as ‘weapons used’ or ‘victim’.” The KNOW-IT approach takes advantage of the “common practice among writers of including predictable information rich linguistic constructions in close proximity to related proper names.” Hence, KNOW-IT identifies and categorizes proper nouns, then identifies generic relationships among the concepts embodied by those proper nouns, so-called Concept-Relation-Concept (CRC) triples. (In common with most other IE systems, KNOW-IT also performs part-of-speech tagging to all the words in the text, assigning one of 48 possible grammatical tags, such preposition, determiner, or singular noun.) The user can display the concept structure graphically, penetrate from higher to lower levels of the concept hierarchy, until actual proper nouns are reached. She can also display the relationships in which concepts or proper nouns participate,
the documents in which those proper nouns and relationships occur, the sentences in which they occur, and finally, the full text of the documents in which the sentences occur. DR-LINK and KNOW-IT were originally developed for newswire text, but KNOW-IT has been extended to document types as diverse as technical manuals and WWW home pages.

The methods used by KNOW-IT for proper noun recognition and categorization are those developed by Paik [1993] for the DR-LINK project, as described earlier in the section on proper noun techniques. The CRC triples are the same as those widely used in Conceptual Graph studies, as developed by Sowa and others. CG’s are discussed earlier in the section on formal concepts, and again in the section on relations and conceptual graph mapping. The former discusses Wille’s work on the equivalence of CG’s and formal contexts in Formal Context Analysis (FCA). The latter discusses the work on CRC triples and CG’s in the DR-LINK system, closely related to KNOW-IT.

Although the basic KNOW-IT approach is domain-independent, it can and in some cases has, been extended with knowledge of some specialized domain, e.g., international politics.

2.7 Clustering

“Clustering” of documents is the grouping of documents into distinct classes according to their intrinsic (usually statistical) properties. Clustering is a kind of classification but it differs from the classification for routing purposes discussed in the section above on routing in one crucial respect: In a routing application, the documents are classified in terms of their similarity or relevance to external queries or topics or user profiles. In “clustering,” we seek features that will separate the documents into natural groups based entirely on the internal properties of the collection. Ideally, the groups will be completely separate and as far apart as possible in feature space. But sometimes, overlap of clusters is unavoidable. [van Rijsbergen, 1979] Since clustering depends on the statistical properties of the collection being clustered rather than on matching the documents against some external set of queries, it is normally (but not always see below!) applied to a pre existing collection rather than an incoming stream of documents as in a routing application.
CHAPTER-2: APPROACHES TO INFORMATION FILTERING

Why should documents be clustered? The basic reason is that clustering can reveal the intrinsic structure of a collection, e.g., by topic, subtopic, etc., (assuming of course, that there is a significant internal structure). If a language-independent statistical method such as “n-grams” is used, a collection may also be clustered by language or document type, by topic within language, etc. (See section 3.3.6.) Moreover, by the “cluster hypothesis,” “closely associated documents tend to be relevant to the same requests.” [van Rijsbergen, 1979] Document clustering of a large collection is particularly effective when it is hierarchical, i.e., when the collection is partitioned into (relatively) large, high-level clusters corresponding to broad categories, each high-level cluster in turn clustered into smaller clusters corresponding to tighter, more cohesive categories, which in turn are composed of still smaller, still more cohesive clusters, and so on. Ideally, the lowest level clusters in such a hierarchy will consist of documents that are very similar, e.g., that are all relevant to most of the same topics or queries. Hence, clustering, especially when combined with modern graphical display techniques, can be an effective tool for browsing a large collection and “zeroing in” on documents relevant to some given topic or other criterion. For similar reasons, it can increase the effectiveness of document filtering, i.e., of querying large collections. [Willetts, IP&M, 1988]

Searching a hierarchically clustered collection can proceed either top-down or bottom-up. Top-down searching proceeds as follows:

A top-down search of the cluster hierarchy is performed by comparing (using a similarity measure) the query to cluster representatives [e.g., centroids] of the top-level (largest) clusters, choosing the best clusters, comparing the query with representatives of lower-level clusters within these clusters, and so on until a ranked list of lowest-level clusters is produced. The documents in the top-ranked [of these lowest-level] clusters are then ranked individually for presentation to the user. [Belkin & Croft, ARIST, 1987]

The biggest problem with a top-down search is that the highest-level clusters may be so large and loosely coupled that a representative of such a cluster may bear little resemblance to most of the documents in the cluster. Hence, choosing a cluster at the highest levels becomes almost arbitrary and the search procedure is likely to choose a search path that misses the relevant documents. Hence, top-down searches work best when the clustering method and associated threshold ensure that even the highest-level clusters are reasonably small and cohesive.
A bottom-up search is the inverse of a top-down search. If one starts at the bottom, the clusters should be smaller and more cohesive than at the top. The problem is which one of those bottom-level clusters to choose as a starting point. One approach is to do a conventional query search to find one relevant document. Then one can start the cluster search with the cluster containing that document. Or, one can do a conventional query search to match the query against the representatives of all the bottom-level clusters. The cluster representative that is most similar to the given query then determines the starting cluster. Cluster searching then proceeds upward until a cluster is reached containing the number of documents the user wants to retrieve. [Willett, IP&M, 1988]

Of course, there is no guarantee that the cluster hypothesis is widely satisfied. It can only be verified empirically in any given collection. (See section on cluster validation.) In general, it is possible that a given algorithm will not generate any clusters, or that the clusters will overlap too much to be useful, or that the clusters which are formed will not correspond to meaningful topics of interest to prospective users. In an earlier section on relevance feedback (see above), another possible complication was pointed out: the documents relating to a given topic may form not one, but two or more separate clusters. However, it should be noted that all statistical IF techniques assume that it is possible to separate a collection of documents into at least two classes with respect to any given query, i.e., relevant and non-relevant documents.

Hierarchical clustering offers the potential for very fast filtering search time since most of the searching involves cluster representatives rather than all the documents in each cluster. However, if documents are fully indexed, vector space or boolean filtering without clustering may provide filtering time as fast or better since the only documents that need to be searched are those whose terms match the terms in the given query. If the query is well-formulated, these documents may be only a very small fraction of the collection.

The great virtue of most proposed clustering methods is that they are automatic. Of course, manual assignment of useful categories to each document in a collection is more certain to be useful, but it is very time-consuming and requires substantial manpower for large collections. Automatic clustering offers the hope of eliminating much of this effort. In the preceding section, an intermediate approach was described: Subject categories were assigned manually to a training set; thereafter, categories were assigned to new documents automatically.
Clustering requires some measure of the similarity between documents in document space. The widely used cosine similarity described earlier is an obvious choice. But other similarity measures are available. [van Rijsbergen, 1979] [Salton & McGill, 1983] [Korfhage, 1997] (See the discussion of document query similarity above.) Measures of dissimilarity can also be used, especially since the objective is to maximize the distance between clusters, e.g., between their centroids, in document space. A dissimilarity measure is, in essence, a distance measure, i.e., its value for any two documents $D_1$ and $D_2$ is greater the farther apart $D_1$ and $D_2$ are in document space. (Cosine similarity has the opposite property; it has its maximum value when two document vectors coincide, and has a value of zero when the document vectors are orthogonal. But 1-cosine is a distance measure, increasing with angular distance.)

Document clustering methods are generally distinguished from the descriptors used to characterize each document, and the similarity (or dissimilarity) function of those descriptors that is used by the clustering method. In general, the choice of inter-document similarity measure is independent of the choice of clustering method. And the choice of a similarity measure is (often) independent of the choice of the document descriptors that serve as independent variables of the similarity function. [Willett, IP&M, 1988] (But note that this is not true of the method of Zamir et al., described below.)

