Towards Fast, Effective Nearest Neighbor Text Classification

David D. Lewis, S. Muthukrishnan, Rafail Ostrovsky, Martin Strauss

November 28, 2002

1 Introduction

Great progress has been made in recent years in increasing the accuracy of systems for information filtering, alerting, categorization, and other text classification tasks. A variety of statistical techniques, using both labeled and unlabeled data, have been used to produce highly effective clasifiers without the need for manual rule-building. These techniques have been extensively validated in, for instance, the TREC evaluations and through widespread experiments on publicly availble text categorization data sets.

A disadvantage of the more effective approaches to text classification is that they are also more expensive (in terms of memory usage and compute time) than hand-built rules or the simple classifiers produced by older techniques.

Nearest neighbor text classifiers (NNTCs) are a good example. As with other supervised learning approaches to text classification, an NNTC is provided with a set of *training documents*, i.e. documents which have been manually judged for whether or not they are relevant to one or more topics of interest. However, in contrast to most other supervised learning approaches, NNTCs do not then analyze the training documents and replace them with a derived classifier. Instead they simply store the training documents.

A NNTC classifies a new document (a *test document*) by first finding those stored training documents that are most similar to (i.e. least distance from) the test document, i.e. the *nearest neighbors*.[1] The NNTC then decides to which topics the test document is relevant by using the nearest neighbors. Some NNTC methods simply count how many neighbors belong to each topic, while others take into account the degree of similarity of each neighbor.

Some advantages of NNTCs are:

- *Simplicity:* NNTCs do not require complex, iterative optimization algorithms.
- *Comprehensibility:* The set of neighbors of a test instance can be examined to see why the instance was assigned to particular classes.
- *Nonlinearity:* In contrast to the more popular linear classification approaches, NNTCs can handle complex and subtle distinctions between topics, given sufficient training data. They also make different mistakes than linear classifiers, potentially making them a good contributor to classifier fusion approaches.
- *Topic-level parallelism:* Finding the neighbors of a test document only needs to be done once, regardless of how many topics are of interest. Further, by using sparse data structures, classification time can be made dependent on the number of topics that the neighbors belong to, rather than on the total number of topics.

The last point suggests NNTCs may be an efficient and effective approach to simultaneously monitoring a text stream for a large number of topics of interest. This would require that neighbor-finding be made much faster, however, since past research on NNTCs has emphasized effectiveness and simplicity rather than efficiency.

In this report we discuss our approaches, which apply representation, compression, and matching techniques, for speeding up NNTC. We begin by briefly reviewing the state of the art. We then describe our

[1] We will talk interchangably of high similarities and low distances in this document.

approaches, which draw upon and combine two distinct lines of recent research: sparsity-oriented techniques for speeding up text retrieval in search engines, and theoretical advances in the use of randomized methods for distance finding. We then briefly review our work-to-date, planned work for the rest of this year, and directions we would explore in future project years.

2 Efficiency Approaches for NNTC

In this section we review both existing and our proposed approaches for speeding up NNTC.

2.1 Current State of the Art

Documents in text classification (and text retrieval) systems are stored as vectors of numeric weights corresponding to indexing terms (words, phrases, etc.). An inverted index stores, for each indexing term in a collection, a list of *<ID, weight>* pairs. Each ID is of a document that contains the term, and the weight is the weight of that term in the document.

Inverted indexes were developed for ranked retrieval search engines. In ranking documents for a query, only the inverted lists for query terms need be consulted, a great efficiency advantage over scanning all document vectors. As each inverted list is read, the stored weights are multiplied by query weights and used to update score accumulators for the appropriate documents. The final scores are then sorted to find the highest scoring documents.

The same technique is used in NNTC [16], with the test document playing the role of the query. Aggressive feature selection (i.e. discarding words that have little impact on scores, such as linguistic function words) is often used as well, since this further reduces the number of inverted lists both stored and consulted.

