Learned Vector-Space Models for Document Retrieval

William R. Caid
HNC, Inc.
San Diego, CA 92121

Susan T. Dumais[1]
Bellcore
Morristown, NJ 07960

Stephen I. Gallant
Belmont Research, Inc.
Cambridge, MA 02140

1. General Approach

Bellcore's Latent Semantic Indexing (LSI) system and HNC's MatchPlus system represent two attempts to model and exploit the inter-relationships among terms to improve information retrieval. Most information retrieval methods depend on exact matches between words in users' queries and words in documents. Typically, documents containing one or more query words are returned to the user. Such methods will, however, fail to retrieve relevant materials that do not share words with users' queries. One reason for this is that the standard retrieval models (e.g., Boolean, standard vector, probabilistic) treat words as if they are pairwise orthogonal or independent, although it is quite obvious that they are not. Consider, for example, the terms "automobile", "car", "driver", and "elephant". The terms "automobile" and "car" are synonyms, "driver" is a related concept, and "elephant" is pretty much unrelated. In most retrieval systems the query "automobile" is no more likely to retrieve an article about cars than one about elephants, if neither author uses precisely the term automobile. It would be preferable, however, if a query about automobiles would retrieve articles about cars and even articles about drivers to a lesser extent. A central theme of both the LSI and MatchPlus methods is that term-term inter-relationships like these can be explicitly modeled in the representation and automatically used to improve retrieval.

Thesauri, concept spaces, and statistical term co-occurrence information have been used to expand queries (or documents), with the goal of improving recall. Automatic query expansion, however, has not been very successful. Unless term expansion is done very carefully, precision often suffers. In addition, many thesauri and concept spaces are manually constructed for each new domain and cover only a limited number of term-term relationships like synonymy or "is a" relationships, when many other relationships might be useful for retrieval. There have also been some more general attempts to incorporate term dependencies into document representation and retrieval (Salton et al. 1982; Wong et al. 1987), but these ideas have not been applied to large collections like TREC.

LSI and MatchPlus examine the similarity of the "contexts" in which words appear, and create a feature space representation in which words which occur in similar contexts have vector representations that are near each other. The methods first create a representation which captures the similarity of usage (meaning) of terms and then use this representation for retrieval. The details of what "contexts" are and how they are used to construct the feature space differ for the two methods and will be described more completely in sections 2 and 3 below. Roughly speaking, the word "automobile" will occur in the context as terms like car, motor, model, vehicle, chassis, carmakers, dealerships, minivan, Chevrolet, etc.

[1]To whom correspondence should be addressed.

Importantly, the word "car" will also be used with many of these same words (i.e., in many of the same contexts as "automobile"). The contexts for "driver" will overlap to a lesser extent, and the contexts for "elephant" will be largely unrelated. The derived feature space reflects these inter-relationships. LSI uses a method from linear algebra, singular value decomposition (SVD), to discover the important associative relationships. MatchPlus uses a bootstrapping method based on adaptive neural network learning laws to derive context vectors. It is not necessary to use any external dictionaries, thesauri, or knowledge bases to determine these word associations. Both methods learn associations that are specific to the corpus of text being analyzed. They are also completely automatic, and widely applicable to text databases including to different languages.

The LSI and MatchPlus feature spaces contain many fewer dimensions (typically around 300) than there are unique words in the database (hundreds of thousands in the case of TREC). Because there are many fewer dimensions than unique words, words cannot be independent. Each word is represented as a 300-dimension vector, with its value on each dimension reflecting how much that dimension or feature describes it. Words which occur in similar contexts will have similar feature values and will thus be near each other in the space. Documents and queries are also represented in the same vector space. Both documents and queries are located at the weighted average of their constituent term vectors. For document retrieval, the query vector is compared to the document vectors, and documents are ranked by their similarity to the query. One important consequence of the feature space representation is that a query can be similar to a document even if they share no words in common. Since terms, documents and queries are represented in the same vector space, it is also easy to compute other similarities as well - terms can return nearby documents or terms, and documents can return similar terms or documents. This flexibility means that it is easy to implement relevance feedback and to apply the methods to routing (filtering) and labeling tasks.

Another important consequence of a vector representation is that such representations are ideal for neural network learning algorithms. These algorithms give a way to automatically refine (or generate) queries, based upon a training collection of documents that have previously been labeled as relevant or non-relevant.

