

Indexing for Fast Categorisation

Vaughan R. Shanks

Hugh E. Williams

Adam Cannane

School of Computer Science and Information Technology,
RMIT University, GPO Box 2476V, Melbourne 3001.

{shanks, hugh, cannane}@cs.rmit.edu.au

Abstract

Automatic categorisation is an important technique for the management of large document collections. Categorisation can be used to store or locate documents that satisfy an information need when the need cannot be expressed as a concise list of query terms. Inverted indexes are used in all query-based retrieval systems to allow efficient query processing. In this paper, we propose the application of inverted indexes to categorisation with the aim of developing a fast, scalable, and accurate approach. Specifically, we propose successful variants of inverted indexing to reduce index size: first, quantisation of term-category weights; second, compression of the quantised weights; and, last, storing only those weights that significantly impact the categorisation process. We show that our techniques permits fast, accurate categorisation: index size is reduced by orders of magnitude compared to conventional inverted indexing and the accuracy of categorisation is preserved.

Keywords: Document management, categorisation, compression, inverted indexing, efficiency.

1 Introduction

The volume of information stored and transmitted by electronic means is continually increasing. Documents transmitted via the web, as well as email, electronic publications, and voice transmissions must be carefully managed if users are to meet their information needs. Query-based information retrieval systems — such as Google¹ — allow users to locate documents if an information need can be expressed concisely as a list of terms. Automatic categorisation is an alternative technique that allows a user to specify example documents that match an information need. Moreover, automatic categorisation can be used to store documents in classes, and then structured browsing can be used to explore a collection.

Several web portals have incorporated categories as an alternative to query-based retrieval. For example, DMOZ² offers a manually-maintained category hierarchy, which relies on thousands of volunteers to locate and add web sites. While manual categorisation is highly effective, it does not scale to large volumes of data, and human categorisation is a subjective process. Automatic categorisation using computers can be reasonably accurate, and is better suited to applications such as categorising large volumes of email and voice traffic. Augmentation of existing human-maintained web hierarchies with automatically categorised documents is an important

potential application for automatic categorisation.

A recent survey describes how documents can be accurately categorised using text categorisation techniques [26]. Indeed, recent work in the field of text categorisation has focused on adaptive filtering [35] and online categorisation algorithms [4, 6]. However, the efficiency of text categorisation systems — their speed and space requirements — is rarely considered.

Several open-source text categorisation systems are designed to work accurately using efficient machine-learning algorithms [14, 16, 33]. While these implementations are effective and efficient when applied to machine-learning problems with few categories, they assume that all of the category models can be concurrently stored in main-memory. While this assumption is acceptable for small collections, a disk-based solution is required for a text categorisation system to scale to larger problems such as web categorisation.

Inverted indexes are used for efficient evaluation of queries in search engines [34]. For query-based information retrieval, an index consists of two major components: first, an in-memory list of terms — usually words — that occur in the collection to be searched; and, second, for each term, a postings list of documents that contain those terms. The query evaluation process then follows several steps: first, the query terms are looked-up in the in-memory list of terms; second, for those query terms that occur in the collection, the postings lists are retrieved from disk; third, the lists are processed and similarity of documents to the query is calculated; and, last, matching documents (or summaries) are retrieved and displayed to the user.

For efficient query processing, the postings lists in an inverted index are carefully compressed using integer compression schemes [17, 24, 32, 34]. Compression offers three major efficiency benefits:

1. Compression makes more efficient use of communication bandwidth, as more data can be transferred per time interval than when the data is uncompressed. Indeed, for the fast decompression schemes used for compressing postings lists, the total time cost to transfer compressed data and subsequently decompress it is much less than the cost of transferring uncompressed data
2. More data can be stored in memory than when uncompressed, which reduces disk accesses, that is, caching of lists is improved
3. A compressed index requires less storage space

Compression is therefore essential to improve the performance of retrieval systems.