2.2 New Techniques 1: NN-Specific Inverted Index Heuristics

Published research on NNTC has not emphasized efficiency, and so has not gone beyond basic uses of inverted indexes and feature selection. In contrast, the demands of large-scale search engines have led to considerable research on speeding up inverted index-based ranked retrieval [3, 12, 14, 15]. The major algorithmic techniques are:

- Reading inverted lists, or portions of inverted lists, in order of their impact on scores. The computation is then terminated when the system can show that the document scores or ranks will not be substantially changed by examining more list entries.
- Quantizing term weights to allow compact storage and integer arithmetic.
- Creating accumulators only for documents with high scores.
- Using data structures that allow finding the top-scoring documents without sorting all accumulators.

We refer to these techniques as heuristics because they (with the possible exception of the last) lead to documents getting somewhat different scores than they would if no optimization was done. The degree of approximation must be traded off against the speed of the resulting system.

We will apply these techniques, modifying them to take into account these differences between NNTC and search:

- The "queries" in NNTC are full documents rather than a few words. This both allows, and requires, more pruning in the number and size of inverted lists accessed.
- Search engine design focuses on limiting disk accesses to the inverted index, which is usually assumed to be too large to keep fully in memory. For NNTC, the inverted index both can (given the typically small number of manually labeled documents) and must (to achieve high throughput) be kept in memory. Efficiency concerns therefore move up the memory hierarchy, to issues such as reducing cache misses.

- Some NNTC algorithms, which we call *weighted NNTC algorithms*, take into account the actual numeric similarities of the near neighbors, as well as the identity of the neighbors. We are studying the degree to which weighted NNTC accuracy is degraded when similarities are computed only approximately.

2.3 New Techniques 2: Randomized Projections to Boolean Vectors for Approximate Distance Preservation

Inverted index heuristics are a conservative, engineering approach to speeding up NNTC. Our other approach, in contrast, constitutes a bigger risk and a bigger opportunity, in that it leverages very new work in mathematics and theoretical computer science.

The core idea is that distances among vectors in a high dimensional space are preserved by certain projections to much lower dimensional spaces, and that appropriate projections can be found by randomized polynomial-time algorithms [6]. However, making randomized projection methods for text data competitive with other techniques is a nontrivial challenge. Research on randomized projections has emphasized handling vectors with high dimensionality (i.e. that are very long), but has mostly ignored the *sparsity* of the vectors. Document vectors do indeed have a large number of dimensions, say $d \approx 100,000$. However, document vectors are sparse: any given document's weight on most dimensions will be 0, since most words do not occur in any given document. Inverted index techniques take advantage of this sparsity, and thus avoid operations that scale with the number of dimensions.

For randomized projection algorithms to be superior to inverted index-based methods, we will need to achieve their mathematical approximation gaurantees, while implementing them in a fashion that takes advantage of sparsity.

We have investigated several approaches to date, some of which mathematical analysis has eventually shown to be inappropriate for textual data. Others we are currently implementing for experimental evaluation.

Currently, our most promising approach is based on methods for approximating Hamming distances among boolean (0/1) vectors. At a high level, the approach is as follows:

1. Map each training document vector to a boolean vector in a moderate to very high dimensional boolean space (the "large cube"). The mapping is such that Hamming distances (number of disagreeing bits) between transformed vectors approximate Euclidean distances between the original document vectors.

2. Map the large cube boolean vectors to compressed boolean vectors (which we call *signatures*) in each of several low dimensional boolean spaces (the "small cubes"). Hamming distances in each small cube accurately approximate a particular range of Hamming distances from the large cube.

3. Store the signatures in an index structure that allows rapid lookup of signatures with small Hamming distance to a query signature.

4. To find the nearest neighbors of a test document, we map it to one or more signatures, and use the index structure to find the nearest training document signatures.

The remainder of the section describes these steps in more detail.

2.3.1 Step 1: Mapping to Vectors in the Large Cube

While some representations (including novel ones we are investigating [2]) produce boolean document vectors, the most effective traditional representations produce real-valued document vectors. Our first step, therefore, is to map these real-valued document vectors to a boolean representation (large cube vectors) that preserves distances between vectors.

The difficulty is that the large cube vectors must either be of higher dimensionality than the original document vectors, or non-sparse, or both, since boolean values carry less information than real ones. Computation of large cube vectors must therefore be done very carefully to maintain efficiency. Ideally only those portions of the large cube vectors need to map to signatures should be generated.