In summary, LSI and MatchPlus capture the non-independence of term usage in a text corpus and exploit this to improve retrieval. In both cases, the inter-relationships are derived by an automatic computer analysis of the database texts, and the derived feature space is used to represent the similarity of meaning of terms and to retrieve documents.

2. Overview of Latent Semantic Indexing (LSI)

The LSI model has been described in detail elsewhere (Deerwesteret al., 1990; Dumais, 1991; and Furnas et al., 1988) so we will only briefly review the main ideas in this paper. LSI is an extension of the vector retrieval method (e.g., Salton & McGill,1983) in which the dependencies between terms are explicitly taken into account in the representation and exploited in retrieval. We assume that there is some underlying or "latent" structure in the pattern of word usage across documents, and use statistical techniques to estimate this latent structure. A description of terms, documents and user queries based on the underlying, "latent semantic", structure (rather than surface level word choice) is used for retrieving information.

Latent Semantic Indexing uses singular-value decomposition (SVD), a technique closely related to eigenvector decomposition and factor analysis (Cullum and Willoughby, 1985), to model the associations among terms and documents. As is the case with the standard vector method, LSI begins with a large term-document matrix in which the cell entries represent the frequency of a term in a document. This frequency matrix is then transformed using appropriate term weighting and document length normalization[2]. If there were no correlation between the occurrence of one term and another, then there would be no way to use the data in the term-document matrix to improve retrieval. On the other hand, if there is a great deal of structure in this matrix, i.e., the occurrence of some words gives us a strong clue as to the likely occurrence of others, then this structure can be modeled and we use the SVD to do so.

Any rectangular matrix, *X*, for example a $t \times d$ matrix of terms and documents, can be decomposed into the product of three other matrices:

$$\underset{t\times d}{X} = \underset{t\times r}{T_0} \bullet \underset{r\times r}{S_0} \bullet \underset{r\times d}{D_0'}$$

such that T_0 and D_0 have orthonormal columns, S_0 is diagonal, and *r* is the rank of *X*. This is so-called *singular value decomposition* of *X*.

If only the *k* largest singular values of S_0 are kept along with the corresponding columns in the T_0 and D_0 matrices, and the rest deleted (yielding matrices *S*, *T* and *D*), the resulting matrix, $\hat{X}$, is the unique matrix of rank *k* that is closest in the least squares sense to *X*:

$$\underset{t\times d}{X} \approx \underset{t\times d}{\hat{X}} = \underset{t\times k}{T} \bullet \underset{k\times k}{S} \bullet \underset{k\times d}{D'}.$$

The idea is that the $\hat{X}$ matrix, by containing only the *k* largest independent linear components of *X*, captures the major associational structure in the matrix. These statistically derived indexing dimensions are more reliable than the original term descriptors for retrieval. It is this reduced model, usually with $k \approx$200-300, that we use to approximate the term-document association data in *X*. Note that $\hat{X}$ is only an approximation to the original term-document matrix, *X*. Many of the 0's in the original matrix are now non-zero, and this is what we want.

Since the number of dimensions in the reduced model (*k*) is much smaller than the number of unique terms (*t*), words will not be independent. If two terms are used in similar contexts (documents), they will have similar vectors in the reduced-dimension LSI representation. Instead of representing documents and queries directly as vectors of independent words, LSI represents them as continuous values on each of the *k* derived orthogonal indexing dimensions. In this reduced model, the closeness of documents is determined by the overall pattern of term usage, so documents can be near each other regardless of the precise words they contain, and their description depends on a kind of

[2]Even though we talk about term-document matrices for simplicity, LSI can be applied to any descriptor-object matrix. Although we typically use only single terms to describe documents, phrases could also be included in the matrix. Similarly, an entire document is usually the text object of interest, but as more and more full-text materials become available, smaller, more topically coherent units of text (e.g., paragraphs, sections) may be a more appropriate unit of analysis than the full document. Paragraphs, for example, would then replace documents as columns of the matrix. Many groups have used document components to good advantage in TREC. Regardless of how the original matrix is derived, a reduced-dimension approximation can be computed. The important idea in LSI is to go beyond the original descriptors to more reliable statistically derived indexing dimensions.

consensus of their term meanings, thus dampening the effects of polysemy. In particular, this means that documents which share no words with a user's query may still be near it if that is consistent with the major patterns of word usage.

