ornl

ORNL/TM-13094

OAK RIDGE
NATIONAL
LABORATORY

MARTIN MARIETTA

Toward a Multi-Sensor-Based Approach to Automatic Text Classification

V. R. Dasigi
R. C. Mann

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

ORNL/TM-13094

Computer Science and Mathematics Division

TOWARD A MULTI-SENSOR-BASED APPROACH TO AUTOMATIC TEXT CLASSIFICATION

V. R. Dasigi[+]
R. C. Mann

[+]Department of Computer Science and Information Technology, Sacred Heart University, Fairfield, CT.

DATE PUBLISHED - October 1995

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
managed by
Lockheed Martin Energy Systems
for the
U.S. Department of Energy
Under Contract DE-AC05-84OR21400

CONTENTS

ABSTRACT			v
1	INTRODUCTION & BACKGROUND		1
2	A MULTI-SENSOR NEURAL NET APPROACH		3
3	PROGRAM COMPONENTS		6
	3.1	The Lexical Analysis Group	8
	3.2	The Frequency Matrix Generation Group	9
	3.3	The SVD Group	10
	3.4	The LSI-Based Analysis and Neural Net Input Generation Group	12
	3.5	The Other Sensors Group	15
	3.6	The Miscellaneous Group	16
4	RESULTS		16
5	FUTURE WORK		20
ACKNOWLEDGEMENTS			22
REFERENCES			23
FIGURE 1			27
FIGURE 2			29

ABSTRACT

Many automatic text indexing and retrieval methods use a term-document matrix that is automatically derived from the text in question. Latent Semantic Indexing is a method, recently proposed in the Information Retrieval (IR) literature [Deerwester, et al., 90], for approximating a large and sparse term-document matrix with a relatively small number of factors, and is based on a solid mathematical foundation. LSI appears to be quite useful in the problem of text information *retrieval*, rather than text *classification*. In this report, we outline a method that attempts to combine the strength of the LSI method with that of neural networks, in addressing the problem of text classification. In doing so, we also indicate ways to improve performance by adding additional "logical sensors" to the neural network, something that is hard to do with the LSI method when employed by itself. The various programs that can be used in testing the system with TIPSTER data set are described. Preliminary results are summarized, but much work remains to be done.

1 Introduction & Background

With explosive growth of multi-media data repositories, rapidly increasing connectivity of computer networks and the emergence of and widespread access to the national information infrastructure, there is an urgent need for intelligent tools for access, analysis and filtering of multi-media information. According to news reports, the white house started receiving tens of thousands of messages per day through electronic mail soon after announcing its e-mail address to the public. Fast classification of such messages into categories of contemporary public interest (e.g., health care, taxes, national service, etc.) can be vital to the responsiveness of the administration. Significant progress has been made in document retrieval, going beyond the traditional methods of Boolean and exact keyword matching [Salton, 89] [CACM, 92] [IEEE Expert, 93] [Computer Journal, 92] [van Rijsbergen, 92]. Much work has been done in applying probabilistic methods to IR and by representing documents as vectors in an n-dimensional space [Turtle and Croft, 92]. Work in natural language-based methods, while quite impressive, often suffers from one major drawback - that of scaling up effectively [Smeaton, 92]. Integrated approaches that combine natural language methods with statistical and/or vector-based approaches appear to hold promise, but are relatively rare [Jacobs, 93].

An important first step in supporting automated information retrieval and classification is to derive the indexing information automatically. In the case of text documents, the conventional practice used to be to employ some controlled vocabulary defined by an expert. Instead, most contemporary approaches use terms contained in the text document itself, or their lexical or semantic variants, directly as indexes into the document. The "vector-based" approaches view documents as vectors of such terms. Thus, a "library" of documents would be a collection of term vectors, or alternately, a term-document matrix, where the entries represent the frequency of each term in

each document. Needless to say, such term-document matrices tend to be very sparse and very large.

Latent Semantic Indexing (LSI) is a method recently developed to capture the "latent semantic structure" hidden in the association of terms within documents, as indicated in a term-document matrix[Deerwester, et al., 90]. The large sparse matrix is reduced into three relatively small matrices (one of which is simply a diagonal matrix) by singular value decomposition (SVD) corresponding to the largest several singular values. The original sparse matrix can be approximated by the product of the smaller SVD matrices.

One advantage of this method is that there are potentially fewer elements to deal with after SVD than in the large term-document matrix, that is, the SVD version is more efficient with respect to space. However, once the fact is considered that the large term-document matrix happens to be very sparse, the space efficiency advantage becomes debatable, but there is at least one other very significant justification for performing the SVD. It turns out that a careful selection of the number of singular values included in the SVD, captures the "latent semantic structure" of the collection of documents [Deerwester, et al., 90]. While the SVD results in only an approximation to the original term-document matrix, it is claimed that the approximation gives better results than the original! The intuitive reason for this surprising anomaly is that the approximate version weeds out insignificant associations between terms and documents, and only represents the major associative patterns.

The research reported here represents an effort to go beyond the advantages of LSI, by combining its valuable ideas with the powerful pattern-matching and learning capabilities of neural networks. A major stumbling block in applying neural networks to most IR applications has been that the size of a typical IR problem results in impractically large neural networks. Typically, documents to be classified as well as retrieval queries are

represented as a set of terms, the size of which is at least in the thousands. In such networks there could be hundreds of thousands of connections, not to mention the complexity when lateral inhibition is added for a winner-takes-all effect, e.g., [Wilkinson and Hingston, 92]. Our preliminary results in addressing these issues are summarized.