We are investigating two approaches to producing the large cube representation.

Method 1: Via a Real-Valued Intermediate Representation Method 1 has two steps:

- **Step 1a.** We take the dot product of each original document vector with $d_{\mathrm{LR}} \approx 150$ real-valued, unit 2-norm, random vectors. This randomly projects the document vector to a dense real-valued vector of length d_{LR}.
- **Step 1b.** The real-valued vectors of length d_{LR} are then projected to large cube boolean vectors of length $d_{\mathrm{LR}} \times d_{\mathrm{THRESH}} \approx 15,000$ by comparing each real coordinate with a set of $d_{\mathrm{THRESH}} \approx 100$ data-dependent thresholds.

Kushilevitz, Ostrovsky, and Rabani proposed this method and showed that Hamming distances between the large cube boolean vectors with high probability approximate Euclidean distances between the corresponding original document vectors [8]. The values suggested for d_{LR} and d_{THRESH} above are based on one of our data sets, but in general depend on the number of document vectors and the range of distances at which near neighbors are found. In practice, they would be tuned, along with the thresholds, to a sample of inter-document distances from the training data.

Both steps can be made more efficient by "lazy evaluation", i.e. generating coordinates of vectors only as needed for a particular computation:

- Step 1a uses random vectors which are both dense and high dimensional (the same length, d, as the original document vectors). Having $d_{\mathrm{LR}} \approx 150$ such vectors in memory at the same time is impractical. Fortunately, this can be avoided by generating each such vector using a pseudorandom function with inputs based on the vector ID and dimension ID. A given coordinate can then be regenerated at any time by using the same pseudorandom function with the same inputs. (This is a slight oversimplification, but gives the basic idea of the technique.)
- The value of a coordinate of a large cube boolean vector is found by comparing a coordinate of the d_{LR}-dimensional real-valued vector with a particular threshold. This test is extremely efficient, so the low-dimensional real-valued vectors (plus the recorded thresholds) are actually a better representation of the large cube boolean vector than an explicit representation would be.

Method 2: Direct Mapping from Document Vectors to Large Cube Boolean Vectors by Discrete Sampling Method 2 differs from Method 1 in several ways:

- A randomized mapping is made directly from document vectors to large cube boolean vectors, without the intermediate step of a randomized projection to low dimensional real-valued vectors.
- The large cube vectors are of extremely high dimension, but are also very sparse.

The method assumes that the original document vectors have a Euclidean norm of 1.0. This is true of many text representations, but where necessary the restriction can easily be relaxed.

Each coordinate of the original document vector is mapped to a block of d_{BLKSZ} boolean coordinates in the large cube representation. Each bit of the large cube vector is independently set to 1 with a probability equal to the value of the corresponding coordinate of the document vector. Hamming distances in this representation with high probability approximate Euclidean distances between the original document vectors.

The technique can be viewed as representing each document vector coordinate by a sample drawn from a Bernoulli (0/1, i.e. 1-of-2) distribution with parameter equal to the coordinate. The product of two coordinates is then represented as the intersection of the samples, divided by sample size (the blocksize). Representing the sample as a boolean vector means that blocksize minus the Hamming distance gives the size of the intersection. Kushilevitz, Ostrovsky, and Rabani [7, 8, 11] showed a related technique could be used to preserve L_1 distances between real-valued vectors with high probability, for use in nearest neighbor lookup. Cohen & Lewis [4, 5] showed a similar approach, using a sample drawn from a multinomial (i.e. 1-of-d) distribution with parameters equal to the entire vector of coordinates, to approximate dot products with high probability, also in the context of nearest neighbor lookup.

While our analysis is still underway, it is likely that d_{BLKSZ} will need to be 1000 bits or more, so that large cube vectors will be of dimension 100 million or more. This makes large cube vectors hopelessly impractical to generate explicitly, but fortunately this can be avoided:

- The large cube representation is sparse at the block level. That is, if a coordinate of the document vector is 0, the entire block of corresponding bits in the large cube vector consists of 0's. Thus, as with Method 1, computation of signatures can be done in time proportional to the number of nonzero entries in the original document vectors. Indeed, in contrast to Method 1, there is not even a preprocessing step proportional to the length of the original vectors, and the dimensionality of the document vectors is not required to be known or constant. This has advantages for online filtering.
- A specified bit in a non-zero large cube vector block can be lazily generated by looking up the corresponding coordinate of the original document vector and comparing it to the output of a pseudorandom number generator. It may also be possible to take advantage of the internal sparsity of most non-zero blocks.