One can also interpret the LSI representation geometrically. The result of the SVD is a vector representing the location of each term and document in the *k*-dimensional feature space. The locations of term vectors are given by rows of the T matrix, and the location of documents are given by rows of the D matrix. The document locations correspond to the location given by the weighted average of the words in the document: $D_i = X_i' T S^{-1}$. A query is located in the same fashion, at the centroid of its corresponding term vectors: $D_q = X_q' T S^{-1}$. The dot product or cosine between vectors in the space is used to compare objects. For document retrieval, a query vector is compared to all document vectors and they are ranked by their similarity to the query. However, since both term and document vectors are represented in the same space, similarities between any combination of terms and documents can be easily obtained, and these are useful for a variety of different applications.[3] The similarity between terms *i* and *j* is given by: $TS_i \bullet TS_j$, and the similarity between documents *i* and *j* is given by: $DS_i \bullet DS_j$. Comparing term i and document j is a little different: $T\sqrt{S_i} \bullet D\sqrt{S_j}$. See Deerwester et al., 1990 for details. (The standard vector method has a similar geometric interpretation. In this case, however, a dimension corresponds to a term. The number of dimensions is equal to the number of terms, and terms are independent.)

Unlike many other applications of factor analysis or SVD, there is no need to interpret the dimensions or factors. The similarity between objects can easily be computed without describing the factors, and this is all that is needed for retrieval.

We use the term "semantic" indexing to describe our method because the reduced representation captures the major associative relationships between terms and documents. The LSI technique captures this structure better than simple term or document clustering, because the *k*-dimensional vector space offers much greater representational richness than the hierarchical structures typically used in clustering. LSI partially overcomes some of the deficiencies of assuming independence of words, and provides a way of dealing with synonymy and other associative relationships automatically without the need for a manually constructed thesaurus.

It is important to note that all steps in the LSI analysis are *completely automatic* and involve no human intervention. Documents are automatically processed to derive a term-document matrix. This matrix is decomposed by the SVD software, and the resulting reduced-dimension representation is used for retrieval. While the SVD analysis is somewhat costly in terms of time for large collections, it need be computed only once at the beginning to create the reduced-dimension database. (On a Sparc 10, the SVD takes only about 2 minutes for a 2k $\times$ 5k matrix, and this time increases to about 18 hours for a 60k $\times$ 80k matrix.)

The LSI method has been applied to many of the standard IR collections with favorable results. Using the same tokenization and term weights, LSI has equaled or outperformed standard vector methods in almost every case, and was as much as 30% better in some cases (Deerwester et al., 1990). As with the standard vector method, differential term

[3]Terms and documents are represented in different, though related, *k*-dimensional vector spaces. More specifically, the diagonal matrix *S* specifies the change in basis that transforms the document vector space into the term vector space.

weighting and relevance feedback both improve LSI performance substantially (Dumais, 1991).

3. Overview of MatchPlus

Like LSI, MatchPlus uses reduced dimensional vectors called *context vectors*. Previous descriptions of the context vector approach to word sense disambiguation and document retrieval have appeared in (Gallant 1991a, 1991b; Gallant, Caid, et. al. 1993, 1994; Caid and Carleton, submitted).

Among the similarities between LSI and MatchPlus are:

- fully automated system creation
- ability to compare any two objects selected from {terms, documents, and queries}, and
- applicability of neural network algorithms and other vector manipulations for improving/replacing original queries, based upon user relevance judgments.

There are, however, two main differences in the models. Context vector methods use *exactly the same* vector space for all operations, namely a vector space based upon the terms[4]. After an automated bootstrapping process generates vectors for terms, then query and document context vectors are simply computed as weighted sums of term context vectors. This makes it easy to combine terms, paragraphs, and/or documents into a single query, if desired.

A second difference is the bootstrapping process used to generate term context vectors. MatchPlus uses *local context*, i.e. nearby words, to generate term vectors rather than *document-wide context* as in LSI. The reason for this is to give nearby words more influence than far-away words when creating term representations that must capture a notion of similarity of use among terms. A drawback, however, is that bootstrapping algorithms make one or more passes over the corpus (or a subset of the corpus used for creating stem vectors), rather than working with document-wide term summary statistics like LSI[5].

An advantage to making several passes over the corpus is that it allows “higher order” relationships to be found and represented. For example, if *car* is often used with *race*, and *race* is often used with *Indianapolis 500*, and *Indianapolis 500* is often used with *Indiana*, then MatchPlus bootstrapping might generate some similarity between vectors for *car* and *Indiana*.