2 A Multi-Sensor Neural Net Approach

In part, this work was motivated by the success of the Gene Recognition and Analysis Internet Link (GRAIL) system developed at the Oak Ridge National Laboratory. GRAIL is a pattern recognition system for identifying protein-encoding DNA sequences among anonymous sequences in higher eukaryotes, which may span tens to hundreds of "sparse" kilobases [Uberbacher and Mural, 91] [Xu, et al., 94]. A multi-layer feed-forward neural network receives inputs from several sensors that measure different characteristics of the signals or data sets to be analyzed. The net acts as a classifier and assigns the input pattern to a given number of classes. The net is trained using a set of known data patterns that are representative of the application domain. In its simplest form, GRAIL operates with sensors that give signals that are either "on" or "off", indicating the presence or absence of a particular pattern or characteristic. Sensors can also supply "analog" input signals to the integrating network. The neural net represents a reliable mechanism to integrate the information from multiple sources to form a combined best estimate of the true classification decision. The term "sensor" is interpreted in a broad sense. It can encompass a real physical sensor device or an algorithm that computes a feature of a signal acquired by a physical sensor. The term "logical sensor" is used in the latter context.

We start with the hypothesis that a GRAIL-like system would be very appropriate for classification and filtering of English text documents: It is

hypothesized that the eventual system, being neural network-based, would be capable of integrating in a systematic way existing and new approaches as required by the application. The GRAIL-type system can integrate different kinds of sensors, e.g., statistical and syntactic sensors as well as simple keyword sensors, and other standard techniques already in use by the analyst community, such as LSI or other approaches (e.g., [Salton, et al., 94] and [Damashek, 95]). Initially, the system would be capable of reliably classifying text documents into different classes, and can be eventually modified to retrieve information that matches a specified profile.

Specifically, in this initial effort, we focused on two main goals. First, create input to a neural network that is LSI-based, so that the size of the neural net will be practical, and it can be trained without much difficulty. Further, a second goal is to see if additional sensors can be added easily to the neural net input, to give improved results. The relationship between the LSI component and the neural network is symbiotic. The LSI-based input makes for the input to the neural network to be of a much smaller size than would a long term vector be. Further, LSI is based on a solid mathematical theory, which adds strength to the resulting system. For its part, the neural network makes it easy to add trainability to the LSI-based method, and also makes it possible to integrate other sensors to complement or supplement the LSI-based input.

It has been indicated above that an "LSI-based" input vector is used with a neural network to contain its size. This technique needs some elaboration. A straightforward, but simple-minded input vector for the neural network would be a document represented as a vector of *all possible* term frequencies. Most sets of documents of reasonable size (such as newspaper or magazine stories, novels, scientific articles, etc.) comprise thousands of distinct word roots, which may be viewed as primitive terms. Thus, the size of the input term vector in a simple-minded representation would be a few thousand. LSI

work suggests a way to represent a document using around a hundred "factors", derived from the much longer term vector and the SVD of a "reference matrix"[1].

Actually, the developers of LSI indicate that a query may be viewed as a pseudo-document and may be represented by a vector of a chosen number of factors, so it may be placed in the same vector space as regular document vectors [Deerwester, et al., 90]. This is done as follows. First a reference term-document sparse matrix X is derived from the library of documents that are of interest. This matrix is split into three matrices by SVD, so that

$X = T.S.D'$

Here, X is a txd matrix, where t is the numbers of distinct terms (word roots) and d is the number of documents in the reference collection. The order of T is txk, that of S, which is a *diagonal* matrix, is kxk, and that of D is dxk, where k is the chosen number of factors. (LSI research indicated that a choice of k around a hundred is very effective.) Now, the pseudo-document vector D_Q corresponding to a 1xt query vector Q may be derived simply as:

$D_Q = Q.T.S^{-1}$

In this work, we use this same idea to squash any 1xt document vector into a 1xk vector that serves as input to the neural network. *The only care that must be exercised is to make sure that the reference term-document matrix that is used as the starting point is one that "adequately" represents all concepts of interest.* Note that this requirement is no more stringent than would be required in the standard LSI approach.

Figure 1 shows a high level view of how the proposed method works. The input to the system is an individual document that needs to be classified into one of several categories. Different logical sensors are applied to the docu-

[1]A *reference matrix* is the term-document matrix of a *reference library/collection* of documents. A reference library is simply the collection of documents that "adequately" represents all concepts of interest.

ment, constituting different kinds of preprocessing to derive salient features. One such sensor of interest to this work is one based on the term vector representing the input document. This 1xt vector gets transformed into a 1xk vector, using an SVD-based transformation, as explained above. The features derived by the logical sensors constitute input to a neural network that has already been trained. The output is an indication of the category to which the document belongs. This approach may be adopted to perform other related functions, e.g., document filtering or retrieval.

3 Program Components

Any serious IR system has to deal with vast amounts of data. Evaluation of such a system should involve at least millions of bytes of data (and more recently, giga- and even tera- bytes). To this end, a license to use the TIPSTER data source, distributed by the Linguistic Data Consortium, was obtained. The TIPSTER collection consists of about 3 billion bytes of SGML-coded text data in compressed form, coming from different sources such as the AP News wire, the Wall Street Journal, Ziff-Davis publications, etc., distributed on three CD-ROMs.

In this preliminary effort, the focus was exclusively on the AP news wire stories. On the first CD-ROM, aside from a large amount of non-AP news wire data, there is one compressed AP file for each day of the year 1989. Each such file contained several tens through a few hundreds of news stories and analyses, coded in SGML, with the total file size in the hundreds of thousands of bytee, up to about a megabyte. The programs work with such files, after they are uncompressed. The purpose of the multi-sensor neural net is to classify the news stories into one of ten ad hoc categories (the general/miscellaneous category includes any story that does not seem to belong to one of the others):

1. accidents and natural disasters
2. business and finance
3. crime
4. culture
5. general/miscellaneous
6. obituary
7. politics and government
8. science and technology
9. terrorism
10. weather

For the documents used for training and testing purposes, the categories of the documents as manually determined by a teacher are to be encoded in a file named `categories`. This file is described in more detail later.