Copyright ©2003, Australian Computer Society, Inc. This paper appeared at the Twenty-Sixth Australasian Computer Science Conference (ACSC2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, Vol. 16. Michael Oudshoorn, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

¹See: <http://www.google.com/>

²See: <http://dmoz.org/>

In this paper, we investigate how inverted indexes can be adapted for fast categorisation. Similarly to regular inverted indexes, we store terms that occur in the collection in a main-memory search structure. On disk, we store a variation of postings lists that contain the weight of the term in each of the categories. We propose three techniques that are specific for efficient storage of these postings lists:

1. Quantisation of weights, that is, we approximate floating point weights using small ranges of integer values; this is an adaptation of a recent approach to efficient quantisation in regular inverted indexes [2]
2. Removing weights that do not (or have minimal) impact on the accuracy of categorisation
3. Compressing category-based postings lists

Our experiments with a Rocchio [21] categoriser show that our techniques permit fast, accurate categorisation. Relative to an unoptimised approach — where the index stores all floating point weights — our scheme can categorise documents an order of magnitude faster while maintaining accuracy. Our techniques can be extended to perform multi-category assignment with the addition of a thresholding algorithm to achieve the same improvements in efficiency.

2 Background

In this section, we present a background of categorisation techniques and focus on the application of the Rocchio relevance feedback method. In addition, we introduce inverted indexes, index compression, and recent work in efficient index processing.

2.1 Document Categorisation

Categorisation is often required to store, manage, and retrieve documents. In general, automatic document categorisation consists of a training stage and a categorisation stage. In the training stage, a categoriser learns using example documents from a set of pre-defined categories. During online categorisation, the categoriser uses the generalisations learned from the training stage to label each of a set of unknown documents with zero or more pre-defined labels.

For the purposes of efficient information retrieval, it is convenient to treat a document as an unordered collection of words, or stems of words [26]. These words are the *features* of a document. Other feature extraction schemes have been proposed, such as the extraction of phrases, tagged words, or sequences of individual characters. Although the indexing schemes proposed in this paper are applicable to any kind of unordered features, we consider only words, which we refer to as *terms*.

In text retrieval systems, the vector space model [23] can be used to represent each document in a collection by a vector of weights, where each weight in the vector corresponds to the importance of a term in that document. Similarly, in categorisation, a category can be generalised as a vector of weights by a *linear categoriser* [15]. A category weight vector represents the features of that category, that is, it is a summary of the vectors for all example documents in that category. As we discuss in the next section, the Rocchio method [21] is one popular approach to training a linear categoriser; other examples of linear categorisers include the Perceptron [25], Winnow [7], and Exponentiated Gradient [15]. Linear categorisers

have been shown elsewhere to be efficient in training and online categorisation [27].

Figure 1 shows a simple example of term-category weights generated from a small training set. Each document in the training set is labelled with one of four categories: “food”, “computers”, “bicycles”, or “cars”. Using a learning technique — that we discuss later — term-category weights have been computed based on the occurrence of terms in documents in the training set, with the objective that each term-category weight represent the strength of association between the term and the concept represented by the category. For example, the terms “chip” and “eat” have high weights in the category “food”, indicating a strong association between the category and the terms. In contrast, “chip” and “eat” have low weights in the category “bicycles”.

Consider now the categorisation of an unlabelled document that consists of the phrase “the silicon chip”. A simple approach is to sum the weights of the three terms in the uncategorised document for each of the four categories to determine a score for each category. Ranking these scores in descending order gives “computers” with a score of 2.21, then “food” with 1.21, then “cars” and “bicycles” with equal scores of 0.53. Assuming only one label is to be assigned to each unlabelled document, we can now assign the highest ranked category, that is, we assign the label “computers” to the new document.

In this paper, we focus on the case where each document is assigned to exactly one category, a task often referred to as *routing*. In a routing system, the similarity of an unknown document is computed for each of the pre-defined categories, then the document is labelled with the highest similarity category, regardless of its score. Such an approach can be extended to perform multi-category assignments by the addition of a thresholding algorithm [26]; our techniques for efficient categorisation work without modification for multi-category assignment.

2.1.1 The Rocchio Method

The Rocchio method was first proposed as a method to provide relevance feedback in query-based information retrieval [21]. However, it is also a weight-learning algorithm commonly used in text categorisation. Although more accurate text categorisation techniques exist, the Rocchio method is simple, robust, flexible, and reasonably accurate. In this section, we introduce the Rocchio method and discuss our specific implementation, including our choice of weighting measure.