Finding a good bootstrapping algorithm that captures the “right amount” of similarity of usage and gives good performance is a subtle experimental task. The current version of bootstrapping uses a proprietary two-pass neural network-based approach that generates stem vectors so that pairs of stems are either related (dot product >> 0) or nearly orthogonal (dot product ~0).

---

[4]It would be possible to modify LSI to use a similar term-based representation for all objects.

[5]On the other hand, bootstrapping scales linearly with corpus size, while SVD algorithms scale much worse. On the third hand, training corpus *subsets* seem to work well enough that SVD/bootstrapping time is not a central issue.

For system development and tuning, it has been helpful to look at the list of closest terms to a target term to see how well similarity of usage was captured by various adjustments and modifications. For example Figure 1 shows some learned relationships for the term *Hezbollah* and some phrases that appear in the TREC corpus. Note that *Mousawi*, the Wall Street Journal spelling, and *Musawi*, the AP spelling, were both associated terms. Conventional retrieval systems would treat these two terms as unrelated. Thus context vectors provide an interesting approach to linkage analysis.

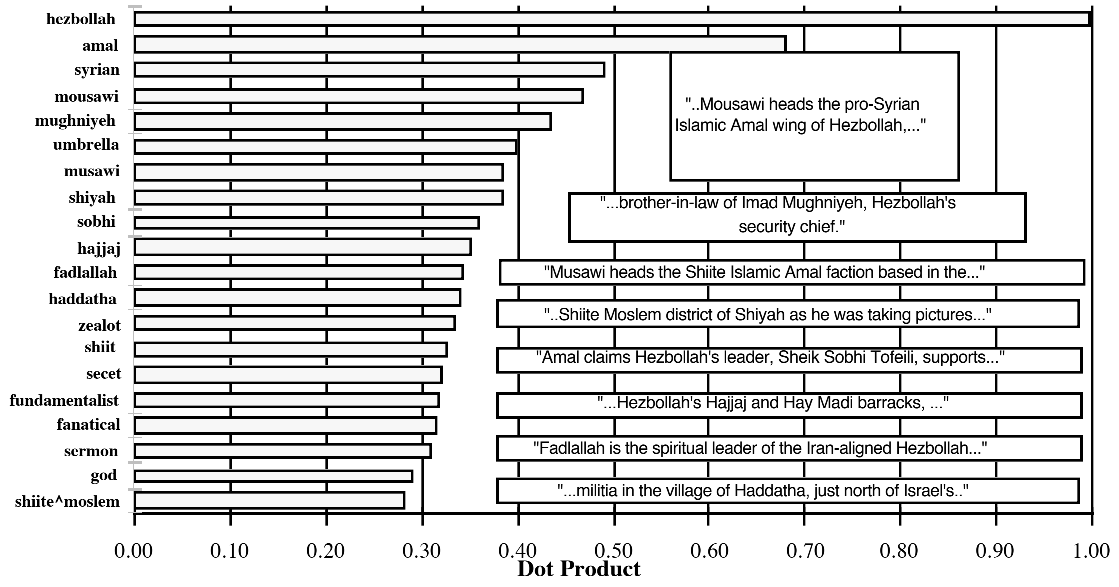

Figure 1: Learned relationships for *Hezbollah* with samples of surrounding context (Caid & Carleton).

<u>4. LSI and TREC</u>

We used TREC as an opportunity to "scale up" our tools, and to explore the LSI dimension reduction ideas using a very large and diverse corpus of word usage data. We were quite pleased that we completed both routing and adhoc queries with no major modifications to the LSI software. Most importantly, we were able to compute the SVD of 50k $\times$ 100k matrices without numerical or convergence problems on standard workstations. The main results from TREC-2 are summarized below. A more complete description of LSI experiments for TREC-1 and TREC-2 can be found in Dumais (1993, 1994).

4.1 Pre-processing

We used SMART[6] for pre-processing documents and queries. The result of document processing is a term-document matrix, in which each cell entry indicates the frequency with which a term appears in a document. The entries in the term-document matrix were then transformed using an "ltc" weight, which takes the log of individual cell entries, multiplies each term (row) by the IDF weight of the term, and then normalizes the document (column) length.