The programs may be divided into a few groups, based on their function. They fall broadly into the following groups: *lexical analysis, frequency matrix generation, SVD group, LSI-based analysis and neural net input generation, other sensors* and *miscellaneous*. Each group is described below. A simple makefile, called `Makefile`, is written to capture the appropriate dependencies and generate the necessary executables. Figure 2 summarizes the relationship between the different groups of programs at a very high level.

3.1 The Lexical Analysis Group

The main purpose of this group of programs is to separate the individual news stories in each file, strip off the SGML codes, and also perform simple morphological stemming. As needed, the programs also filter a set of stop words out of each story document.

The details of the SGML coding are available in a document type definition available on the CD-ROM for each category of news source. It was found to be convenient to write a Lex source file for this purpose, mainly in view of the SGML coding. The unix tool Lex takes this source file as input and generates a C source program. This C program in turn takes as input a file conforming to the lexical description in the Lex source file, and produces the required output. There are three different Lex sources files (that are mostly similar) written for slightly different purposes: `aplex.in`[2], `aplexbasic.in` and `singlex.in`. From these Lex sources, using `Makefile`, one can respectively produce the executables `aplex`, `aplexbasic` and `singlex`.

All the executables accept an uncompressed AP news wire file containing a number of stories for a single day, coded in the predefined SGML format, as standard input by redirection. The executable `aplex` strips the SGML off, filters stop words, stems words using a simple Porter stemmer [Porter, 80], and writes the words of each individual story into a separate file. These files are named with sequential numbers. It expects the number for the first file to be specified a file named `lex.in`, and if such a file is not present, the starting file number defaults to 1.

The function of `aplexbasic` is somewhat simpler. All it does is to strip the SGML off, and separate the individual stories into separate files. These files are numbered as described above, with the difference that all of them contain the extension `.basic`. For example, the default name for the first file

[2]There are other Lex source files, namely, `wsjlex.in`, `doelex.in` and `zflex.in`, that have been written and are available, to serve purposes similar to `aplex.in`.

would be `1.basic`. The purpose of `singlex` is to take a single such file (the name of which ends in the `.basic` extension) as standard input by redirection, and produce the corresponding file (without the `.basic` extension) that contains a list of filtered and stemmed words, by output redirection. Thus, `singlex` may be used, for instance, to produce the file `1` from `1.basic`.

3.2 The Frequency Matrix Generation Group

The program `merge.c` goes through a specified sequence of files that have been generated by `aplex` as above. First, it derives individual term frequency distributions for each file, and then merges them into a single summary file. In doing so, it keeps track of unique terms, and it uses a simple format to squash multiple successive zero entries. (A lone 0 appears by itself, but if there are `n` (¿ 1) successive zeroes, they appear as `nz`.) This latter point is important considering that the term-document frequencies derived in this work were more than 98% sparse! the program asks for the starting and ending file numbers, which should be the names of files containing stop-word-filtered, stemmed terms. All the files in the specified inclusive range must be present. The resulting sparse term-document matrix is written into a file whose name is the same as the last file number, except for the extension `.1` that is added. If more files were later added beyond the "ending file" above, `merge`[3] has an option so it can continue just with the new files. It creates temporary files corresponding to each of the files in the specified range, with extensions `.2` and `.1`, but destroys all but the very last one of them, as indicated above.

Another program, `convert.c`, asks for the name of the sparse term-document matrix generated by `merge`, and performs a conversion on it to generate the file `matrix` that conforms to the SVD functions. The sparse

[3]The executables corresponding to a C program file are named the same as the source file, without the `.c` extension.

matrix is expected by the SVD functions in a "Compressed Column Storage format" [Berry, et al., 93], which is different from the straightforward representation indicated above. This conversion is required because the new format cannot be derived until the straightforward sparse matrix representation is completely known. The program `convert.c` is described in more detail later in Section 3.4, because it does more than just converting the matrix into a new format.

3.3 The SVD Group

The bulk of these programs was taken from the SVDPACKC package [Berry, et al., 93]. The SVDPACKC is a set of C functions for performing the singular value decomposition of a matrix, with special attention to sparse matrices, using eight different methods. Of all the available methods, the "Single Vector Lanczos" method (using an eigensystem adequate for determining a number of the largest singular values (and corresponding singular vectors)) was chosen, because of its performance.

The program, called `las2.c`[4], expects two input files: `matrix` and `lap2`. The file `matrix` is the sparse matrix represented in a compressed column storage format, ans is derived by `convert` from the output of `merge`. The other file, `lap2`, specifies the input parameters to `las2` on a single line, as follows (taken directly from [Berry, et al., 93]):

name lanmax maxprs endl endr vectors kappa, where

- *name* is a string defining the name of the data set.
- *lanmax* is an integer specifying the maximum number of Lanczos iterations allowed.

[4]See [Berry, et al., 93] for the naming convention and many other details.

- *maxprs* is an integer which indicates the number of singular triplets desired.
- *endl* and *endr* specify the two end points of an interval within which all unwanted singular values lie.
- *vectors* indicates if singular values and vectors are needed (TRUE) or just singular values (FALSE).
- *kappa* is the relative accuracy of Ritz values acceptable as singular values.

It is important to note that *maxprs* must not exceed *lanmax*, and also that the algorithm generally ends up computing a number of singular values different from *maxprs*. The number of computed singular values is really dictated by *lanmax*, which, when not large enough (although it *must* always be at least *maxprs*), may result in fewer than *maxprs* singular values. If it is large enough, it could sometimes even result in more than *maxprs* values, too.

The program *las2*, as modified, produces several output files: `lav2`, `lao2`, `T.S` and `T.SINV`. The file `lav2` is binary as described in [Berry, et al., 93]. Apart from some header information, it essentially contains the D matrix followed by the T matrix (referring to the end of Section 2). It is important to note that the values are written out in column major order, either t values or d values per column, k times (as defined in Section 2), and in *the order corresponding to increasing singular values*. Thus, the most important values appear at the end. The file `lao2` is an ASCII file containing, among other things, the singular values themselves, and is ignored for our purposes. `T.S` and `T.SINV` are also binary files and contain some header information first. The header consists of two long integers representing the number of singular values and vectors written out (which may be different from k) and the

number of terms t. In `T.S`, this header is followed by the product $T.S$, which is a txk matrix, in column major order, with the same qualifications as for `lav2`. `T.SINV` has the product $T.S^{-1}$, which is also txk, following the header, as in `T.S`.