2.3.2 Step 2. Mapping to Signatures

Step 1 provides a way to map document vectors into large cube vectors while preserving distances. Step 2 then maps each document's large cube vector to several lower dimensional ($d_{\mathrm{SIG}} \approx 15 - 17$ bits) boolean vectors, called *signatures*.

The goal here is that Hamming distances between signatures approximate Hamming distances between corresponding large cube vectors. This is not possible in general, of course, since there is a substantial loss of information in going from the large cube to the small cube. However, Kushilevitz, Ostrovsky, and Rabani [8, 11] showed that multiple small cubes can be used, each preserving a different range of Hamming distances from the large cube.

Each signature is produced by computing exclusive ORs (XORs) of randomly chosen bits from a large cube vector. Different ranges of distances can be preserved by varying the density of bits chosen. As described above, we can generate bits from the large cube vectors lazily, so that only those bits necessary for computing the signatures are produced. Further, for Method 2, sparsity lets us take advantage of the fact that bits equal to 0 leave the XOR unchanged.

2.3.3 Steps 3 and 4. Indexing and Searching Signatures

One way to use the collections of signatures would be as follows:

1. Produce for a test document its signature in each small cube.
2. For each small cube, find the signatures closest to the corresponding test document signature.
3. Find the cube where the nearest neighbors fall into the range of distances accurately approximated by that cube, and use those as the neighbors.

Kushilevitz, Ostrovsky, and Rabani [8, 11] describe the approach in more detail and subtlety. However, while computing Hamming distances between signatures is much faster than computing distances between original document vectors, it is not fast enough for the above approach, which scales with the number of training documents, to be practical.

It therefore seems necessary to index the signatures, and have a good strategy for searching those indexes. We are currently analyzing several possible index structures:

- *Spiral search*[8, 11]: The Hamming neighbors of the query signature are generated one at a time and we check if they are present in the small cube. A hash table is a possible implementation.
- *Precomputed neighbors indexed by hashed signatures*[8, 11]: For each small cube, we precompute *all* $2^{d_{\mathrm{SIG}}}$ possible signatures that a test document could have. For each possibility, we precompute the list of near neighbor signatures for training documents, using the maximum neighborhood size desired for our NNTC system. At query time, we do a binary search among the small cubes, mapping the test document to a signature and looking up the stored list, until the small cube is found where the distances to the neighbors fall within the range accurately captured by that cube. (It is likely that many of the lists where distances fall outside of the accurately represented range can be discarded, but we are still working out the details of this.)

- *Trie on transposed signatures*[9]: We first transpose the set of signatures for each training document, producing a new set of vectors with one bit for each range of distances. We build a trie based on the first transposed vector from each training document, another using the second transposed vector for each training document, and so on. At query time, we map the query vector to its signatures lazily, while searching the tries in parallel to find sufficiently small subtrees containing potential neighbors. The best of these can be found by explicitly computing actual distances between the query vectors.

2.4 New Techniques 3: Randomized Mappings to Real-Valued Vectors

We have also investigated algorithms based on randomized mappings to real-valued vectors, as discussed in detail elsewhere[13]. So far we have not produced a method that promises any speedup over inverted index methods, but we continue to study this.

3 Schedule

NNTC algorithms are both promising and less well understood than other text classification approaches. We therefore have a phased approach to developing and testing these algorithms.

The phases are:

1. **Phase 1.** Develop NNTC algorithm.

   Evaluation Criteria: Effectiveness on typical scenarios under some, possibly manual, parameter setting.

   Discussion: We have begun by implementing published NNTC algorithms. As work progresses we will develop new algorithms to meet the needs of the message filtering scenario (see Sections 3.0.2 and 3.0.3).