To train a categoriser using the Rocchio method, the j th term of the feature vector w for a category C is updated by each example document feature vector v_i as follows:

$$w_j' = \alpha w_j + \frac{\beta}{|C|} \sum_{i \in C} v_{ij} - \frac{\gamma}{n - |C|} \sum_{i \notin C} v_{ij}$$

where n is the total number of training examples, C is the set of positive training examples, i is a training document, and α , β , and γ are constants. In our experiments, we use constants of $\beta = 16$ and $\gamma = 4$, the same as those reported by Lewis et al. [15]. The α constant is unnecessary since the initial vector w has all terms zeroed. After training the categoriser, we set all negative term weights to zero [13, 15], and normalise the weight of each category to a unit vector; normalisation of the category vectors ensures that training categories with fewer examples are not dom-

	CATEGORIES			
	<i>food</i>	<i>computers</i>	<i>bicycles</i>	<i>cars</i>
<i>brakes</i>	0.00	0.00	0.71	0.71
<i>chip</i>	0.71	0.71	0.03	0.03
<i>drive</i>	0.00	0.71	0.00	0.71
<i>eat</i>	1.00	0.10	0.05	0.13
<i>pedals</i>	0.00	0.00	1.00	0.00
<i>motor</i>	0.00	0.00	0.00	1.00
<i>silicon</i>	0.00	1.00	0.00	0.00
<i>the</i>	0.50	0.50	0.50	0.50

Figure 1: A term-category weight matrix for a simple collection.

inated by larger categories. Weights are calculated using a ranking measure as discussed next.

The well-known *tf.idf* metric [22, 34] is the fundamental basis of most vector-based ranking measures used in text retrieval techniques. In this measure, the importance of a term in a particular document increases with the frequency of occurrence in that document, and the inverse of the frequency of occurrence in the collection as a whole. In addition, a weight function which lowers term importance in longer documents has been shown to improve ranking by preventing domination by longer documents [28].

The Okapi BM25 weight formula is a state-of-the-art vector-based ranking measure [20] that combines document frequency, inverse collection frequency, and document length. Our Rocchio categoriser uses a measure similar to BM25 — which in unreported experiments we have found is more accurate than *tf.idf* — and weights terms using the following formulation:

$$w = \frac{tf}{0.5 + 1.5 \times (dl/dl_{avg} + tf)} \times \frac{\ln((N + 0.5)/df)}{\ln(N + 1)}$$

where dl is the length of the document, dl_{avg} is the mean document length, tf is the number of times the term occurs in the document, df is the number of documents in which the term appears in the entire collection, and N is the number of documents in the collection. In cases where $df > \frac{N}{2}$, (resulting in the second term of the equation becoming negative) we set this to zero.

After training, for online categorisation, the document vectors of unlabelled documents are normalised so that cosine similarities [34] can be computed between the document and each of the pre-defined categories. The unlabelled document is then labelled with the highest similarity category:

$$\text{category label} = \arg \max_i \left(\sum_{j=0}^t w_j c_{ij} \right)$$

where w_j is the weight of the j th term in the document term vector, t is the number of terms in the collection, and c_{ij} is the weight of the j th term in the weight vector of category i .

2.2 Inverted Indexing

Inverted indexes are used for fast query evaluation in all practical query-based text retrieval systems. An inverted index consists of two main components [34]:

1. A *vocabulary* or term dictionary that contains the set of terms that occur in the collection; in many cases, terms are pre-processed or selected terms removed, but this outside the scope of our discussion here

2. For each term in the vocabulary, a *postings list* or *inverted list* that contains information concerning the location of the term within the collection

During the query evaluation process, the vocabulary is held in main-memory, while the postings lists are on disk.

Following the notation of Zobel and Moffat [36], postings lists have the following form. For each term t there is an inverted or postings list that contains posting tuples of the form $\langle f_{d,t}, d \rangle$, where $f_{d,t}$ is the frequency f of term t in document d . One posting is present in the inverted list for each document that contains the term t . Lists containing postings are sufficient to support the well-known ranked and Boolean query paradigms.

Consider an example postings list for the term “Richmond”:

$$\langle 2, 12 \rangle \langle 1, 27 \rangle \langle 4, 31 \rangle$$

This list indicates that the term occurs twice in document 12, once in document 27, and four times in document 31. Document numbers d are usually ordinally numbered and, as we discuss later, an auxiliary structure is used to resolve document numbers to physical disk locations of documents. To support other query modes — such as phrase or proximity queries — lists contain additional information such as word offsets within each document; we do not discuss the structure of more complex postings lists further here.