In the SVD group, there is another program named `mags.c`, not supplied as a part of SVDPACKC. This program computes the magnitudes of the vectors corresponding to the documents in the reference library, and writes the number of documents, followed by the magnitudes, one per line, into an output file specified by the user (which it asks for). In computing the magnitudes, `mags` uses the `T.S` and `lav2` files mentioned above.[5]

3.4 The LSI-Based Analysis and Neural Net Input Generation Group

The program `convert`, mentioned in Section 3.2, apart from converting the sparse matrix from one format to another, also generates four other *executable* files: `neural`, `LSIcat`, `input` and `LSI`. The executable logic for these files, to be explained later in this section, is specified, respectively, in `neural.c`, `LSIcat.c`, `input.c` and `LSI.c`. The program `convert` creates temporary files, named, `%@+3.c`, `%@+4.c`, `%@+2.c` and `%@+1.c`, respectively, by adding to the these `.c` files some data definitions corresponding to the terms in the reference library of documents. (It leaves the original `.c` files unchanged.) It then generates the executables by compiling the temporary files and later deleting them. As can be expected, the data about the terms in the reference library is directly available in the original term-document matrix specified to `convert`, as indicated at the end of Section 3.2.

Now, we describe the programs `neural`, `LSIcat`, `input` and `LSI`, that are

[5]Another related program `magsraw.c`, which is currently not used, was also written, which computes the magnitudes of the document vectors straight from the original term-document matrix file (whose name it asks for), independent of any SVD.

generated by `convert`. The purpose of `neural` is to generate the training or test input for the neural network, into a file called `NEU.nna`[6]. from a sequence of input document files. These input document files should be those created by `aplexbasic` (see Section 3.1). The program asks for the numbers of the first and last file to be classified (for training or testing), and also for the desired number of "factors". This latter input value corresponds to the number of singular values (and vectors) used in computations, and is required for predictable control. This was necessitated by the fact that the SVD programs supplied in SVDPACKC do not give the user precise control on the number of singular values computed. As long as there are at least as many singular values (and vectors) available as the specified number of factors, just the right number of the more significant values are used from the file `T.SINV`, which is assumed to be available. This program also assumes the availability of a `categories` file[7], which is used to create the training output (or reference output for testing). The program also repeatedly generates and later deletes a temporary file named `%@+3`.

`LSIcat` can be used to classify a sequence of input document files using an LSI-based method, for comparison against the performance of the multi-sensor neural network. The LSI approach, as described in [Deerwester, et al., 90], was originally intended for document *retrieval*, rather than *classification*. We modified it to do classification as follows. First, the document from the reference library that matches an input document best is identified. Next, the category of the reference document is looked up in the `categories` file and is reported as the category for the input document in question.

[6]In this work, we used a neural network tool named NeuralWorks, trademarked by NeuralWare, Inc. This package requires the training and test file names to end in `.nna`.

[7]This file is to list the number of the category for each of the document files, in the format of the file number (ignored), followed by a tab, followed by the number of the category to which the file belongs, one pair per line. This list must be immediately preceded by a line starting with a hyphen.

The input document files for `LSIcat` should be those created by `aplexbasic` (see Section 3.1). As with `neural`, `LSIcat` also asks for the numbers of the first and last file to be classified (for training or testing), and also for the desired number of "factors". It uses the two files `T.S` and `lav2` in computing the best matching reference document. Further, the program asks for the name of the "document magnitudes" file, that must already have been created by the `mags` program (see Section 3.3). This program repeatedly generates and later deletes a temporary file named `%@+4`.

The purpose of `input` is to generate a single input vector for the neural network, from a single input document file, which, either is created by `aplexbasic` (see Section 3.1) or is simply some other kind of ASCII file. The program asks for the desired number of "factors" to be used from the `T.SINV` file, and also for the name of the input file. This program is currently not used much, and needs to be modified for incorporating the category information into the output file it creates, namely `SINGLE.OUT`. It also generates and later deletes a temporary file named `%@+2`.

The program `LSI` can be used to match a single input document file (presumably created by `aplexbasic` (see Section 3.1) or is simply some other kind of ASCII file) against all the documents in the reference library. It simply computes, using the LSI method in a straightforward manner, the cosine values of the angles between the input document vector and the vectors of all the reference library documents, and writes these values into a file name that it asks for. In this process, it also asks for the name of the "document magnitudes file" (as does `LSIcat`) and also for the desired number of "factors". It uses the right number of factors from the two files `T.S` and `lav2` in computing the cosine rankings. At the end it asks if another document file is to be matched. It generates and later deletes a temporary file named `%@+1`.

3.5 The Other Sensors Group

The creation of a very simple second sensor to the neural net, with the goal mainly being a proof of concept, works as follows. The program `gensensor2.c` can create a program that generates a second input vector (viewed as a second sensor) for the neural network. It assumes the existence of the file `sensor2.c`, which represents the executable portion of the second logical sensor for the neural net (to be described shortly). It asks for the number of categories, and also assumes the existence of as many *category profile* files as the number of categories. These files are supposed to contain a number of keywords[8] which constitute a profile for the respective category. These files are assumed to have the names `cat#.basic`, where # stands for the category number (refer to the beginning of Section 3). (The words in the profile files are stemmed and written to corresponding files, whose names start with the prefix `s` added to the beginning of the category profile file names.) It also creates a temporary file (`%@+5.c` by adding data corresponding to the category profiles to the logic of the `sensor2.c` file, compiles it to create the executable `sensor2` and then deletes the temporary file. It also uses another temporary file named `%@+6`.