2. **Phase 2.** Automatic tuning for NNTC algorithm.

   Evaluation Criteria: Effectiveness across range of topics and training documents.

   Discussion: Past NNTC research has often relied on manual tuning of parameters such as neighborhood size and thresholds to particular data sets. We will develop methods that, to the extent possible, automatically set these parameters based on the training data, while still incorporating prior knowledge when available.

3. **Phase 3.** Fast, *batch* implementation of algorithm.

   Evaluation Criteria: Efficiency of batch training. Efficiency of classification

   Discussion: This is the phase on which our design efforts to date have been focused. By a batch implementation, we mean one that assumes that all training data is received before any test documents are classified. Indexing approaches are more easily understood, implemented, tested (and possibly ruled out) in a batch context.

4. **Phase 4.** Fast, *online* implementation of algorithm.

   Evaluation Criteria: Efficiency of online training. Efficiency of classification.

   Discussion: An online implementation is one that can incorporate new training documents at any time, and classifies each test document based on all training documents received up to this point. To put it another way, if manual feedback is available after a test document is classified, it then becomes a training document.[2] This is a better fit to message filtering than is batch training.

[2]The machine learning literature often uses the term "online learning algorithm". We prefer to distinguish batch vs. online *implementations* of arbitrary learning algorithms. Any "batch" learning algorithm can be made online by retraining from scratch as each training example is received, while any "online" learning algorithm can be run with a single batch of training data before any test data is seen. The real issue is efficiency in training computation, classification computation, and memory usage.

While conceptually each algorithm will go through the four phases above, we obviously do not start from scratch with each algorithm. For instance, the efficiency work which has been our focus so far is applicable to Phase 3 of almost any NNTC algorithm (though parameter settings may vary for particular algorithms). The same is largely true of Phase 4 techniques as well, though algorithm-specific work will be more frequent here.

In the remainder of this section we discuss our work to date, work scheduled for the remainder of the first year of the project, and our plans beyond that.

3.0.1 Work to Date

We began by modifying the Lemur text retrieval system from Carnegie Mellon University [10] to support online filtering. We constructed several data sets, benchmarked linear classification methods, and participated in the TREC evaluation. This work is discussed in detail elsewhere [1].

We then added a state-of-the-art (i.e. inverted index based) NNTC implementation to our Lemur-based system, and are currently tuning and benchmarking it for various NNTC algorithms. We have so far achieved a document throughput of 85,000 test documents/hour against a training set of 80,000 documents judged with respect to 100 topics. (Thus the classification rate is 8.5 million classifications/hour.) We do not have effectiveness results on our full test set yet, but effectiveness is clearly superior to that of the linear models used in our TREC work.

In parallel with the above, we pursued mathematical and design work on indexing methods. We describe our results to date in Sections 2.2 to 2.4.

3.0.2 Work in Remainder of Year 1

Our work in the remainder of Year 1 focuses on these areas:

- **Adapting Search Engine Heuristics to NNTC:** We will implement and test combinations of the speedup heuristics described in Section 2.2.
- **Online Implementation of Inverted Index Updates:** In message filtering scenarios, training documents will become available intermittently. For inverted index techniques to be useful, they must be able to easily incorporate new training documents as they become available. Again, published research on search engines will be helpful.
- **Batch Implementation of Randomized Projection Methods:** We will complete our analysis of the randomized boolean projection methods, and implement and test the best of them. We will compare their efficiency to that of inverted index-based heuristics.

3.0.3 Work in Year 2 and Beyond

Work in Year 2 and beyond would focus on both adapting our methods more closely to the operational constraints of message filtering, as well as extending them to tasks of detecting novel and alarming patterns of documents.

- **Online Implementation of Randomized Projection Methods:** We will adaptthe best of the randomized project methods to allow online incorporation of training data. This will require mathematical, algorithmic, and statistical advances, since the projection methods currently known assume all data is available in advance.
- **Supporting Partially Labeled Data:** In message filtering scenarios, not only do training documents become available incrementally, but each training document will typically have been judged for relevance with respect to only one or a few of the topics. This complicates the definition of the "neighborhood" of a test document, and in the worst case would require that all training documents be used in scoring each test document. The most promising solution to this problem is adapting the technique of *pseudofeedback*, where known training judgments are used to fill in missing ones. Pseudofeedback is an ad hoc and poorly understood technique, and naive implementations of it for NNTC would be

inefficient. However, because of the close connection between pseudofeedback and the concept of neighborhoods, we believe that bringing more mathematical sophistication to bear should pay substantial benefits.