In general, research into clustering has focused on clustering methods, and algorithms for implementing these methods efficiently with respect to space and time. Willett concedes that the “similarity coefficient may affect the clustering that is obtained.” The method of Zamir et al. [SIGIR98], described below, provides at least preliminary evidence that a novel document descriptor, combined with novel inter-document and inter-cluster similarity functions, can produce dramatic improvement in cluster quality.

Two main strategies have been used for clustering. Both require a document-to-document similarity measure. The first strategy requires that the complete collection of documents to be clustered be available at the start of the clustering process. Hence, one may call this the “complete” strategy. (One might also call it the “static” strategy, since it assumes that the collection of documents is not going to change, i.e., documents are not going to be added or deleted, during the clustering process.) Most methods based on the complete/static strategy start by generating a list containing the similarity of every document pair in the collection. Hence, if the collection contains $N$ documents, the list
will contain \( N(N-1)/2 \) similarities. Methods using this “complete” strategy are expensive because of the large number of similarities involved, and the large number of comparisons that must be performed as documents are combined into clusters, and clusters are combined into larger clusters. A straightforward implementation requires that the interdocument similarity matrix, containing \( O(N^2) \) elements, must be searched \( O(N) \) times, once for each “fusion” of two clusters. Hence, the time requirement is \( O(N^3) \) and the space requirement is \( O(N^2) \). More sophisticated implementations have reduced the time requirement to \( O(N^2) \), but even this is prohibitive for document collections of realistic size such as the larger TREC collections. On the other hand, because these “complete” cluster methods take advantage of the full set of inter-document similarities, they meet van Rijsbergen’s three criteria for theoretical soundness: [van Rijsbergen, 1979] [Salton, 1989] (1) The clustering is unlikely to be altered drastically as new documents are added, (2) Small errors in the description of documents lead to small errors in clustering, and (3) the method is independent of the initial order of document reference, e.g., the order of document pair similarities. Essentially, these methods “attempt to seek an underlying structure in the data [rather than] impose a suitable structure on it.” [van Rijsbergen, 1979] Moreover, the expense is incurred mainly (apart from updates for new documents) when the collection is indexed, not at document filtering time. However for realistically large values of \( N \), the execution time and storage requirements even for pre-processing is prohibitively large. [Willett, IP&M, 1988]

The second major approach is the “incremental” strategy. Incremental methods [Zamir et al., SIGIR 98] [Salton, 1989] assume that the document collection to be clustered is arriving in a stream as the clustering proceeds. Hence, as each document arrives it is added to some cluster, or becomes the seed of a new cluster. When document \( i \) arrives, the \( i-1 \) documents that preceded it are already clustered. The \( i \)th document may be added to one of those existing clusters. It may become the seed of a new cluster. Or the existing clusters may be re-clustered in the process of adding the \( i \)th document. If reclustering is not allowed by the method, i.e., if the \( i \)th document must be added to one of the existing clusters, the method may be termed single-pass, not just incremental.
2.7.1 Hierarchical Cluster Generation ("Complete/Static" Methods)

A complete algorithm may start by considering all the documents as a single cluster and then breaking it down into smaller clusters ("divisive" clustering). Or, the algorithm can start with the individual documents and group them together into progressively larger clusters ("agglomerative" clustering). (Since agglomerative clustering produces a hierarchy of clusters grouped into larger clusters, it is often called Agglomerative Hierarchical Clustering, or AHC for short.) In the latter case, the similarities are sorted in descending order. Initially, each document is considered a separate cluster. The general rule is that at each stage the two most similar clusters are combined. Initially, the most similar documents are combined into a cluster. At that stage, “most similar” means having the highest similarity of any document pair. Thereafter, we need a criterion for deciding what “most similar” means when some of the clusters are still single documents and some are multi-document clusters that we have previously formed by agglomeration (or when all of the clusters have become multi-document). The various agglomerative cluster methods are distinguished by the rule for determining inter-cluster similarity when one or both clusters being compared are multi-document. [Willett, IP&M, 1988]

In “single-link” clustering (the most famous clustering method), the similarity between two clusters is defined to be “the similarity between the most similar pair of items, one of which appears in each cluster; thus each cluster member will be more similar to at least one member in that same cluster than to any member of another cluster.” The algorithm is called “single-link” because two clusters can be combined on the basis of one high similarity between a document in the one cluster and a document in the other. It is also called “nearest neighbor” clustering because two clusters are combined on the basis of the two documents, one from each cluster, that are nearest to each other. Hence, each cluster is formed by a chain of nearest neighbor document-to-document single links.

In “complete-link” clustering by contrast, the similarity between two clusters is defined to be “the similarity between the least similar pair of items” one of which appears in each cluster. “[T]hus each cluster member is more similar to the most dissimilar member of that cluster than to the most dissimilar member of any other cluster.” [Salton, 1989] Hence, in the “complete-link” algorithm, the similarity between two clusters (which determines whether they should be combined or not) depends on all the similarities between documents in the one cluster and documents in the other.
For any clustering method a similarity threshold can be applied, i.e., two clusters will be combined only if their inter-cluster similarity is greater than some threshold, $T$. Or the clustering process can be terminated when some halting criterion is reached, e.g., a pre-specified number of clusters. In the case of agglomerative clustering, the number of clusters is successively reduced until all clusters have been combined into a single cluster, the “root” of the hierarchy. If a halting criterion is specified, the agglomeration may stop when the criterion is satisfied, e.g., when successive levels of clustering have reduced the number of clusters to a specified value $NC_H$. Zamir et al. note that these AHC algorithms “are very sensitive to the halting criterion when the algorithm mistakenly merges multiple ‘good’ clusters, the resulting cluster could be meaningless to the user.”

Note that by either criterion, it is quite possible for a document in cluster $C_1$ to be more similar to some documents in cluster $C_2$ than to some other documents in $C_1$. Ideally, one would want to impose the criterion that every document in $C_1$ is closer to every other document in $C_1$ than to any document in any other cluster $C_2$. However, such strict criteria for “cohesion and isolation” of clusters appear to be too strict; in experiments, “very few sets could be found to satisfy” such criteria. [van Rijsbergen, 1979]

“Complete-link” clustering is considerably more computationally expensive (in either space or time but not both there is a trade off) than “single-link” clustering, but it has the advantage that one can generate clusters such that every pair of documents in a given cluster is above a specified similarity threshold. A document $D_i$ whose similarity to any other document $D_j$ is lower than the specified threshold will not be in any cluster. By contrast, in “single-link” clustering, the members of a cluster are chained together by “single links” such that the similarity between any two documents connected by a link, i.e., any two documents that were nearest neighbors at some stage of cluster combination, is guaranteed to be above the specified similarity threshold, but a pair of documents that are both in the same cluster but which were not directly chained together by the clustering process are not guaranteed to be above the threshold. Hence, single-link clustering allows document pairs with very low similarity to be in the same cluster. For that reason, complete-link clustering is probably better suited to IF applications. [Salton, 1989] In particular, complete-link clustering tends to produce small, tightly
bound, cohesive clusters, whereas single-link clustering tends to produce large, loosely-bound, “straggly” clusters. [Willets, IP&M, 1988]

A third agglomerative clustering method, “group-average” clustering, is intermediate between single-link and complete-link in that each member of a cluster has a greater average similarity to the remaining members of the cluster than it does to all members of any other cluster.