The query evaluation process using an inverted index proceeds as follows. First, for each term that appears in the query, the term is located in the vocabulary. Second, for each query term that does occur in the vocabulary, the postings list is retrieved from disk. Third, the postings lists are processed, and statistics *accumulated* concerning the similarity between each collection document and the query; for example, a similarity measure such as the Okapi BM25 discussed in the previous section may be used. Fourth, highly-weighted document accumulators are determined — usually using a heap data structure — and the physical locations on disk of the matching documents determined using an auxiliary *mapping table*. Last, the documents are retrieved from disk, and either the complete document or summaries shown to the user.

The document numbers in inverted lists are typically ordered by increasing document number d so that the difference between document numbers can be stored, rather than the raw value. This improves the compressibility of lists. However, other organisations of lists have been recently investigated, with the aim of permitting only partial processing of lists in response to queries while ensuring that the postings that are processed are those that contribute significantly to the overall results. For example, in frequency-sorted indexes [18, 19] the postings are ordered by $f_{d,t}$. In impact-ordered indexes the postings are ordered by decreasing ranking weight [1, 2]. However, regardless of the overall list organisation, compression of postings lists using integer coding techniques has been shown to be essential for fast query evaluation [24, 32, 34].

3 Inverted Indexing for Fast Categorisation

In this section, we propose the application of inverted indexing to categorisation. Specifically, we investigate how Rocchio-based categorisation can be supported using an inverted index and we propose novel techniques for efficient evaluation. In practice, these techniques are applicable to all term-based categorisers; thus, for example, the other linear categorisation

techniques — such as Winnow — that we described in the previous section could be supported using our structures.

Categorisation has several different characteristics to query-based text retrieval. First, when using the Rocchio method, all terms have a weight for all categories; this is in contrast to query-based retrieval, where each document does not contain all terms in the collection. Second, for each term, each category has a weight that is a floating point number; in contrast, integer document numbers and frequencies are required for query-based retrieval. Third, there are typically far fewer categories in a collection than there are documents. Last, a training process to derive category-term weight tuples is required in text categorisation. These characteristics necessitate careful application of indexing techniques to categorisation tasks.

Given the above characteristics, an intuitive application of inverted indexing is to:

1. Store an in-memory vocabulary of terms that occur in the collection; the techniques used are identical to those of query-based retrieval
2. Store a postings list for each term, where each posting is of the form $\langle c, w_{c,t} \rangle$, where c is an ordinal category number and $w_{c,t}$ is the weight w of term t in category c

For example, consider again the four categories “food”, “computers”, “bicycles”, or “cars”, and the term “eat” from Figure 1. Using our approach to storing postings, the list for “eat” would be:

$\langle 1, 1.00 \rangle \langle 2, 0.10 \rangle \langle 3, 0.05 \rangle \langle 4, 0.13 \rangle$

This indicates that category 1 has a weight of 1.00, category 2 has a weight of 0.10, category 3 has 0.05, and category 4 has 0.13.

By following this approach, all inverted lists are of the same logical length, since each term has a weight for each category. Because of this, the category number c need not be explicitly stored for each weight, assuming the weights are stored in ordinal category number order. In practice, therefore, the above list may be stored as:

$\langle 1.00 \rangle \langle 0.10 \rangle \langle 0.05 \rangle \langle 0.13 \rangle$

Omission of explicit category numbers improves the compressibility of the lists.

We identified earlier in this section that a key difference between categorisation and query-based indexes is the requirement to store floating point numbers in the former. This presents a difficult problem: assuming weights are represented using a standard 32-bit floating point representation, the size of the inverted index for m categories and n terms is $4 \times m \times n$ bytes. For example, if a categoriser were trained using the entire OHSUMED set of journal abstracts [12] — which contains 14,626 categories¹ and a vocabulary of 156,511 terms — an index of almost 9 gigabytes would be required, that is, the index is roughly 24 times larger than the collection. This is a marked contrast to the size of an index for a query-based retrieval system where — when the index stores both document numbers and word offsets — the index is only 10%–20% the size of the collection when stored using integer compression schemes [24]. In the next section, we discuss practical techniques to reduce the index size.

¹These topics are from the NIH *Medical Subject Headers* (MeSH).