The program `sensor2` asks for the starting and ending file numbers, which define the range of the input document file numbers. These files *must* have been created either by `aplex` or `singlex` (see Section 3.1). That is, the input files to `sensor2` must be files of stop-word-filtered, stemmed words, one per line. It creates an output file named NEU.S2, in which each line is simply a vector (corresponding to an input document), whose size equals the number of categories. The components of the vector represent what fraction of the terms in the document correspond to the words in the profile. The degree of match for the "general" category is negatively related to the best match for

[8]There must be at least two keywords for each category, except for the "general" category, which is supposed to just have any single word.

any of the other categories.

The program `neural3.c` creates a combined neural net input vector from `NEU.nna` and the `NEU.S2` files. Recall, from the preceding description of the `neural` program, that the `NEU.nna` file contains LSI-based input vector and training output (or test reference) values. It actually asks for the two input file names and the desired name for the combined file. The latter file is created as the output.

3.6 The Miscellaneous Group

This group contains several programs: `results.c` analyzes the neural net output and reports the percentage pf correct results; `rantest.c` creates several random sets of neural net training and test input pairs from a single given file of neural net input vectors; `histcat.c` gives a simple histogram of the distribution of files in each category, as indicated by the `categories` file; etc.

4 Results

A major stumbling block in this work was the manual categorization of documents for training and testing purposes. The neural network is of a substantial size, with more than one hundred input nodes and about ten output nodes. Such a network typically requires several thousand training inputs, and this requirement increases with the number of hidden units. Manual classification of thousands of news stories was extremely time consuming, even if it was trivial in a number of cases. Such data are not already available with the TIPSTER data set. We could manually categorize only 480 news stories, which is what we worked with. Consequently, the following results are far from conclusive. We believe that they do, however, point out that

the approach is promising, in spite of the fact that the neural net is very inadequately trained.

Of the 480 documents, the first 380 were used as the "reference library". That is the term-document matrix used in *all* LSI/SVD operations has 380 columns. The number of SVD "factors" used in this work was 112. The neural net inputs created by `neural`, were also based on this matrix. The neural net was a simple feedforward net with back propagation, and used the delta rule for learning and the tanh transfer function. It was tested in two configurations; one with 112 inputs (just based on LSI alone) and another with 112 LSI-based inputs *plus* another 10 inputs based on simple category profiles. Both configurations used 10 output units, one for each category. The neural nets that performed best for each configuration were chosen, which happened when the single sensor neural net (with just the 112 LSI-based inputs) has 9 hidden units, and the other, two sensor neural net (with 122 inputs) had 10 hidden units.

When LSI method was used by itself to perform classification on the 100 documents that are not part of the reference library, there was essentially just a single experiment. This was so because, generating the SVD for each new reference library was computationally very expensive even when several megabytes of main memory was used by the program. The results from the `LSIcat` program, which performed the LSI-based classification were somewhat surprising. Although the performance of LSI in classifying the known library documents was a perfect 100%, the percentage of correct results when the 100 new documents were used dropped to 54%.

We present the results of our neural network experiments in two tables. The percentages of correct results shown in the tables represent the peak performance that did not get any better with more training. For testing the neural nets, inputs were created for all the 480 documents, including a correct answer for each case. This "answer" was to be used either for training the

Data Set No.	No. of Iterations	Percent Correct with Test Data	Percent Correct with Training Data
1	48K	58	80.70
2	16K	72	77.67
3	64K	72	76.98
4	64K	62	77.21
5	32K	62	76.05
6	48K	62	76.98
7	80K	62	80.47
8	48K	58	78.14
9	64K	68	77.44
10	48K	60	78.14
11	48K	64	78.84
12	32K	68	76.05

Table 1: Results of Classification with the Single Sensor Neural Net

neural net or for comparison, in the case of a testing. The `rantest` program was then used to generate a dozen pairs of files from this set of 480 inputs. Each pair of files had a training file and a test file. Each of the dozen pairs were generated by a different random distribution of 430 inputs into the training file and the other 50 inputs into the test file. The same pairs of data sets were used to test both the single sensor neural net and the two sensor one.

Table 1 shows the performance of the 112-input neural net on each of the dozen different data sets. In each set, the test file was tested after the neural net was trained for anywhere between 16,000 and 64,000 iterations. On the test set, the correctness percentage ranged from a minimum of 58% to a maximum of 72% for the dozen sets. When the training set itself was

Data Set No.	No. of Iterations	Percent Correct with Test Data	Percent Correct with Training Data
1	48K	62	85.11
2	48K	70	84.42
3	32K	76	84.19
4	64K	64	83.72
5	32K	64	84.88
6	16K	66	83.26
7	32K	66	84.88
8	16K	62	84.19
9	32K	70	83.49
10	32K	58	83.02
11	16K	64	80.47
12	32K	72	83.02

Table 2: Results of Classification with the Two Sensor Neural Net

used as a data set, the performance was between 76.05% to 80.7% correct.

Table 2 shows the performance of the 122-input neural net on each of the dozen different data sets. Again, in each set, the test file was tested after the neural net was trained for anywhere between 16,000 and 64,000 iterations. On the test set, the correctness percentage ranged from a minimum of 58% to a maximum of 76% for the dozen sets. When the training set itself was used as a data set, the performance was between 80.47% to 85.11% correct. It may be seen that in the case of individual data sets, there was an improvement in performance with 10 out of the 12 data sets in the two sensor neural net, compared to the single sensor version, with marginal decrease of performance in the other two cases. But in one of these two cases (data set 2), the superficially better performance of the single sensor neural net decreased a

few percent with more training. These anomalies can perhaps be attributed to the simple-mindedness of the second sensor used (see Section 3.5).