A fourth agglomerative clustering method, Ward’s method, “joins together those two clusters whose fusion results in the least increase in the sum of the [Euclidean] distances from each document [in the fused cluster] to the centroid” of the cluster. Evidently, this method is only defined when Euclidean distance is used for computing interdocument similarity. The centroid of a cluster is the average of the document vectors comprising the given cluster.

The defining characteristic of a cluster method is the rule that defines the clusters. For example, if the method is single link clustering, then the defining cluster rule is the one stated above: Two clusters are combined into a single cluster, if their closest members (according to the given inter-document similarity measure) are closer (more similar) than either is to the closest member of any other cluster.

In general, each of these cluster methods should be distinguished from the algorithms that have been developed to implement it since a given method often can be implemented by many different algorithms, each algorithm having its own distinct performance characteristics regarding space and time. [Willets, IP&M, 1988] For example, many algorithms are known that produce single-link clusters. They are all the “same” from the outside, i.e., given the same N documents, they will produce the same hierarchy of clusters. However, they may vary considerably in their space and time requirements. The SLINK algorithm [Sibson, 1973] has been shown to achieve optimal performance for single link clustering: $O(N^2)$ time complexity and $O(N)$ space. Trade-offs are possible.

The complete-link method has the same time complexity, $O(N^2)$, as single-link, if it also has access to $O(N^2)$ space; however, if it only has $O(N)$ space, it requires $O(N^3)$ time complexity. [Voorhees, PC]

### 2.7.2 Heuristic Cluster Methods

The term “heuristic” has been used by authors such as van Rijsbergen [1979] (he also uses the term “ad hoc”) to characterize methods that take shortcuts to achieve greater
efficiency in terms of space and time requirements. In particular, such terms refer to cluster methods that do not generate or do not access the full $O(N^2)$ set of interdocument similarities in a collection of $N$ documents. Such methods effectively make fewer (sometimes far fewer) effective “passes” through the interdocument similarity matrix or its equivalent. Heuristic methods tend to violate van Rijsbergen’s three criteria for theoretical soundness. In particular, it is characteristic of many such methods that the clusters for a given set of $N$ documents vary depending on the order in which documents are initially referenced. In the case of the “Buckshot” method discussed below, the method starts with a random sample of the $N$ documents; hence, the clusters produced will vary from one execution of the clustering method to another as the random sample varies.

In return for sacrificing theoretical soundness (and correspondingly, reducing one’s confidence that the true underlying structure of the collection has been captured), these methods tend to execute in $O(N)$ time. Actually, some of them execute in $O(kN)$ time (sometimes called rectangular time bounds, see below), where $k$ is equal to the number of clusters desired, or proportional to the number of clusters. If $k$ is a constant, especially a small constant, these are linear time methods. However, if the $k$ is proportional to $N$ (as for some methods and purposes it may be), then the method becomes $O(N^2)$ after all.

The distinction between heuristic and non-heuristic methods cuts across another distinction: that between incremental and non-incremental methods. Briefly, incremental methods operate entirely in “update” mode, generating and modifying clusters “on the fly” as each document is accessed. Below, we discuss some non-incremental heuristic methods. In the next section, we discuss incremental methods in some detail. Purely incremental methods tend also to be heuristic, e.g., single pass methods and the like. However, we discuss one novel incremental method, STC, that succeeds in being linear time and non-heuristic.

In one respect, the very term “heuristic” is misleading with respect to clustering methods. In conventional usage, the concept of a “heuristic” algorithm or method of solving a problem implies that there is a perfect solution to the given problem. Heuristic methods are, by definition, not guaranteed to find that perfect solution. One employs a heuristic method only if methods that are guaranteed to find the perfect answer are either (a)
unknown, or (b) computationally impractical, e.g., far too costly with respect to space or time, or not guaranteed to terminate at all. However, in the realm of document clustering, a “perfect” result is not defined. In general, the “best” set of clusters depends on the objective for which clustering is being performed, e.g., classification, information filtering, browsing, thesaurus generation, etc. Moreover, the best cluster method even for a given objective may depend on the statistical characteristics of the collection. Hence, even the so called “complete” or $O(N^2)$ methods are not guaranteed to produce the “best” result.

A heuristic algorithm, associated with Rocchio, [Rocchio, 1996] was developed on the SMART project. It begins with applying a density test to each document that has not yet been clustered, thereby identifying “cluster seeds,” documents “that lie in dense regions of the document space, that is items surrounded by many other items in close proximity.” [Salton, 1989] For example, density may be defined by requiring that $n_1$ documents have a similarity (Rocchio typically used cosine similarity) of at least $p_1$, and $n_2$ documents have a correlation of $p_2$. All documents sufficiently similar to a seed, i.e., having a similarity to the seed that exceeds a pre-specified threshold, form a cluster. Clusters may overlap, i.e., a document may be assigned to more than one cluster. [van Rijssbergen, 1979] In a second, iterative stage, the clusters formed in the first stage are adjusted to conform to certain pre-specified parameters, e.g., minimum and maximum documents per cluster, number of clusters desired, degree of overlap permitted, etc. Documents too far removed from cluster seeds, or occupying regions insufficiently dense, remain unclustered, or undergo a separate clustering process in a third stage.

Much more recently, Cutting et al. [SIGIR 1992] have developed two linear-time heuristic clustering methods, called Buckshot and Fractionation, respectively. More precisely, they are rectangular bound, i.e., $O(kN)$, methods. These clustering methods have been developed for use in an interactive browsing technique called “Scatter/Gather.” (See section on User Interaction below.) For this application, $k$ is small and constant, e.g., eight in the examples given by the authors. Hence, $O(kN)$ is practical and effectively linear. Moreover, Cutting et al. [SIGIR 1993] have developed an enhancement to Scatter/Gather that works even for very large corpora, e.g., gigabytes. Hence, Buckshot and Fractionation continue to be useful even for TREC-sized corpora.
The essential “trick” of Buckshot and Fractionation is to use one of the “complete” $O(N^2)$ clustering methods, but to apply it very incompletely to the original large corpus. The result is to generate rough clusters very rapidly. The centroids of these clusters then become “seeds” around which the entire corpus can be clustered on the basis of simple document-centroid similarity.

For example, in Buckshot, clustering is applied initially to a random sample of $KN$ documents. Hence, an $O(N^2)$ method applied to this sample clearly runs in $O(kN)$ time. (Cutting et al. use Group Average Agglomerative Hierarchical Clustering (AHC) as their $O(N^2)$ method.) This AHC method is used to obtain $k$ clusters from the random sample. The centers of these clusters are then used as seeds around which the entire corpus is clustered. Hence, in Buckshot, a complete, hierarchical clustering method is used, but it is applied to a small sample of the corpus.

In Fractionation, the entire corpus is initially partitioned into $N/m$ buckets, each of fixed size $m$ documents, where $m > k$. Each of the buckets is then clustered using an AHC method. The objective of the clustering is to reduce each bucket from $m$ individual documents to $mr$ clusters, where the $r$ is a pre-specified reduction factor ($r < 1$). Hence, after the first clustering stage, there are $mr$ clusters in each of the $m$ original buckets. Each of these $mr$ clusters is then treated as an individual “virtual document” of size $1/r$ original documents. Since there are $N/m$ buckets, there are now $mr(N/m) = Nr$ virtual documents. These virtual documents are now processed exactly like the original documents, i.e., partitioned into $N_{r/m}$ buckets, each of which is then clustered into $mr$ virtual documents at the second stage. Hence, after stage two, there are $(N_{r/m})^2mr = Nr^2$ virtual documents. This process continues until after $j$ stages, there are $Nr^j < k$ virtual documents or clusters. One final agglomerative stage produces $k$ clusters, whose centers then become the seeds for clustering the entire corpus on an individual document basis. Hence, in Fractionation (in contrast to Buckshot), the complete hierarchical clustering method is applied to the entire corpus, but the clustering performed is coarse, because the corpus is partitioned arbitrarily into buckets, and the clustering is applied separately to each bucket rather than to the corpus as a whole.