For disk 1 and disk 2, there were 742331 documents, 512251 unique stems, and 81901331 non-zero entries in the term-document matrix. To decrease the matrix to a size we thought we could handle, we removed tokens occurring in fewer than 5 documents. The resulting 742331 document $\times$ 104533 term matrix was the starting point for results reported in this paper. The "ltc" weights were computed on this matrix.

4.2 SVD analysis

The SVD program takes the ltc term-document matrix as input, and calculates the best "reduced-dimension" approximation to this matrix. The number of dimensions, k, was between 200 and 300 in our TREC experiments. The specific values of k used below were determined by available memory. We suspect that better performance would be obtained with somewhat larger values for these large, diverse collections.

We were recently able to compute the SVD of the full 742k $\times$ 104k matrix described above. For the official TREC runs, we used a sample of documents from this matrix. For the routing experiments, we used the subset of 68385 unique training documents for which we had relevance judgments (disk 1 and disk 2). The SVD analysis was computed on the relevant 68385 document $\times$ 88112 term subset of the above matrix, and a 204-dimension approximation was used. For the adhoc experiments, we took a random sample of 70000 documents. A 199-dimension SVD approximation was computed for this 70000 document $\times$ 82968 term matrix. The 672331 documents not included in this sample were "folded in" to the 199-dimension LSI space, and the adhoc queries were compared against all 742k documents (disk 3). The vector for the "folded in" document was located at the centroid of the all term vectors in the document. (Note that these are very small samples of the database. The fact that they work quite well, as we demonstrate below, is quite promising.)

4.3 TREC-2: Routing experiments

For the routing queries, we created a filter or profile for each of the 50 training topics. The 336306 new documents from disk 3 were automatically pre-processed as described above. A new document's vector was located at the centroid of its term vectors, and compared to each of the 50 routing filter vectors. The 1000 best-matching documents were returned for each topic. We submitted results from two sets of routing queries. In one case *routing_topic (lsir1)*, the filter was based on just the topic statements. This method makes *no* use of the training data, representing the topic as if it was an adhoc query. In the other case *routing_reldocs (lsir2)*, the filter was derived by taking the vector sum or centroid of all relevant training documents for each query. This is a somewhat unusual variant of

[6]The SMART system (version 11.0) was made available through the SMART group at Cornell University. Chris Buckley was especially generous in consultations about how to get the software to do somewhat non-standard things.

relevance feedback; we *replace* (rather than combine) the original topic with relevant documents. These two extremes provide baselines against which to compare other methods for combining information from the original query and feedback about relevant documents.

The main routing results are shown in Table 1. The lsir1 and lsir2 runs differ only in how the profile vectors were created - using only words in the topic statement for *lsir1 routing_topic*, and using all relevant training documents for *lsir2 routing_reldocs*. Not surprisingly, the lsir2 run, which takes advantage of the known relevant documents, is better on all measures of performance. The improvement in average precision is 31% (.2622 vs. .3442), and there is an average of 1 additional relevant document in the top 10 using the lsir2 filters.

	lsir1	**lsir2**	**r1+r2**
	(topic wds)	**(rel docs)**	**(sum r1 r2)**
Rel_ret	6522	7155	7367
Pr at 10	.5480	.6660	.6620
Pr at 100	.3799	.4524	.4394
Avg prec	.2622	.3442	.3457
Q >= Median	27 (4)	40 (9)	42 (6)
Q < Median	23 (0)	10 (0)	8 (0)

Table 1: LSI Routing Results. Comparison of topic words vs. relevant documents as routing filters.

We also tried a simple combination of the lsir1 and lsir2 profile vectors, in which both components had equal weight. The results of this analysis are shown in the third column of Table 1, labeled r1+r2. This combination does somewhat better than the centroid of the relevant documents in the total number of relevant documents returned and in average precision. (We returned fewer than 1000 documents for 5 of the topics and not all documents returned by the r1+r2 method had been judged for relevance, so we suspect that performance could be improved a bit more.) The r1+r2 method is a variant of *relevance feedback*, and provides benefits that are comparable in size to those observed for more standard relevance feedback in TREC-2 (Buckley et al., 1994).

Compared to other TREC-2 systems, LSI does quite well, especially for the routing_reldocs (lsir2) and r1+r2 runs. In the case of lsir2, LSI is at or above the median performance for 40 of the 50 topics, and has the best score for 9 topics. LSI performs about average for the routing_topic (lsir1) run even though no information from the training set was used in forming the routing vectors in this case.