We believe that one thing the results conclusively indicate is that the neural nets need more training. There is a clear improvement of classification results in the neural net approach compared to the LSI method by itself. And the two sensor version, with a very simple-minded second sensor, seems to do better in most cases than the single sensor version.

5 Future Work

The work is far from complete. Clearly the neural networks need more training, and more training requires more data. Automatic tools, as described in Section 3, are in place for generating data from AP news wire stories, but each of the stories needs to be manually categorized to complete the data. Once this is done, different possible neural network architectures need to be tried for best performance with each different set of input sensors.

Other important extension to the work here would be in the area of other input sensors. LSI is a simple, yet very powerful technique for information representation. However, it is limited by the vocabulary seen so far in the reference library.[9] One way to address this limitation is to add other sensors which are sensitive to newer words and other patterns in the input. The second sensor that was implemented attempted to do that using a very simple technique. The slight improvement of results even with such a simple addition is very encouraging, adding strength to our original hypothesis that a multi-sensor neural network approach would be very effective for information classification and retrieval.

[9] In our data, the vocabulary from the reference library was rather large - about 10,025 unique stemmed terms. This set, however, includes "terms" like 1, 100th, $2, 1974, etc., which comprise about 5-6% of the terms.

Natural language (NL) processing techniques hold some promise in this direction [Schank, et al., 81] [Lewis, et al., 89] [Smeaton, 92] [Lewis, 92]. The depth of analysis using established NL techniques perhaps needs to be controlled carefully to optimize performance. Deep NL analysis runs into the problem of scaling up well; sensors based on shallower NL processing a la [Schank, et al., 81] might provide just the right balance. Krovetz suggests an approach sensitive not just to words, but word senses (the classic issue of *polysemy*), which adds more discriminatory power, but requires somewhat deeper processing[Krovetz, 90]. The LSI approach appears to capture some aspects of *synonymy* to the extent that synonyms co-occur in related documents, but perhaps does not offer fine discrimination of the different senses of polysemous words.

The basic idea of the second sensor based on category profiles that was used in this work can be strengthened by various means. Instead of creating profiles manually, as done in our simple version, profiles could be automatically generated. One way to generate category profiles automatically is using LSI itself. The answer to the question how important is term i to document j (or how strongly associated are they) can be used appropriately, when the category of document j is already known. However, it is doubtful that this technique will add substantial new information to the neural net, that is not already contained in the LSI input. Another technique would be to use existing methods that suggest significant terms for predefined topics, e.g., [Zhou and Dapkus, 95].

The approach may also be adopted to perform related functions, such as document filtering and document rertrieval. The categories may be chosen to fit individual interest profiles, and new documents can be identified to fit particular profiles. Alternately, the neural net can be set up to accept two sets of inputs, one corresponding to a query (viewed as a "pseudo-document") and the other corresponding to a candidate document, with the output in-

dicating a degree of match. The net can be run with each document in the collection (from which retrieval is to be made) as a candidate. Those document satisfying predefined retrieval criteria (e.g., the top ten documents, all documents exceeding a threshold degree of match, etc.) may be retrieved.

The SVD programs are very memory intensive. In our experiments, they took several megabytes of main memory. At one point, the program ran out of main memory, and could only be run by reducing precision from `double` to `float`. This problem can easily become acute, even with single precision, as the size of the term-document matrix increases. As the number of documents in the reference library increases, so does the number of terms, and consequently, the memory problem becomes doubly more acute. The number of factors selected in the SVD also directly affects the memory requirement. So, when larger matrices are involved, the application may have to be moved to a supercomputing or parallel computing environment.

On the software engineering front, the components of the research prototype developed in this work need to be integrated together into a system that is easy to use, with an effective user interface.

Acknowledgements

The first author wishes to thank Reinhold Mann, Head, Intelligent Systems Section, for the opportunity and support to work on this project and the facilities provided. He also thanks the Oak Ridge Institute for Science and Education for the summer fellowship that made this work possible. Thanks are also due to Mike Berry of University of Tennessee, Knoxville and his students for distributing the SVDPACKC software, to Chuck Glover of the Intelligent Systems Section (ISS) at ORNL for help with the NeuralWorks package, and to Ron Lee of the ISS for help with general systems problems. Availability of some general lexical utilities from New Mexico State Univer-

sity's Consortium for Lexical Research is also acknowledged.

References

[Berry, et al., 93] Berry, M., T. Do, G. O'Brien, V. Krishna and S. Varadhan, SVDPACKC (Version 1.0) User's Guide, Technical Report CS-93-194, Department of Computer Science, University of Tennessee, Knoxville, April, 1993 (revised October, 1993).

[CACM, 92] Communications of the Association for Computing Machinery, Special Issue on Information Filtering, 35(12), December, 1992.

[Computer Journal, 92] The Computer Journal, Special Issue on Information Retrieval, the British Computer Society and Cambridge University Press, 35(3), June, 1992.

[Damashek, 95] Damashek, M., Gauging Similarity with n-Grams: Language-Independent Categorization of Text, *Science*, 267, pp. 843-848, 10 February, 1995.

[Deerwester, et al., 90] Deerwester, S., S. Dumais, G. Furnas, T. Landauer and R. Harshman, Indexing by Latent Semantic Analysis *Journal of the American Society for Information Science*, 41(6), pp. 391-407, 1990.

[IEEE Expert, 93] IEEE Expert, Special Issue on Using AI in Text-Based Information Retrieval, 8(2), April, 1993.

[Jacobs, 93] Jacobs, P., Using Statistical Methods to Improve Knowledge-Based News Categorization, *IEEE Expert*, 8(2), pp. 13-23, April, 1993.

[Krovetz, 90] Krovetz, R., Information Retrieval and Lexical Ambiguity, *Working Notes of AAAI Spring Symposium on Text-Based Information Systems*, pp. 70-72, 1990.

[Lewis, et al., 89] Lewis, D., B. Croft and N. Bhandaru, Language-Oriented Information Retrieval, *International Journal of Intelligent Systems*, 4, pp. 285-318, 1989.