Given the $k$ seeds or cluster centers (produced by either Buckshot or Fractionation), how are the $N$ documents clustered? As a simple approach (similar to Rocchio), each document can be assigned to the center to which it is most similar. This process can be
refined by iteration, i.e., after each document in the corpus have been clustered around one of the seeds, the centers can be recomputed, and once again, each document can be assigned to the most similar center. Several other refinements are employed by Cutting \textit{et al.} First, they summarize each cluster as a \textit{profile}. The profile of a cluster is a term vector equal to the sum of the term vectors representing the documents in the cluster. Hence, in successive iterations, instead of adding a document to the cluster with the most similar center, one can add the document to the cluster with the most similar profile vector. They also define \textit{split} and \textit{join} algorithms. The former ‘separates poorly defined clusters into two well separated parts,’’ and the latter ‘merges clusters which are too similar.’ In general, the choice whether to use these refinements, and how many refinements and iterations to apply, involves a trade-off between speed and accuracy. However, the entire method including refinements, runs in $O(kN)$ time.

Buckshot and Fractionation partition the corpus, i.e., they do not support overlap (although they can be modified to support overlap). [Zamir \textit{et al.}, SIGIR ‘98] Neither algorithm is incremental. Moreover, Buckshot generates its initial cluster samples from a random sample of the initial corpus. Hence, it is not deterministic, i.e., in successive runs, it may generate different clusters “although [in the author’s experience] repeated trials generally produce qualitatively similar partitions.” [Cutting \textit{et al.}, 1992] A further risk is that “when one is possibly interested in small clusters,... they may not be represented in the [random] sample.” [Zamir \textit{et al.}, SIGIR 1998]

Buckshot and Fractionation (and the STC clustering method described in the next section) are all motivated by the desire to allow the user to browse a large document collection interactively. Such an application requires that the clustering be performed very rapidly, e.g., in seconds, even for very large collections. If the collection is available in advance, e.g., hours before the user begins browsing, then hybrid approaches to speed up the interactive clustering are possible. The essential idea is that a fixed cluster hierarchy is generated off-line. The nodes of this hierarchy then become “virtual documents” to be clustered on-line (as in Fractionation above).

Clearly, the “complete” $O(N^2)$ methods described in the preceding section can be used to generate this cluster hierarchy. However, for large collections, e.g., thousands of documents or more, $O(N^2)$ methods are far too slow even for off-line clustering. Hence, Cutting \textit{et al.} [SIGIR ‘93] propose using an $O(kN)$ clustering method such as their own
Buckshot or Fractionation, even in the off-line stage. These algorithms only produce one level of cluster, i.e., they produce a “flat” partition of the collection into $k$ clusters. However, the flat clustering method can be applied iteratively to partition each of the original $k$ clusters into $k$ sub-clusters, then to partition each of these sub-clusters yet again, and so on. The partitioning stops only when individual documents are reached. This iterative procedure generates the required cluster hierarchy. Since the clustering at each level runs $O(kN)$ time, the entire hierarchy can be generated in $O(kN \log N)$ time. (In their reported experiment, Cutting et al. generated a cluster hierarchy for a DARPA Tipster collection of 700,000 documents, occupying 2.1 gigabytes of text, and containing over a million unique words, in forty hours on a Sun SPARCStation 10. While faster hardware would obviously do the job more quickly, it is evident that the production of such cluster hierarchies remains an off-line task.)

Given this pre-computed cluster hierarchy, Cutting performs constant-time clustering interactively, based on the assumption that a fixed number of virtual documents $M >> k$ is to be clustered. $M$ is chosen such that the clustering can be performed in the desired constant time bound, regardless of the true number of actual documents in the collection. Cutting obtains the $M$ virtual documents (also called meta-documents) by starting at the root of the cluster hierarchy, or at any node in the hierarchy designated by the user. He replaces the initial node by its children. Each child node is itself a virtual document. At each subsequent stage, he finds the child that has the most leaves and replace that child by its children. He continues this process until $M$ children, i.e., $M$ virtual documents, have been accumulated. Then, he clusters these $M$ virtual documents into $k$ clusters. In his published experiment, he found that interactive clustering took approximately 20 seconds.

Note that most clustering methods (including those used by Cutting) involve computing inter-document or document-centroid similarities. These similarities typically involve a similarity function such as cosine similarity applied to documents represented as term vectors, where each term in a vector is a word or phrase. When the documents being clustered are virtual documents, the term vectors will be very long; the term vector for virtual document $V_i$ will contain a non-zero value for every term in any of its descendent leaves, i.e., for any of the actual documents of which it is composed. In other words, the set of terms in a virtual document is the union of the terms in its component actual
documents. To keep the similarity computations from being very slow, Cutting truncates the term vectors of every virtual document, retaining only the fifty highest weighted terms. (This also reduces substantially the space required to store the cluster hierarchy.) Schutze et al. [SIGIR ’97] find that with such truncation, “the speed increase is significant while surprisingly the quality of clustering is not adversely affected.”

The Cutting constant-time clustering method clusters virtual documents, where each virtual document is essentially a node from some level in the pre-computed cluster hierarchy. The clusters produced are always unions of these virtual documents. Of course, as the browsing user focuses more narrowly, on lower levels of the cluster hierarchy, the virtual documents that get clustered may be correspondingly narrow, i.e., may contain small numbers of actual documents. But to some degree, the user is limited by the original pre-computed hierarchical structure. Silverstein et al. [SIGIR ’97] address this issue with their “almost-constant-time” cluster method.

Like Cutting, Silverstein assumes a pre-computed cluster hierarchy, $H$, covering a document corpus $C$ of size $N$. However, whereas Cutting only allows the user to designate a node (or perhaps several nodes at some level) in $H$ as the starting point for clustering, Silverstein assumes that the user has obtained (and wishes to cluster for browsing) some subset $S$ of actual documents from $C$. For example, the subset $S$ might be obtained by executing a query $Q$ against $C$ using some IF engine. Silverstein wishes to map $S$ into $H$ in such a way that the clusters reflect $S$ but use the pre-computed $H$ to speed up the clustering. He acquires $M$ virtual document nodes from $H$ for clustering as Cutting does. However, he departs from Cutting in two significant ways. First, in expanding the set of virtual documents to reach his goal of $M$, he uses a “goodness” test to weed out the nodes that contain the smallest proportion of documents from $S$. Specifically, when a node is replaced by its children, this goodness test is used to identify the worst child, the one with the smallest proportion of documents from $S$. This “bad seed” is replaced by some of its children. The replacement process is also a weeding out process: Specifically, child nodes that contain no documents from $S$ are discarded. Each child node that contains only a single document from $S$ (actually less than $c$ documents from $S$, where $c$ is a small constant) is replaced by a “singleton” node that contains only the document(s) from $S$. After $M$ nodes have been accumulated, Silverstein clusters these $M$ nodes (actually the union of the $M$ nodes and the “singleton”
nodes) into $k$ clusters, as Cutting does. Finally, he goes through the all the clustered nodes, removing actual documents that are not in $S$.