4.4 TREC-2: Adhoc experiments

A query vector was located at the vector sum of all words in all topic fields. The 1000 best-matching documents for each query were returned. We submitted two sets of adhoc queries - lsiasm and lsia1. We had intended to compare a standard vector method using SMART (lsiasm) with our LSI analysis using the same pre-processing (lsia1). Unfortunately, there were some errors in translating from LSI numbers to document names, so the official lsia1 results are incomplete and misleading. We have corrected this translation problem, and the correct results are labeled lsia1*. These results are summarized in the first two columns of Table 2.

	Full Set		Subset w/ Judgments		Subset w/ Judgments Summary Topics	
	lsiasm	lsia1*	lsiasm	lsia1*	lsiasm	lsia1*
Rel_ret	7869	6987	9493	9596	8043	8676
Pr at 10	.5020	.5100	.5020	.5220	.3420	.3680
Pr at 100	.4306	.3922	.4306	.4466	.3344	.3710
Avg prec	.3018	.2505	.3700	.3789	.2589	.3008
Q >= Median	37 (2)	25 (1)				
Q < Median	13 (0)	25 (0)				

Table 2: LSI Adhoc Results. a) Comparison of standard vector method with LSI (corrected version, but missing relevance judgments) - columns 1 and 2. b) Comparison of standard vector method with LSI using only documents for which relevance judgments were available - columns 3 and 4. c) Comparison of standard vector method with LSI using only documents for which relevance judgments were available, using only the Summary field - columns 5 and 6.

In terms of absolute levels of performance, both lsiasm and lsia1* are about average. The SMART results (lsiasm) are somewhat worse than the SMART results reported by Buckley et al., Fuhr et al., or Voorhees in TREC-2, but this is because we used a different term weight and did not include phrases. Much to our disappointment, the reduced-dimension LSI performance is somewhat worse than the comparable SMART vector method. However, it is important to realize that many of the documents returned by lsia1* were not judged for relevance because they were not submitted as an official run! For lsiasm, the 5000 documents corresponding to the top 100 documents for each query were all judged since this was an official run, and 2153 were relevant. For lsia1*, only 4073 documents of the corresponding documents were judged and almost as many, 2122, were relevant. It is quite likely that some of the 927 unjudged documents in lsia1* are relevant.

Because the missing relevance judgments make direct comparisons between SMART and LSI difficult, we decided to look at performance for just the 38175 unique documents for which we had adhoc relevance judgments. These results are shown in the third and fourth columns of Table 2. The most striking aspect of these results is the higher overall level of performance. This is to be expected since we are only considering the 38175 documents for which we have relevance judgments, and there are 700k fewer documents than in the official results. Considering only this subset of documents, there is a small advantage for LSI compared to the SMART vector method. This advantage is much smaller than we have previously seen with smaller IR test collections. The most likely reason for this is that the TREC topics are much richer and more detailed descriptions of searchers' needs than are found in typical IR requests. The average TREC query contains 51 unique words, and many of these are very specific names. Since LSI is primarily a recall enhancing method it has little effect on these already very rich queries. This is much the same conclusion that groups who tried various kinds of query expansion reached.

We tried one additional analysis using the new "Summary" field for each topic. The Summary field alone is used as a much shorter query. These results are summarized in the last two columns of Table 2. As expected, overall performance is lower than with the complete topics. More interestingly, the difference between LSI and the standard vector method is now larger - 16% in average precision. This is still a somewhat smaller advantage than we have seen in previous experiments, but even the summary queries have an average of 11 unique words.

LSI can offer performance advantages compared to a standard vector method for the large TREC collection, and the advantages are larger with shorter queries. The exact nature of this advantage (e.g., which documents are retrieved by LSI but not the standard vector method) needs to be examined in more detail.

4.5 Improving LSI Performance

Although we were quite pleased that we were able to use the existing LSI tools successfully in TREC, there is still a good deal of room for improvement in retrieval accuracy and efficiency. First, there are many variations in weighting, number of dimensions, use of phrases, etc. that we need to examine. Second, we have just begun to look in detail at retrieval failures, and these analyses should be quite informative. A lack of specificity appears to be the main reason for false alarms (highly ranked but irrelevant documents). This is not surprising because LSI was designed as a recall-enhancing method, and we have not added precision-enhancing tools although it would be easy to do so. Many of the misses (documents returned by other systems but not LSI) represent articles that were primarily about one topic but contained a small relevant section. We suspect we could reduce this problem by using smaller contexts than entire documents. Third, we would like to be able to compute larger SVDs, although we have been quite pleased with the results of our sampling. Finally, the efficiency of the retrieval/matching algorithm needs to be improved for handling large databases, and parallel algorithms are quite promising.