[Lewis, 92] Lewis, D., An Evaluation of Phrasal and Clustered Representations on a Text Categorization Task, *SIGIR-92 Proceedings*, pp. 37-50, 1992.

[Porter, 80] Porter, M., An Algorithm for Suffix Stripping, *Program*, 14(3), pp. 130-137, July, 1980.

[Salton, 89] Salton, G., Automatic Text Processing, Addison-Wesley Publishing Company, Reading, MA, 1989.

[Salton, et al., 94] Salton, G., J. Allan, C. Buckley and A. Singhal, Automatic Analysis, Theme Generation and Summarization of Machine-Readable Texts, *Science*, 264, pp. 1421-1426, 3 June, 1994.

[Schank, et al., 81] SChank, R., J. Kolodner and G. DeJong, Conceptual Information Retrieval, *Information Retrieval Research*, Oddy, R., S. Robertson, C. vanRijsbergen and P. Williams (Eds.), Butterworths, Boston, pp. 94-116, 1981.

[Smeaton, 92] Smeaton, A., Progress in the Application of Natural Language Processing to Information Retrieval Tasks, *the Computer Journal*, 35(3), pp. 268-278, 1992.

[Turtle and Croft, 92] Turtle, H., and B. Croft, A Comparison of Text Retrieval Methods, *the Computer Journal*, 35(3), pp. 279-290, 1992.

[Uberbacher and Mural, 91] Uberbacher, E., and R. Mural, Locating Protein-Coding Regions in Human DNA Sequences by a Multiple Sensor-Neural Network Approach, *Proc. Natl. Acad. Sci. USA*, 88, pp. 11261-11265, 1991.

[van Rijsbergen, 92] van Rijsbergen, C., Probabilistic Retrieval Revisited, *the Computer Journal*, 35(3), pp. 291-298, 1992.

[Wilkinson and Hingston, 92] Wilkinson, R. and P. Hingston, Incorporating the Vector Space Model in a Neural Network used for Document Retrieval, *Library Hi Tech*, 10(1-2), pp. 69-75, 1992.

[Xu, et al., 94] Xu, Y., R. Mural, M. Shah and E. Uberbacher, Recognizing Exons in Genomic Sequence using GRAIL II, *Genetic Engineering, Principles and Methods*, Plenum Press, 15, June, 1994.

[Zhou and Dapkus, 95] Zhou, J. and P. Dapkus, Automatic Suggestion of Significant Terms for a Predefined Topic, *Third ACL Workshop on Very Large Corpora*, MIT, Cambridge, June 1995.

Multi-Sensor/Neural Net Intelligent Information Retrieval

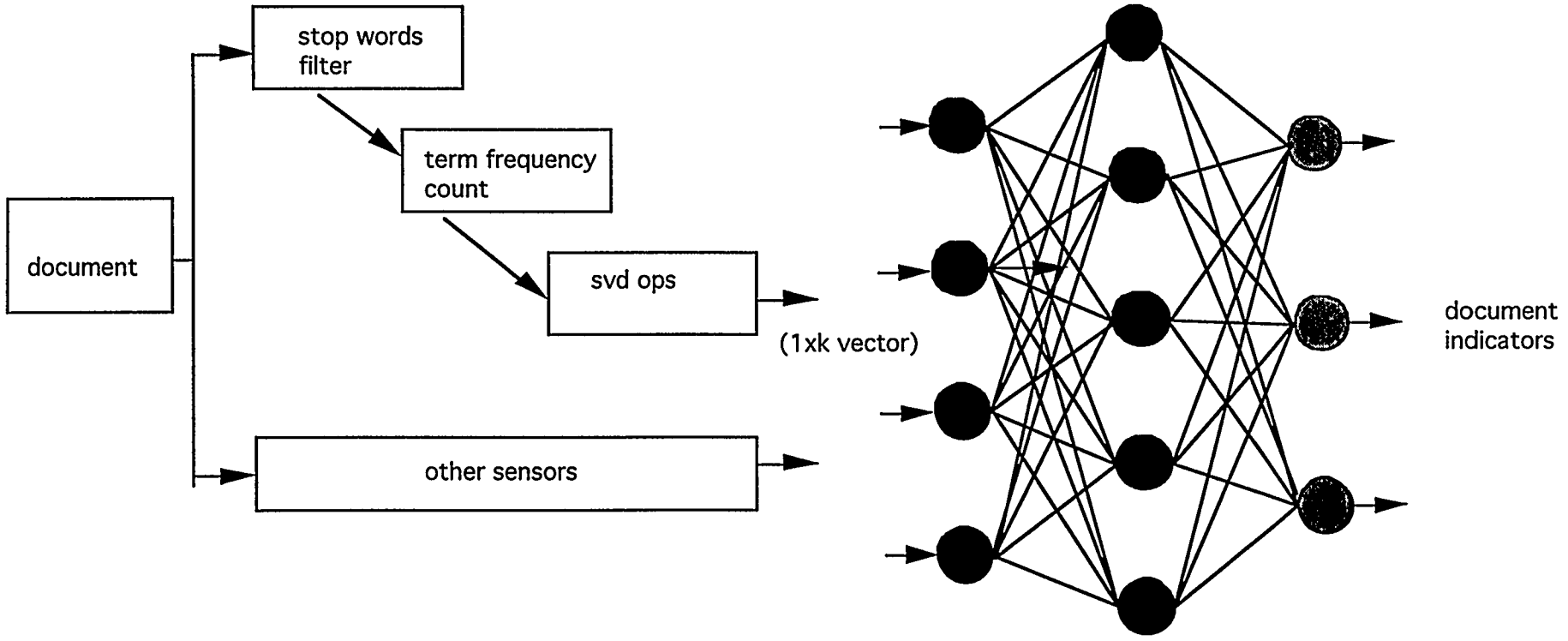

Figure 1. Schematic of multi-sensor/neural network approach to intelligent text retrieval and document classification.