The Cutting method is constant time (at user interaction time, not cluster hierarchy precomputation time, of course) because it clusters a fixed number $M$ of nodes, where $M$ is chosen so that the clustering time is acceptable to an interactive user. The Silverstein method is almost-constant-time because it requires the generation of a function (implemented as a table) that identifies the documents in $S$ that are contained in any given node $n$ of $H$. The computation of this table takes $O(|S| \log N)$ time. Hence, computation of this table is not constant, and cannot be precomputed because it depends on $S$, which is specified by the interactive user. However, the time to compute the table is “dwarfed, in practice by the contribution of the [constant] clustering step.”

It should be stressed that both the Cutting and Silverstein methods depend on the availability in advance, i.e., off-line, of the collection to be clustered (or in the case of Silverstein, the collection from which a subset is to be clustered). Moreover, this collection must be available well in advance of user interaction, because the required precomputation of a cluster hierarchy is a time-consuming process. By contrast, the STC method, described in the next section, achieves comparable or better run-time speed without any precomputation. It is genuinely incremental.

Heuristic algorithms fail van Rijsbergen’s three criteria for theoretical soundness, but have been found “about as effective as those based on hierarchical agglomerative methodologies [e.g., single-link and complete-link]” in “many filtering settings.” [Salton, 1989]

### 2.7.3 Incremental Cluster Generation

Incremental methods also make use of a similarity measure but they don’t require that similarities be pre-computed for all document pairs. Indeed, all document pairs are not available initially, since by definition, incremental methods cluster a stream of incoming documents. The similarities are computed “on the fly” as the documents stream past the incremental cluster system (or what comes to the same thing, as the incremental system makes a “pass” through the document collection). All incremental cluster methods may be said to make a single pass through the documents, in the sense that as the $i$th document is accessed and processed, the result is the best set of clusters the method can
produce with $i$ documents. By the time the $N$th document, i.e., the “final” document, is processed, the entire collection has been clustered and the method is “done.” Note that it doesn’t make any difference to an incremental method whether the collection of $N$ documents is available initially, or whether it is a stream of $N$ documents arriving in sequence, e.g., $N$ documents that have been retrieved from the Web by an IF engine. In either case, an incremental method processes the $i$th document as if it only has knowledge about the first $i$ documents, and hasn’t yet encountered documents $i+1$ to $N$.

One can distinguish between methods that are purely incremental, and methods that can run in either an incremental or non-incremental mode. For example, the single-link method discussed in the preceding section is classically executed in a static mode, i.e., it starts with $N$ documents. Algorithms exist for clustering the $N$ documents according to the single-link method. Algorithms also exist for adding an additional $(N+1)$th document, without starting over from scratch. Such update procedures produce the same effect as if one had started from scratch. One can see that such algorithms must exist for the single-link method by considering the effect they must produce in terms of the single-link clustering rule. The $N$ documents have been grouped into a hierarchy of clusters according to the single-link rule. Document $D_{N+1}$ must be linked to the document $D_i$ ($i \leq N$) with which it is most similar. The effect is to link it into every cluster in the hierarchy that contains $D_i$. If $D_{N+1}$ is equally similar to several “closest” documents $D_p, D_j, D_k$ (which need not be very similar to each other, and hence may be in different clusters up to a very high level in the hierarchy), then the clusters to which they belong will be linked together into a single cluster (if they were not already so linked). It is plain that the cluster hierarchy formed by adding document $D_{N+1}$ continues to obey the single-link clustering rule.

However, if an update procedure exists, then why not use it starting with document $D_2$ (the 2nd document encountered), and throw away any non-incremental methods that work on the first $N$ documents, where $N \gg 2$? If the non-incremental procedure is retained, then it must be because it is more efficient (in documents clustered per unit of time) than the update procedure, or because it produces a data structure better-suited to efficient document or cluster filtering.
On the other hand, if the update procedure is the only procedure used in the clustering, then the implementation is purely incremental. If the update procedure is the cluster defining rule, then the cluster method is incremental.

All incremental methods make a single pass through the documents. However, they are not all single-pass methods. The essential distinction is whether the method, on encountering the \( i \)th document, revisits earlier documents \( j < i \), or existing clusters, with a view to possible re-clustering. If it does, it is not a “pure” single pass method, since it may visit documents more than once, in the process of making its “single” pass. Actually, virtually all incremental methods do some revisiting, but a method that does no re-clustering, and minimizes document revisiting is “more” single-pass than a method that revisits and re-clusters extensively. In general, single-pass methods are order dependent, i.e., the clusters formed depend on the order in which documents are processed. The reason is evident; there is no opportunity to revise clusters formed from early documents on the basis of information in documents that are processed later. This violates one of van Rijsbergen’s three criteria for theoretical soundness.

A nearly single-pass method works as follows: If a new document is similar enough (according to some similarity measure and threshold) to one of the preceding documents, they are combined into a cluster. Similarly, if a new document is similar enough to a cluster formed from two or more of the preceding documents, it is added to that cluster. A document is added to all clusters to which it is similar enough. (To simplify the process of computing the similarity of a document and a cluster, a centroid is computed for each cluster. Each new document is then compared to the centroid of each existing cluster. If a document is added to a cluster, its centroid is recomputed. Of course, recomputation of the centroid of a cluster involves revisiting the documents that are members of the given cluster, so even this simple algorithm involves some revisiting.)

When the single pass is completed, a number of clusters have been formed. The results may depend on the order in which the documents were examined. This process may result in very large clusters, or clusters with a large amount of overlap. Hence, one usually specifies such parameters as minimum and maximum cluster size, and maximum overlap, etc. The clusters formed in the original pass through the data are then adjusted, e.g., by cluster splitting and merging, to conform to these parameters.
Zamir et al. [SIGIR 98] have developed a novel incremental clustering method, called Suffix Tree Clustering (STC). The STC method is motivated by a problem that arises frequently when querying the Web with an IF engine: A huge ranked list of documents is retrieved, of which only a very small number are relevant to the user’s query. To make matters worse, the relevant documents are often far down the list of documents returned. In IF terminology, precision is often very low. Zamir et al. propose to alleviate this problem by clustering the documents returned to the user, and labeling each cluster with phrases that characterize (one hopes) its common topic(s). The user then browses through the clusters, picking out and ranking the clusters that appear most relevant to her query on the basis of their labels. She can then begin examining documents in the most relevant cluster. If she needs or desires more information than she finds in the first cluster, she can go on to the next most relevant cluster (as determined by its labels and the user’s judgment), and so on. In this way, the number of documents that she must sift through to find relevant data can be reduced by at least an order of magnitude. (See the section on User Interaction for further discussion of browsing.)

Note that STC, although motivated by the difficulties of Web filtering, is applicable in any case where the number of documents $N$ retrieved, is large, and the precision is low. It is particularly applicable if the user is doing her filtering interactively, and wants to see her output very quickly.

STC has a number of very noteworthy features. Some of these features are found in other clustering methods, but no other method (to the author’s knowledge) combines them all. (1) STC is a linear time method, i.e., the time required to cluster $N$ documents rises only linearly with $N$. This is an essential requirement whenever $N$ is very large, as it is in many practical applications. (There are constant-time and almost-constant-time clustering methods (see previous section), but they depend on off-line pre-clustering.) (2) STC is incremental, which means that it can begin clustering documents as soon as the first document arrives; it doesn’t have to wait until all the retrieved documents have arrived before it begins clustering, as non-incremental methods do. (By contrast, the complete clustering methods discussed in an earlier section are non-incremental and non-linear time. The heuristic clustering methods discussed in the preceding section are either linear time, but non-incremental, or near constant time, but require substantial pre-processing, and hence are also non-incremental.) (3) The STC method is non-
heuristic in the sense that the clusters it produces are independent of the order in which the documents it clusters are initially accessed. (4) The STC method does not require pre-specification of either the number of clusters to be generated, or the maximum or minimum size of the resulting clusters. Heuristic methods commonly have such halting or cleanup criteria; even AHC algorithms are often used in conjunction with such criteria. STC only requires pre-specification of two parameters, a cluster overlap measure and the number of “best clusters” to be reclustered (see below). But Zamir et al. find that STC is not very sensitive to the value of the overlap parameter. (5) The STC method permits a document to be placed in more than one cluster (a characteristic that is termed cluster overlap). This is a very important consideration when the objective is to cluster documents by topic, since a given document may be about multiple topics.