5. MatchPlus and TREC

5.1 System Generation

A subset of the TREC/TIPSTER corpus was chosen as a training corpus consisting of approximately 320 Megabytes of sampled text from the 1,800 Megabytes of text on disks 1 and 2. This training corpus contained 120,000 stems (out of 620,000 stems total). The MatchPlus neural network learning algorithm was used to determine a context vector representation for each of the 120,000 stems, based upon their context in the training corpus, after which document context vectors were computed. Note that only 18% of the stems in the two disks was used as the basis for the generation of document context vectors; remaining (low frequency) stems were assigned zero vectors. The resulting stem and document context vectors were used for the TREC retrieval and routing experiments.

5.2 Ad Hoc Retrieval Experiments

Two runs were submitted as part of the ad hoc retrieval experiments. The first run was a totally automated run, the second involved relevance feedback. In the totally automated run, the entire topic description was converted to a context vector as a query, with a weight of 2 applied to the "Topic" section. In addition, a $Match(4) filter was applied to terms that occurred in the "Concept" section of the topic description. This filter had the effect of placing those documents that contained at least 4 of the concept terms above all other documents in the ranking. For all documents, context vector dots products were computed to determine document relevance and determine those documents in the submission list. All processing was totally automated.

For the relevance feedback experiment, a totally automated run was performed as described above. Then the top 20 documents for each of the 50 topics was read and judged relevant/not relevant. The context vector for all relevant documents for a topic were summed, normalized and added to 0.7 times the totally automated query vector. The resulting vector was used as the basis for a context vector-only (no $Match) and all corpus documents were ranked by dot product. (The 0.7 factor was empirically determined through experimentation with other data.) Results for both ad hoc runs are shown in the table below.

5.3 Routing Experiments

Two routing runs were submitted. Both submissions were based on neural network learning techniques used to determine system weighting based on judged documents. In both runs, results from stem weight learning and full context vector learning approaches were combined with ad hoc processing runs to produce the final submission.

The *stem weight learning* approach used a perceptron learning variant to compute weights for terms in a query, with one weight associated with each term in the query. Every judged document in the corpus provided an example to the network training algorithm. The "Pocket Algorithm" (Gallant 1990) was used to train the networks for TREC. (Standard backpropagation of errors was also tried, but there was apparently insufficient non-linearity in the underlying problem to benefit from a more complex model.) When the learning was complete, the resulting weights were then used as term weights for normal context vector query processing.

The *full context vector learning* approach is similar to the stem weight technique. Here, we attempted to learn an entire query context vector rather that just weights for stems. In this approach, document vectors were used as the inputs to the network, and relevance of the associated document was used to determine the weight adjustment. Weight adjustment were performed for all relevance judged (relevant and non-relevant) documents. The resulting weights were then used directly as a routing query vector. Training was by the Pocket Algorithm.

5.3.1 Routing Submission 1: "Best Candidate"

The first submission was a mixture from 4 sources: the two neural network techniques described above, the fully automated query (the first ad hoc submission) and the relevance feedback query (the second ad hoc submission). For each of the 50 topic queries, the best scoring source was estimated using training data. Then the best candidate's top 1000 documents were then used as the submission for that topic. This was repeated for each topic.

5.3.2 Routing Submission 2: "Mix and Match"

The second routing submission was based on weighted "mix and match" approach. In this run, a submission for each topic was formed by combining the results from each of the four methods on a document-by-document basis. The "quality" of each source was determined based upon 11-point average precision/recall on a separate test corpus. Then, each document from each source was given a modified relevance score proportional to the quality estimate of that source and inversely proportional to document ranking. The final results list was formed by combining results document-by-document from each list.

Performance Metric	Ad hoc: Fully Automated	Ad hoc: Relevance Feedback	Routing: Best Candidate	Routing: Mix and Match
Rel_ret	6944	6911	5833	6407
Prec at 10	.5680	.5680	.6020	.6100
Prec at 100	.4520	.4784	.4228	.4090
Avg. Prec.	.2787	.2882	.2810	.2927

Table 3: MatchPlus Results Table for TREC-2.