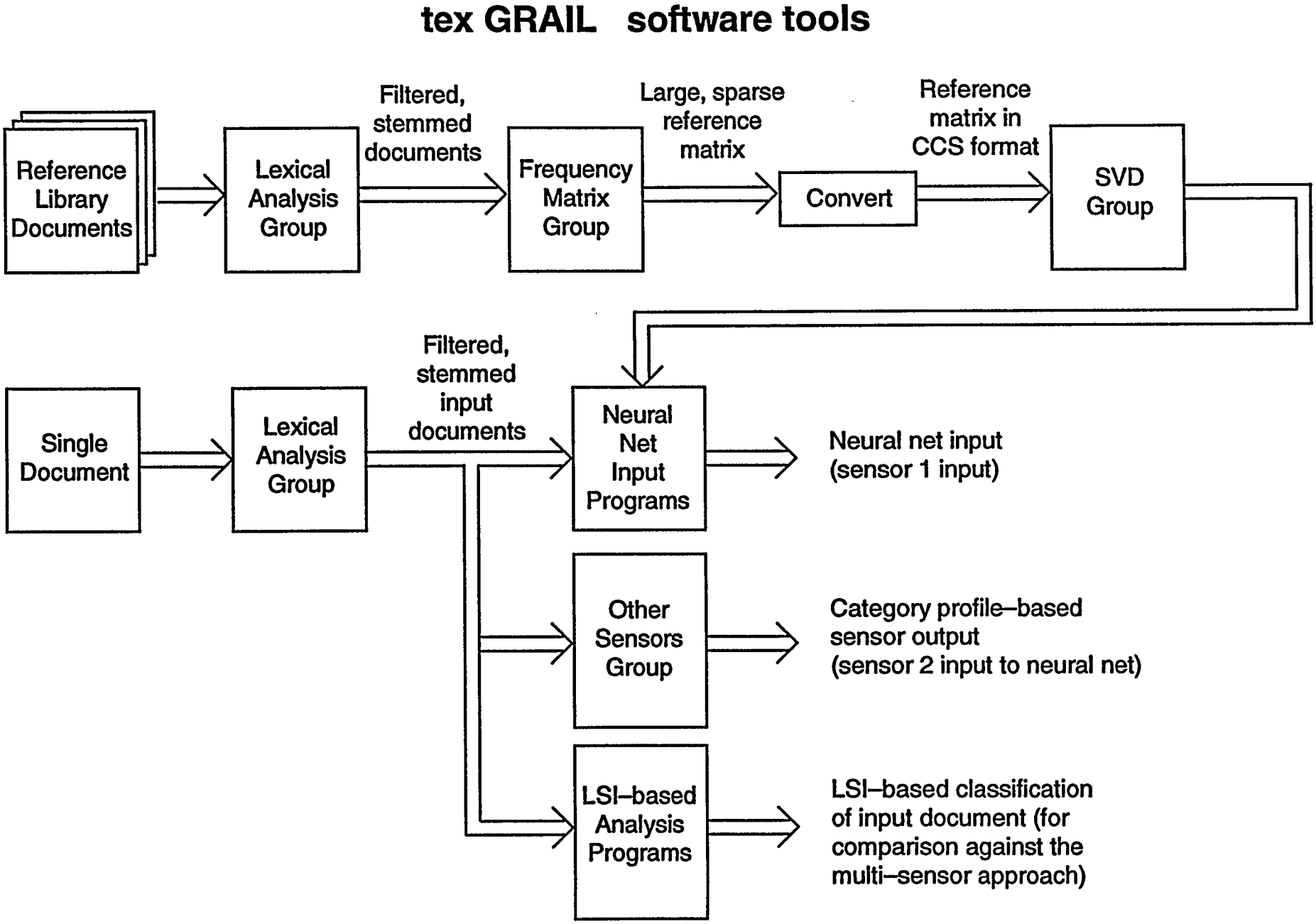

Figure 2. Schematic diagram of texGRAIL module interdependencies.

INTERNAL DISTRIBUTION

1. J. Barhen
2. V. Baylor
3. J. R. Einstein
4. C. W. Glover
5. W. C. Grimmell
6. X. Guan
7. H. E. Knee
8. R. W. Lee
9. S. Matis
10. S. McKenney
11-15. R. C. Mann
16. E. M. Oblow
17. C. E. Oliver
18. S. Petrov
19. V. Protopopescu
20. N. S. V. Rao
21. M. Shah
22. R. F. Sincovec
23. CSMD Reports Office
24-25. Laboratory Records, ORNL-RC
26. Document Reference Section
27. Central Research Library
28. ORNL Patent Office

EXTERNAL DISTRIBUTION

29-30. Office of Scientific and Technical Information, P. O. Box 62, Oak Ridge, TN 37831.

31. Office of Assistant Manager for Energy Research and Development, Oak Ridge Operations, U.S. Department of Energy, P. O. Box 2008, Oak Ridge, TN 37831.

32-36. Professor V. R. Dasigi, Department of Computer Science and Information Technology, Sacred Heart University, Fairfield, CT 06432-1000.

37. Professor R. K. Srihari, Center for Excellence for Document Analysis and Recognition, SUNY-University at Buffalo, UB Commons, 520 Lee Entrance-Suite 202, Buffalo, NY 14228-2567.

38. J. Blair, JBX Technologies, 25 Moore Road, Wayland, MA 01778.

39. D. Harman, NIST, Building 225, Room A216, Gaithersburg, MD 20899.

40. J. Prague, ORD, P. O. Box 4132, Washington, DC 20505.

41. J. Baker, ORD, P. O. Box 4132, Washington, DC 20505.

42. B. Glatz, NPIC, P. O. Box 70967 Southwest Station, Washington, DC 20024-0967.

43. M. Berry, Department of Computer Science, 107 Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301.