3.1 Optimised Inverted Indexes

Compression techniques for storing integers in inverted lists have been investigated in detail elsewhere [17, 24, 32, 34]. However, these techniques cannot be directly applied to storing floating point numbers. One possible approach, therefore, is to treat each floating point number as two integer components, that is, a mantissa and exponent, and to then compress these integers using an integer compression scheme. In preliminary work on fast categorisation, Shanks and Williams [27] followed this approach, and stored each component using the non-parameterised Elias delta codes [8, 32, 34].

However, this approach to compression is unsatisfactory. Storing two integers for each floating point value is inefficient, and it is unclear whether the degree of precision represented by each floating point value is necessary. Indeed, in recent work by Anh and Moffat [2] and separate work by Franz [9], it was shown that storing integer approximations of weights in query-based retrieval techniques improves both the accuracy and efficiency of query evaluation.

We therefore investigated the application of a quantisation technique using a similar approach to that of Anh and Moffat [2]. As discussed in Section 2.1.1, our category weight vectors are normalised, that is, all category weights have values between zero and one. It is therefore possible to divide this interval into a fixed number of equally-sized *bins* — sub-intervals of equal range — and to represent each floating point weight by its integer bin number. For example, if the interval is partitioned into 256 bins, bin 1 represents the range 0.0000 to 0.0039, bin 2 the range 0.0040 to 0.0078, and so on. If the number of bins is a power of 2, the number of bits required to encode each bin value is constant. Thus, for example, by dividing the interval into 256 bins, each bin is represented by an 8-bit value. As the number of bins representing the interval decreases, the size of the index also decreases, but the approximation of the original weight becomes less accurate. We would therefore expect that there is a tradeoff between categorisation speed and accuracy, and we explore this later in Section 5.

Continuing our example from Figure 1, and assuming 256 bins, our postings list for “eat” is now:

$\langle 255 \rangle \langle 25 \rangle \langle 13 \rangle \langle 33 \rangle$

This indicates that the value for the category “eat” is in the range 0.9961 to 1.0000, the value for “computers” in the range 0.0977 to 0.1015, and so on. During categorisation, quantised values are translated to an approximation of their original floating point value; the integer approximations can also be used directly in similarity calculations, but we have not yet explored this in practice.

As we show later in Section 5, a 312 Mb collection of newswire documents with 1,830 categories requires an index of 1,675 Mb when 32-bit floating point values are stored, and around 419 Mb when our initial quantisation approach is applied. Not surprisingly, our results show that use of this technique is around four times faster than processing of the raw floating point values. However, as we discuss next, further improvements are possible.

Figure 2 shows the frequency of occurrence of non-zero bin values when all Reuters collection category weights are partitioned into 1,000 bins; we discuss the properties of this collection later in Section 4 but this distribution of weights is typical of the collections we have used. While not shown, around 85% of the partitioned weights are zeros and, of the remaining

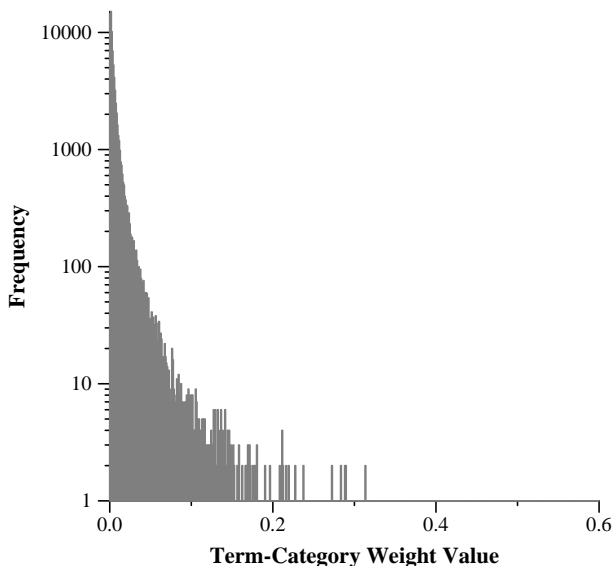


Figure 2: A histogram of category weights for the Reuters collection index, showing the distribution of non-zero term-category weight values partitioned into 1,000 bins. Non-zero weights comprise only around 15% of the weights.

weights, less than 0.4% have values greater than 0.1. The highest weight is 0.5603. Therefore, compression schemes that are appropriate to small integers are likely to be effective in reducing the index size further.