- **Weighted Labeling of Training Examples:** One innovation in our TREC-2002 work was differential weighting of training documents, particularly pseudolabeled training. Supporting example weighting in NNTC would aid not just pseudofeedback, but also the ability to use training data from sources of variable quality. As with partially labeled data, weighted examples complicate the notion of "neighborhood", since the same example will often have different weights for different topics.

- **Applying NNTC Speedup Techniques to Linear Classifiers:** Some of the efficiency techniques we develop may be applicable to a broader range of classifiers than just NNTCs. Instead of finding training documents with high similarities to a test document, we could find the linear classifiers with the highest dot products with the test document. The big difference is that each linear classifier has its own threshold, so yet again the definition of neighborhood must change.

- **Novelty and Pattern Detection:** A major goal of work in Year 2 and beyond would be detection of unusual documents and suspicious clusters of documents. Neighorhood methods are clearly applicable, but limiting false alarms would require the ability to incorporate analyst knowledge of clues to interesting deviations. One possibility is to use this knowledge to weight the dimensions used in computing neighbors (much as in standard term weighting methods). A major difference is that unlabeled as well as labeled documents would need to be incorporated into an evolving representation, making it necessary to take secondary storage access into account.

References

[1] Andrei Anghelescu, David D. Lewis, Vladimir Menkov, and Paul Kantor. Training a Rocchio classifier for adaptive filtering. In E. M. Voorhees and D. K. Harman, editors, *The Eleventh Text REtrieval Conference, TREC 2002*, Gaithersburg, MD, 2003. Department of Commerce, National Institute of Standards and Technology. To appear.

[2] Endre Boros and David J. Neu. Feature selection and batch filtering. In E. M. Voorhees and D. K. Harman, editors, *The Eleventh Text REtrieval Conference, TREC 2002*, Gaithersburg, MD, 2003. Department of Commerce, National Institute of Standards and Technology. To appear.

[3] Chris Buckley and Alan F. Lewit. Optimization of inverted vector searches. In *Eighth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97–110, 1985.

[4] Edith Cohen and David D. Lewis. Approximating matrix multiplication for pattern recognition tasks. In *Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 682–691, 1997.

[5] Edith Cohen and David D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30:211–252, 1999.

[6] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[7] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In ACM, editor, *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 614–623, New York, NY, USA, 1998. ACM Press.

[8] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[9] S. Muthukrishnan. A simple, coarse vector nearest neighbors algorithm. DIMACS MMS Project Report, November 2002.

[10] Paul Ogilvie and Jamie Callan. Experiments using the Lemur toolkit. In E. M. Voorhees and D. K. Harman, editors, *The Tenth Text REtrieval Conference, TREC 2001*, pages 103–108, Gaithersburg, MD, 2002. Department of Commerce, National Institute of Standards and Technology.

[11] Rafail Ostrovsky and Yuval Rabani. Method for determining approximate Hamming distance and approximate nearest neighbors of a query. United States Patent 6,226,640, May 1 2001.

[12] Falk Scholer, Hugh E. Williams, John Yiannis, and Justin Zobel. Compression of inverted indexes for fast query evaluation. In Micheline Beaulieu, Ricardo Baeza-Yates, Sung Hyon Myaeng, and Kalervo J:arvelin, editors, *Proceedings of SIGIR 2002: The Twenty-Fifth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229, New York, NY, 2002. Association for Computing Machinery.

[13] Martin Strauss. Distances from sketches. DIMACS MMS Project Report, November 2002.

[14] Howard Turtle and James Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995.

[15] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Von Nostrand Reinhold, New York, 1994.

[16] Yiming Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In W. Bruce Croft and C. J. van Rijsbergen, editors, *SIGIR 94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 13–22, London, 1994. Springer-Verlag.