These results demonstrate the viability of the context vector approach, and are especially encouraging given that the MatchPlus system is in its second year of existence.

6. Discussion

These experiments have pointed out an important issue: *how much learned similarity among terms is best?* There is a continuum from zero sensitivity (as in standard Boolean or SMART-like vector space models) to extreme sensitivity, where a document with no term in common with the query is often preferred to documents having many terms in common with the query. For example, consider a simple query such as “Paramount.” A Boolean system or SMART will first retrieve all documents containing this term, and the remaining documents will be judged equally relevant. A system such as LSI or MatchPlus will rank non-Paramount documents concerning “movies” or “Viacom” ahead of non-Paramount documents that do not contain these related terms, and this seems clearly beneficial. On the other hand, a document that is focused on Viacom might be ranked ahead of a document that mentions Paramount only one time. This may or may not be desirable, depending upon the needs of the person making the query. Moreover, if queries contain a large percentage of the key terms (as did the TREC corpus), this reduces the benefit of learned similarities.

In summary, the three main advantages to the learned vector approaches are (1) arguably improved precision, (2) suitability for learning algorithms, and (3) flexibility for other applications, including finding relations among terms and visualization of documents and terms.

The TREC experiments have demonstrated the computational feasibility of using automatically derived term-term similarities for very large corpora for retrieval. We look forward to continued development of these approaches.

7. References

Buckley, C. and Salton, G. Automatic retrieval with locality information using SMART. In D. Harman (Ed.) The First Text REtrieval Conference (TREC-1), NIST Special Publication 500-207, 1992, 59-72, 1993.

Caid, W.R. and Carleton, J.L. A Context Vector-Based Approach to Text Characterization and Retrieval. Submitted.

Cullum, J.K. and Willoughby, R.A. Lanczos algorithms for large symmetric eigenvalue computations - Vol 1 Theory, (Chapter 5: Real rectangular matrices). Brikhauser, Boston, 1985.

Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. Indexing by latent semantic analysis. Journal of the Society for Information Science, 1990, 41(6), 391-407.

Dumais, S. T. Improving the retrieval of information from external sources. Behavior Research Methods, Instruments and Computers, 1991, 23(2), 229-236.

Dumais, S. T. (1993). LSI meets TREC: A status report. In D. Harman (Ed.) *The First Text REtrieval Conference (TREC-1)*. NIST special publication 500-207, 137-152.

Dumais, S.T. (1994). Latent Semantic Indexing (LSI) and TREC-2. In D. Harman (Ed.). *The Second Text REtrieval Conference (TREC-2)*. NIST special publication 500-215, pp. 105-115.

Foltz, P. W. and Dumais, S. T. Personalized information delivery: An analysis of information filtering methods. Communications of the ACM, Dec. 1992, 35(12), 51-60.

Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T. K., Harshman, R. A., Streeter, L. A., and Lochbaum, K. E. Information retrieval using a singular value decomposition model of latent semantic structure. In Proceedings of SIGIR, 1988, 465-480.

Gallant, SI (1990) Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks* 1(2):179-192.

Gallant SI (1991a) Context vector representations for document retrieval. *AAAI-91 Natural Language Text Retrieval Workshop*: Anaheim, CA.

Gallant SI (1991b) A practical approach for representing context and for performing word sense disambiguation using neural networks. *Neural Computation* 3(3):293-309.

Gallant SI, Caid WR et al (1993) HNC's MatchPlus system. In D. Harman (Ed.) *The First Text REtrieval Conference (TREC-1)*. NIST special publication 500-207, pp. 107-111.

Gallant SI, Caid WR et al (1994) Feedback and Mixing Experiments With MatchPlus. In D. Harman (Ed.). *The Second Text REtrieval Conference (TREC-2)*. NIST special publication 500-215, pp. 101-104

Salton, G., Buckley, C. and Yu, C.T. An evaluation of term dependence models in information retrieval. In: Proceedings of the 5th Annual International ACM-SIGIR Conference, 1982, ACM, New York, 151-173.

Salton, G. and McGill, M.J. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

Wong, S.K.M., Ziarko, W., Raghavan, P.C.N, and Wong, P.C.N. On modeling of information retrieval concepts in vector spaces. ACM Transactions on Database Systems, 1987, 12(2), 299-321.