Choices of compression scheme for our bin-based index include the non-parameterised Elias delta and gamma codes discussed earlier, parameterised Golomb-Rice codes [10, 32, 34], and a variable-byte scheme [32]. Of the Elias codes, we would expect the gamma code to be most effective for this application, since they are at least as compact as the delta code for integers smaller than 15. Another approach is to determine the frequency of occurrence of each bin value, and to emit minimum-redundancy codes [29] for each bin; we have not explored this in practice.

We show later in Section 5 that the Elias gamma codes can reduce bin index sizes by up to a factor of almost 8, and improve document categorisation speed by around a factor of 1.3. We have not experimented with Golomb-Rice codes, but we would expect a further small improvement if a suitable compression parameter is chosen; it is unclear how this parameter might be determined. Variable-byte codes — which have been shown to be the fastest available scheme for query-based retrieval [24] — are impractical for bin-based categorisation indexes, since the smallest value is stored in eight bits and the vast majority of values are small.

3.2 Bin Pruning

As we showed in the previous section, the vast majority of weights are zero, and most non-zero weights are less than 0.1. Therefore, an intuitive approach to reduce index size further — and improve processing speed — is to omit zero weights from the index. Moreover, it is likely that low weight values, that is, those in low-numbered bins may also be able to be omitted without significantly affecting accuracy. We call this approach *bin pruning*. This approach is similar to the idea of partial list processing in query-based retrieval [17, 19] where only a fraction of each postings list is processed in response to a query, and similar to

the approach of omitting low-weight quantised values in impact-ordered indexes [2].

The drawback of bin pruning is that ordinal category numbers must be included in the index, since weights may no longer be from consecutive categories. Overall, index size is reduced only if the extra space required to store category numbers is less than the space saved by deleting small weights. In addition, the category vectors must be re-normalised to unit length following the deletions.

In our bin pruning approach, we omit the first N lowest weight bins. For example, if we have 256 bins, and delete the first 3 bins, we remove all weights from the index that have quantised values of 0, 1, or 2. After deleting weights from the first N bins, we subtract $N - 1$ from the remaining values, and store the remaining values using Elias gamma codes. In addition — as for document numbers in document-ordered inverted indexes for query-based retrieval — rather than storing category numbers, only the differences between category numbers are stored. We report results with our bin-based indexes, and bin pruned indexes in Section 5.

4 Test Collections

Four collections of documents were used in the experiments described in this paper. A summary of statistics for these collections is shown in Table 1.

A modified version of the Reuters-21578 collection [5] was used to measure the accuracy of our categorisers². Several different methods have been proposed to divide this collection into training and test data, and we follow the *Mod-Apte* split [3]. We further restricted the collection to include only those documents that could be used effectively for a routing task. To form the routing set, we ranked the categories by training set frequency, and chose the top 20 categories that had at least one test set document not assigned to any of the other 19 categories currently in the top 20. Next, we selected from the training and test sets only those documents that were assigned to exactly one of the 20 categories. The collection resulting from this process has 7,257 training documents and 2,532 test documents; the unrestricted Reuters-21578 collection has 9,603 training documents and 3,299 test documents.

To examine how our categorisers scale to large numbers of categories, we also trained our categorisers using Associated Press (AP) newswire data from TREC disks 1 and 2. TREC is an ongoing collaborative initiative to improve the effectiveness of text retrieval techniques [11]. From this collection, we extracted the contents of BYLINE tags to obtain 1,830 different author names, which we then used as categories. Documents which did not have BYLINE tags were discarded. The resulting training set is approximately 328Mb and consists of 86,139 documents. While it unclear as to the practical benefit of assigning documents to author categories, this experiment does allow examination of the performance of the categoriser with a large training set and a larger number of categories.

Articles from the Wall Street Journal (WSJ) from TREC disks 1 and 2 were assembled as a test set to evaluate the efficiency of the categoriser on a large data set. This test set is around 500 Mb in size and consists of 173,252 documents. During the categorisation process, the category labels assigned to doc-

²The Reuters-21578 collection is available from <http://www.daviddlewis.com/resources/testcollections/reuters21578/>

Collection	Use	Size (Mb)	Distinct Terms	Documents	Categories
Reuters	TRAIN	6.60	25,766	7,257	20
Reuters	TEST	2.00	—	2,532	20
AP	TRAIN	312.64	97,293	86,139	1,830
WSJ	TEST	503.48	—	173,252	—

Table 1: A summary of the collections used. The Reuters collections are used to measure accuracy, while the AP and WSJ collections are to measure efficiency.

uments in the WSJ test set by a categoriser trained on the Reuters or AP training sets may not have any significance, as the WSJ documents are not organised into the Reuters or AP categories. However, the elapsed time gives an indication of the efficiency of the categoriser on large volumes of newswire text.