(6) A feature that sets STC apart from practically all other clustering methods is that it uses ordered strings of words, which it calls “phrases,” as its document descriptors. (Most other methods of text document clustering use unordered sets of words.) Moreover, STC uses the presence of a phrase in two documents as its inter-document similarity measure for clustering purposes. The presence of a shared phrase is also the STC primary (first stage) cluster method rule. (In sharp contrast, all of the other clustering methods described above are totally indifferent to how similarity of text documents is defined, or indeed even to the fact that the objects being clustered are text documents.) In other words, if $D_1$ and $D_2$ share at least one common phrase (as defined below), they are combined into a “base” cluster. Hence, instead of an $N^2$ matrix of inter-document similarities, STC employs a structure called a “suffix tree,” that indexes a document collection by the phrases of which its documents are composed. The index itself grows linearly with the size of the text, and can be updated and accessed in linear time. Experiments by Zamir et al. indicate that the use of ordered strings as document descriptors is critical to the success of the STC method.

STC employs a second stage of clustering, using a different clustering rule. The second stage rule combines the clusters produced by the first stage according to the proportion of documents that they share (see below). In other words, cluster overlap is permitted, but if the percentage of overlap is very high (Zamir et al. use a parameter of 50%), the clusters are combined into a larger cluster. Note that STC is not hierarchical; only two stages of clustering are performed. Hence, no root cluster is generated, and no halting
criterion is required. STC is also not iterative (like Rocchio’s method). Rather, it is incremental. The two clustering stages are performed every time a new document arrives or is accessed.

STC indexes the collection of documents as a “suffix tree,” hence the name of the method. In most past applications, a body of text has been viewed as an ordered string of characters. The suffix tree has been employed as an efficient representation of the string for such purposes as finding (in linear time): a given word $w$ in the text, the first occurrence of $w$ in the text, the number of occurrences of $w$ in the text, etc. [Crochemore et al., 1994] Each path in the tree from root to a leaf node represents a “suffix” of the text. Each internal node of the tree represents a “prefix” shared by two or more suffixes. In STC, the text is viewed as an ordered string of words. The application is to cluster documents that share one or more word strings. (Zamir et al. call these strings “phrases,” but no syntactic structure is implied.) A suffix tree is constructed to represent and index all the sentences in a collection of documents. Each sentence $SE_i$ is treated as an ordered string of $n$ words, $w_{i1}, w_{i2}, \ldots, w_{in}$. Every sub-string $w_{ij}, \ldots, w_{in}$ for $1 <= j <= n$, is a suffix of $SE_i$. In other words, there are $n$ suffixes for every sentence of length $n$ words, the string beginning at word 1, the string beginning at word 2, etc. (Technically, there are $n+1$ suffixes, because the set includes the “empty” suffix, located at the root of the tree.) A suffix tree is constructed for the first sentence of document 1, updated with the suffixes of the second sentence, and so on. As each new document arrives and is added to the growing collection, the suffix tree is updated to reflect all the new suffixes in its sentences that were not encountered in any previous document, and all the re-occurrences of suffixes that were previously encountered in one or more earlier documents. As each new sentence $SE_i$ of the current document $D_j$ is processed, the suffix tree is updated to reflect all the new suffixes in $SE_i$ that were not encountered in any previous document or any previous sentence of $D_j$, and all the re-occurrences of suffixes that were previously encountered in one or more earlier documents. Each distinct suffix becomes the path, the series of edges, leading to a leaf node of the suffix tree. Each leaf node is labeled with the suffix whose path leads to that node, and is indexed by all the documents in which the given suffix appears. Similarly, the path to each internal (non-leaf) node represents a prefix, a string of words that begins two or more suffixes. The internal node is labeled with its prefix, and indexed to identify all the documents in which its suffixes occur.
For example, document $D_1$ may contain the suffix “The quick brown fox jumped over the lazy dog.” Call this suffix $S_1$. (Actually, there will be $n$ suffixes for this sentence, corresponding to: “dog,” “lazy dog,” “the lazy dog,” etc., up to and including the suffix representing the complete sentence.) Each non-leaf (internal) node represents the first $n_1$ words, the prefix, of two or more suffixes. There will be $n-1$ implicit prefixes for this sentence: “The,” “The quick,” “The quick brown,” “The quick brown fox,” etc. However, these prefixes remain implicit, until one of them is encountered in a subsequent sentence of $D_1$ or a subsequent document $D_2$, with a different continuation. Each internal node must have at least two children, each child representing a different continuation of the parent. Document $D_2$ may contain a sentence with the suffix, $S_2$, “The quick brown fox leaped over the sleeping collie.” When $D_2$ is encountered, the implicit prefix node “The quick brown fox” becomes an explicit node $P_a$. A third document $D_3$ may arrive containing a sentence with the suffix $S_3$, “The quick are livelier than the dead.” In that case, an internal explicit node is created representing the prefix, $P_b$, “The quick.” This node will have two children, one the leaf node for $S_3$, the other the internal node for prefix $P_a$ (a continuation of $P_b$), branching in turn to the two children representing the suffixes $S_1$ and $S_2$. In this way, all of the word sequences representing (and characterizing) a collection of documents can be efficiently represented, and easily updated as new documents arrive, containing new suffixes, and new instances of existing suffixes. Each internal node of the suffix tree represents a prefix, and the set of one or more documents containing that prefix, e.g., in the above example there is an internal node representing the prefix, $P_b$, and the documents, $D_1$, $D_2$, and $D_3$, containing it. Each leaf node represents a suffix, and the set of documents containing it, e.g., there is a leaf node representing $S_3$, and the document $D_3$ containing it. The set of documents associated with a node of the suffix tree is called a “base cluster,” and the phrase that is common to those documents is its label. $D_1$, $D_2$, and $D_3$ comprise a base cluster whose label is $P_b$. Note that a phrase can be either a prefix or a suffix, or even both, i.e., it can be a suffix of a sentence in one document, and a prefix of a sentence in another document.

Note that, for purposes of clustering, we are only interested in phrases (prefixes or suffixes) that occur in two or more documents. However, we create a node for every suffix of a sentence in a given document $D_1$, and for every prefix that occurs in two or
more sentences of $D_1$. That is because we don’t know in advance whether a given prefix or suffix in $D_1$ will be encountered again in a later document. However, we only form base clusters for phrases that occur in at least two documents.

Any sequence of words in a sentence of the collection is a phrase in STC. The sequence need not be a phrase in any syntactic sense, though syntactic phrases will be included (provided they are composed of contiguous words). Also observe that a given document may (and normally will) contain multiple phrases, e.g., $D_1$ contains $P_a$, $P_b$, and $S_1$. Some of these phrases will be unique to the given document, others will be shared with other documents. It should also be noted that two documents may share more than one phrase, e.g., $D_1$ and $D_2$ share both $P_a$ and $P_b$. Hence, the base cluster with label $P_a$ and the base cluster with label $P_b$ overlap, i.e., share $D_1$ and $D_2$. On the other hand, the two base clusters do not coincide; the base cluster for $P_b$ also contains $D_3$, a document that is not in the base cluster for $P_a$.