Special consideration had to be applied when comparing the Reuters-trained categoriser to the AP-trained categoriser, on the WSJ test set. The Reuters-trained categoriser has a vocabulary of 25,766 terms, while the AP-trained categoriser has 239,809 terms, hence when processing the WSJ set, the AP-trained categoriser will generally find more of the distinct document terms in its term dictionary. For this reason, efficiency testing on the WSJ data was conducted using only the 20 highest-weighted terms from each document. In this way, with the exception of very short documents, exactly 20 terms will be extracted from each document, regardless of the size of training dictionary used. Using this approach, it is possible to directly compare the elapsed times of execution of categorisers trained with the 20- and 1,830-category training sets, to demonstrate the scalability of our implementation.

5 Results

The accuracy of our categorisation techniques are reported using the measures macroaveraged recall, macroaveraged precision, and macroaveraged F_1 . Recall and precision are well-known text retrieval measures [34] that respectively quantify the ability of a retrieval technique to retrieve all relevant answers and the ability of a technique to return only relevant answers.

In the context of text categorisation, recall for a category is the ratio of correct assignments made to a category by the categoriser divided by the total number of possible correct assignments. Precision for a category is defined as the ratio of correct assignments made by the categoriser divided by the total assignments made by the categoriser to the category. The F_1 measure [30] is a combination of recall and precision:

$$F_1 = \frac{2rp}{r+p}$$

We use F_1 as our primary measure of accuracy as it provides a convenient summary of recall and precision, and allows two techniques to be compared directly. The macroaveraged recall, precision, and F_1 is the mean of the individual category scores for these measures. We examine the relative performance of our techniques using these measures and do not aim to set new benchmarks for categorisation accuracy; in a production system, choice and tuning of the categorisation technique and similarity measure may further improve accuracy.

We measure the speed of our categoriser as elapsed time. All results reported are averaged over five runs, and the disk and CPU cache are flushed before each run. All timing experiments were carried out on

Training Collection	Encoding	Index Size (Mb)	Time per Document (ms)
Reuters	Float Raw	2.06	1.385
	8-bit Raw	0.59	1.362
	8-bit Delta	0.18	1.379
	8-bit Gamma	0.18	1.351
AP	Float Raw	1675.00	56.871
	8-bit Raw	419.43	13.760
	8-bit Delta	54.75	9.997
	8-bit Gamma	54.52	7.561

Table 2: A comparison of index sizes and elapsed time for several different encoding schemes. Only the 20 highest-weighted terms from each document in the WSJ collection were processed.

Encoding	μ -Macro Recall	μ -Macro Precision	μ -Macro F_1
Float Raw	0.8909	0.7680	0.7984
8-bit	0.8902	0.7654	0.7946

Table 3: A comparison of the categorisation accuracy of floating point and 8-bit quantisation. The Reuters test collection is used with the Reuters training data.

identical machines with dual 866 MHz Intel Pentium III processors, 256 Mb of main-memory, SCSI disk drives, running the Linux operating system (kernel version 2.4.2-2smp). The machines were under light-load, that is, no other significant processes were running on the machines during timing. Again, in our experiments, we examine the relative performance of our techniques, and do not aim to set a benchmark for categorisation performance; our research prototype would be faster if implemented in production.

Table 2 shows a comparison of categorisation performance for our initial indexing approaches that do not include bin pruning. The “Float Raw” scheme stores weights as uncompressed 32-bit values, the “8-bit Raw” scheme stores quantised weights in 256 bins as uncompressed values, and the “8-bit Delta” and “8-bit Gamma” schemes store the 256 bin values using the respective Elias compression schemes. The elapsed times shown are average per-document categorisation times for the around 500 Mb WSJ collection, using only the 20 highest-weighted terms per document.