The construction of the suffix tree not only indexes the document collection, linking each phrase (including suffixes) to the documents that contain it; it also performs the first stage of clustering, the identification of the base clusters. The second stage of clustering combines base clusters that are “similar.” Base cluster similarity is defined as having a high degree of document overlap. In other words, base cluster $C_1$ (defined by containing the phrase $P_1$) and base cluster $C_2$ (defined by containing the phrase $P_2$) are said to be similar if a high proportion of the documents in these two clusters contain both $P_1$ and $P_2$. Specifically,

$$\frac{|C_1 \cap C_2|}{|C_1|} > 0.5$$
$$\frac{|C_1 \cap C_2|}{|C_2|} > 0.5$$

where 0.5 is the overlap threshold chosen by Zamir et al.

The cluster $C_{12}$ formed by combining $C_1$ and $C_2$ contains the union of the documents in $C_1$ and $C_2$ (and is labeled with both $P_1$ and $P_2$). Similarly, $C_2$ may be combined with $C_3$ into $C_{23}$ on the basis that a high proportion of the documents in these clusters contain both $P_2$ and $P_3$. $C_{12}$ and $C_{23}$ are then combined into $C_{123}$ on the basis of the common linking cluster $C_2$. The result is a form of single-link clustering with $C_1$ linked to $C_2$. 

169
which is linked to $C_3$. However, the undesirable chaining effect in which the end points of the chain are quite dissimilar is much less pronounced in the STC case because the chaining is taking place at the level of base clusters rather than individual documents. The composite clusters that emerge from stage two are labeled with the set of phrases that label their component base clusters.

Each new document that arrives causes the suffix tree to be updated. If document $D_i$ contains a suffix $S_j$ that did not occur in any of the preceding documents $D_1$ to $D_{i-1}$, then a new leaf node is added to the suffix tree with value $S_j$ (and maybe a new internal node as well, if the new suffix is a new continuation of an existing prefix $P_{old}$ that was previously only the beginning of one suffix or suffixes in only one document, $D_h$, $h < i$, but is now the prefix of suffixes in two documents and hence rates its own internal node.) Corresponding to the new leaf node, a new base cluster is created, containing initially just the one document $D_i$ and labeled with the new suffix, $S_j$. (If a new internal node is created too, a new base cluster will be defined for that node, containing $D_i$ and $D_h$, and labeled with the shared prefix, $P_{old}$.) If $D_i$ contains a phrase $P_k$ (prefix or suffix) that is already in the suffix tree, i.e., that occurs as the value of a node in one or more of the first $i-1$ documents, then $D_i$ must be added to the base cluster with label $P_k$.

Note that this may have the effect of either increasing or decreasing the similarity of base cluster $C_k$ to other base clusters. If $D_i$ also contains the phrase $P_j$, it increases the overlap of $C_k$ with $C_j$ (or creates such an overlap if it did not exist previously). If $D_i$ does not contain phrase $P_j$, it decreases the overlap (if any). Hence, $D_i$ may push the overlap between $C_j$ and $C_k$ over the user-defined threshold, causing clusters $C_j$ and $C_k$ to be combined if they were not combined previously. Similarly, $D_i$ may push the overlap between $C_j$ and $C_k$ below the threshold, causing combined cluster $C_{jk}$ to be split apart into its component base clusters $C_j$ and $C_k$. Hence, each new document may cause re-clustering of existing clusters. (STC is an incremental clustering method, but definitely not a pure one-pass method, as that term has been defined here.)

The potential re-clustering when a new document $D_i$ arrives requires that inter-cluster similarity be recomputed for all affected base clusters, and for all clusters composed of affected base clusters. The number of such clusters goes up as each new document
arrives. Hence, the number of required similarity computations can rise dramatically as the population of documents grows. Since the objective is to support realtime clustering (remember that the original motivation was to support browsing of documents retrieved by an interactive user from the Web!), the number of similarity computations is held constant by only comparing updated base clusters with the $q$ “best” base clusters, where $q$ is a parameter; $q=500$ in the research reported here. The term “best” does not refer to relevance to the user’s query; that is determined interactively by the user after all the retrieved documents have been clustered. Instead, it refers to a measure of a cluster’s merit as a cluster.

Base clusters are ranked by score $s(B)$, computed as:

$$s(B) = |B| \cdot f(|P|)$$

where $|B|$ is the number of documents in base cluster $B$, $|P|$ is the effective length of phrase $P$, and $f(P)$ is a non-linear function of phrase length. The function $f(P)$ “penalizes single word phrases, is linear for phrases that are two to six words long, and becomes constant for longer phrases.” (Note that, strictly speaking, the “break points;” two and six, of $f(P)$ are also parameters of the STC method. Clearly, other values could be chosen, e.g., the phrase length score could be allowed to increase linearly up to a length of eight. Zamir et al. do not discuss how they arrived at these values, or any study of the effect of altering these values.) The length of a phrase is the total number of words, excluding stop words; for Web applications, the usual stoplist is expanded to include common Internet words like “java” and “mail.” Hence, a phrase consisting of a single stoplist word has a length of zero. The value of $f(0)$ is zero, so base clusters defined by sharing a single stopword are discarded. Note that although stop words do not contribute to the length of phrases in which they occur, and hence do not contribute to the ranking of the corresponding base cluster scores, they can serve to distinguish phrases. In the above example, if document $D_4$ arrives ending with the suffix, “The slow brown fox can’t jump over anything,” the phrase “The quick” becomes two distinct phrases, “The quick” and “quick,” because the phrase, “The,” now becomes a separate node with two distinct children, “quick” and one beginning with “slow.” The base cluster corresponding to “The” is discarded because its score is zero.

After the $N$th (final) retrieved document has been processed, all $N$ documents have been clustered; the STC clustering process is completed. The user can then browse these
clusters, judging on the basis of their labels which are most likely to contain relevant documents. (Zamir hypothesizes that the phrases that label a cluster will prove effective descriptors of that cluster’s content for effective human browsing, but this belief had not yet been tested in the reported research.) When she finds the most promising cluster, the user can “drill down” and look at titles or whatever other “snippets” the Web engine has returned. When she finds an interesting “snippet,” she can drill further to look at the full text of the corresponding page. Zamir assumes that even at the cluster level, the number of entities generated by STC will be greater than the user can comfortably browse. So, he ranks the final set of clusters, assigning each cluster a score “based on the scores of its base clusters, and their overlap.” Hence, the user only has to (is allowed to?) browse the \(p\) best clusters. Again, “best” is a measure of cluster quality, e.g., number, size and overlap of its component base clusters (and hence of its coherence), length of the phrases that label it (longer phrases are likely to be more descriptive), etc. Cluster relevance is determined interactively by the browsing human user. The human browser sees the number of documents in each cluster, and the phrases of its base clusters.

The STC method appears to achieve the quality of a “complete;” i.e., \(O(N^2)\) method (see discussion of Cluster Validation below), while running in linear time, i.e., \(O(N)\). The “secret” is the nature of the “similarity” measure STC uses, and the efficient data structure and algorithm STC uses to index the documents and compute the similarity. Practically all other cluster methods use a measure such that if document \(D_1\) is similar to document \(D_2\), and document \(D_2\) is similar to \(D_3\), one cannot assume that \(D_1\) is similar to \(D_3\). In a word, these measures, e.g., cosine similarity, are non-transitive. As a result, every pair of interdocument similarities needs to be computed and accessed for “completeness.” By contrast, STC forms its base clusters on the basis of shared phrases. If \(D_1\) and \(D_2\) share a phrase, and \(D_2\) and \(D_3\) share the same phrase, then \(D_1\) and \(D_3\) certainly share that phrase too! Hence, STC can perform complete clustering at the base cluster level without incurring the \(O(N^2)\) penalty. STC achieves \(O(N)\) time and space by employing a suffix tree to index the document collection, and an efficient algorithm due to Ukkonen [Algorith, 1995] [Nelson, 1996] to build and update the suffix tree. The second-stage clustering of base clusters is not transitive, but involves clustering of base clusters, not documents. Moreover (and this is the most “heuristic” element of the method), during the incremental re-clustering of base clusters, only the \(q\) “best”
existing clusters are revisited, as noted above. This keeps the time (actually the maximum time) required for stage two constant as the number of documents grows.

Finally, it should be noted that because it is incremental, the performance of STC in the application domain for which it was developed, e.g., clustering of documents retrieved from the Web, may be much better than linear. STC can cluster the documents as they are arriving. Hence, by the time the last (Nth) document arrives, STC’s clustering may be nearly completed. In a reported experiment, cluster results are returned “to the user a mere 0.01 seconds after the last document is received by” the Web filtering engine. However, it should be noted that clustering in this case was performed on the “snippets” extracted from Web pages by the IF engine, not the full text of the actual pages. This is compared by Zamir to the truncation of document vectors by Cutting et al. [SIGIR ’93] and Schutze et al. [SIGIR ’97] described earlier.

2.7.4 Cluster Validation

Since even randomly generated data can be clustered, it is important to determine whether the clusters produced when a given clustering method is applied to a given collection, are meaningful. It is even more important to determine whether the clusters produced contribute to effective information filtering. In other words, are the clusters produced likely to satisfy the cluster hypothesis?. If a query or browsing method locates and retrieves a cluster of appropriate size, is it likely that many or most of the documents in that cluster will be relevant to the query, or of interest to the browsing user? If the user relaxes the cluster threshold, retrieving documents that were close to the boundary of the original cluster, are these new documents likely to be at least partially relevant to the user’s need?

Several approaches to cluster evaluation with specific applicability to document filtering have been tried. These approaches try to determine whether a given collection is a good candidate for clustering, i.e., whether clustering will promote filtering effectiveness. One approach, due to van Rijsbergen and his associates [van Rijsbergen et al., 1973] is to compare the average interdocument similarity among relevant documents to the average similarity among relevant-nonrelevant document pairs. This average can be computed for a given query or over a set of queries. If the cluster hypothesis holds, the average similarity among relevant documents should be substantially larger than the average over relevant-nonrelevant pairs. A second approach,
due to Voorhees, is to determine for each document relevant to a given query how many of its nearest neighbors are also relevant to the query. In her experiments, Voorhees [TR 85-658] considered the five nearest neighbors to each relevant document. These two methods both require that a query or set of queries be applied to the collection and that relevance judgments be applied to the documents retrieved by these queries. The assumption is made that the results for the given queries characterize the given collection in the sense that other queries applied to the collection will give similar results. A third approach, due to El-Hamdouchi and Willett [JIS, 1987] depends entirely on properties of the collection itself, or more precisely on the terms that index the documents in the collection. They calculate a term density, defined as the number of occurrences of all index terms in the collection (the number of postings) divided by the product of the number of documents in the collection and the number of unique index terms. This density is a measure of how densely populated the term-document matrix is. The theory is that the greater the term density, the more frequently documents will share terms, and hence the better a clustering can represent degrees of similarity between documents. In a reported comparison of these methods, the term density measure correlated best with effectiveness of cluster searching. [Willetts, IP&M, 1988]

As the size of distributed collections and the corresponding size of filtering sets grow, the application of clustering to user interactive browsing also grows in importance. A number of the clustering methods described above, are specifically aimed at this application domain. Hence, some evaluations of effective clustering have also been aimed at this application. Browsing experiments have been conducted to evaluate the effectiveness of clustering for this purpose. These experiments are discussed further in the section below on User Interaction. However, one test of filtering effectiveness [Zamir et al., SIGIR ’98] that simulates browsing will be discussed here. This test compared the STC clustering method (described in the preceding section) against several heuristic clustering methods (described in the section on Heuristic Methods) and $O(N^2)$ methods (discussed in the section on Complete Methods). Specifically, it compared STC against four linear-time heuristic methods: Single-Pass, K-means (this is the Rocchio method), Buckshot, and Fractionation), and one $O(N^2)$ method: Group-Average Hierarchical Clustering (GAHC).
The strategy adopted by Zamir et al. is based on results reported by researchers who conducted actual browsing experiments. These experiments indicate that a user is usually (about 80% of the time) able to select the cluster containing the highest proportion of documents relevant to her need, on the basis of the cluster labels or summaries provided to her. Hence, Zamir generated 10 queries, retrieved documents from the Web for each of those queries, and then manually generated human relevance judgments for each of the 10 filtering sets, relative to the query for which it was retrieved. Then, they clustered each of the filtering sets using each of the cluster methods, setting parameters as appropriate so that 10 clusters were generated for each filtering set/cluster method pair. Then, for each filtering set and cluster method, they automatically selected the “best” cluster, i.e., the cluster containing the highest proportion of relevant documents, then the next best, and so on, until they had selected clusters containing 10% of the documents in the “collection,” i.e., in the given filtering set. This was based on the assumption, noted above and borne out to some extent in practice, that users can select the best clusters on the basis of their labels or summaries. In all cases, the cutoff was 10% of the documents in the given set; this meant that the cutoff might occur in the middle of a cluster, even in the middle of the first cluster, if that cluster was large for a given cluster method. The resulting 10% documents were then ranked, and the average precision computed, averaged over all 10 collections. (Since STC supports document overlap, a given document might appear in two or more selected clusters. For purposes of ranking, such duplicates were discarded.) Equalizing the number of clusters generated, and the number of documents ranked, across methods and collections, allowed for a fair comparison of cluster methods. Note that Zamir ranked documents by cluster, i.e., the documents in the best cluster were ranked higher than the documents in the next best cluster, and so on. Zamir et al. do not specify how they ranked documents within a given cluster. However, Hearst et al. rank documents within a cluster using two different criteria: closeness to the cluster centroid, and similarity to the original query.

The results reported by Zamir et al. show that STC out-performed the other methods, even the GAHC method, by a significant margin. GAHC out-performed the other methods, which is not surprising considering that it is an $O(N^2)$ method (and consequently far too slow for interactive use, and even for off-line use of large collections). It is striking that STC, an $O(N)$ and incremental method, out-performed
GAHC, which is neither! Zamir et al. concede that their results are preliminary; indeed, the title of their paper refers to the reported study as a “feasibility demonstration.” The results are preliminary for (at least) four reasons. First, the results were obtained from non-standard, e.g., non-TREC, collections, retrieved from the Web by 10 arbitrary queries. (This was deliberate, since Web filtering is the intended application of STC.) Second, the resulting collections were (relatively) small, e.g., 200 documents each. (On average, there were about forty relevant documents for each query.) Third, the relevance judgments were generated by the researchers rather than by independent judges, as in TREC. Fourth, the study did not use actual human users, performing actual interactive browsing. However, their system, MetaCrawler STC, has been fielded on the Web, so that statistics can be gathered from actual users. The data set employed is also being published on the Web, so that other researchers can replicate and validate these experiments.