As expected, the size of the indexes are unreasonably large when 32-bit floating point values are stored or 8-bit quantised values are stored without compression: the floating point index for the AP collection is more than five times the size of the collection, and the 8-bit quantised index around 1.3 times. Moreover, the document categorisation speed is slow for the larger AP collection, where long lists of 1,830 uncompressed floating point values are retrieved for each term. In contrast, 8-bit quantised indexes stored with Elias codes are reasonable in size, while also offering reasonable document processing times; the Elias gamma coded AP collection index is 18% of the size

Bit Precision	μ -Macro	μ -Macro	μ -Macro	Index Size (Mb)		Time per document (ms)	
	Recall	Precision	F_1	Reuters	AP	Reuters	AP
8	0.8902	0.7654	0.7946	0.18	54.52	1.328	7.278
7	0.8876	0.7534	0.7861	0.18	53.97	1.333	7.307
6	0.8933	0.7441	0.7771	0.17	53.60	1.333	7.319
5	0.9011	0.7273	0.7659	0.17	53.40	1.333	7.284
4	0.8954	0.7028	0.7461	0.17	53.32	1.368	7.013
3	0.8133	0.5902	0.6341	0.17	53.29	1.356	7.036
2	0.4856	0.2922	0.3290	0.17	53.29	1.345	7.486
1	0.0500	0.0019	0.0037	0.17	53.29	1.351	7.561

Table 4: A comparison of index sizes, accuracy, and elapsed time for different bit precisions, gamma encoded. Only the 20 highest-weighted terms from each document in the WSJ collection were processed. The bit precision determines the number of bins used in quantisation.

Bins Pruned	μ -Macro	μ -Macro	μ -Macro	Index Size (Mb)		Time per document (ms)	
	Recall	Precision	F_1	Reuters	AP	Reuters	AP
0	0.8902	0.7654	0.7946	0.36	211.41	1.449	16.109
1	0.8919	0.7690	0.7983	0.15	5.07	1.328	3.302
2	0.8921	0.7664	0.7973	0.14	4.02	1.304	3.030
3	0.8867	0.7526	0.7849	0.13	3.37	1.304	2.857
4	0.8870	0.7515	0.7849	0.13	2.84	1.299	2.730
5	0.8854	0.7463	0.7797	0.13	2.49	1.304	2.649

Table 5: A comparison of index sizes, accuracy, and elapsed time for different numbers of bins pruned. Weights are stored in 256 bins and are Elias gamma coded after subtracting the number of pruned bins from each value. Differences between category numbers are stored gamma encoded. Only the 20 highest-weighted terms from each document in the WSJ collection were processed.

of the collection — which is comparable with that of query-based retrieval index — and the processing time for the 1,830 category collection is only around 5 times slower than that for the 20 category collection.

Importantly, as Table 3 shows, there was almost no difference in accuracy between the floating point and 8-bit bin schemes. When categorising the Reuters test set into the Reuters categories, we found that the “Float Raw” scheme F_1 was 79.8% compared to the “8-bit” bin F_1 of 79.5%. In practice, such a difference is unlikely to be statistically significant [31]. We therefore experimented only with 256 and less bins in the remainder of our experiments.

Table 4 shows the effects of different numbers of bins on the accuracy and efficiency of categorisation. Weights were quantised to the bit precision shown and then Elias gamma coded. For example, a bit precision of 4 results in 2^4 or 16 bins of equal size over the interval 0 to 1. The 8-bit quantisation results are reproduced from Table 2. With decreasing numbers of bins, macroaveraged F_1 falls from 79.5% for 256 bins to around 74.6% for 16 bins. Interestingly, while precision falls by around 6% over the same interval, recall remains roughly constant. Perhaps not surprisingly, index size and evaluation time do not fall significantly with increasing bin granularity: this is because most values that are stored are small values and there is no saving in storing these compressed with Elias gamma codes when the number of bins is reduced. Overall, however, 8-bit precision may be favoured since precision is often seen as more important than recall in retrieval tasks, and there is only a small saving in both index size and query evaluation time for smaller bin values.

Table 5 shows the effect of our bin pruning strategy. We show only the effect of bin pruning on our 256 bin quantised index, however similar trends are observed when pruning after more coarse grain quantisation. For illustration only, we include as the first

row the cost of including explicit category numbers in the index without pruning any bins; this scheme would not be used in practice. After 256 bin quantisation, only 44 of the 25,766 Reuters terms have no zero weights at all. Therefore, almost all lists have potential to be reduced in size if category numbers are stored efficiently. Indeed, overall, the average number of non-zero weights in postings lists is only 3.019, suggesting that the tradeoff will favour further compression. As an aside, the terms that have weights for all categories appear, in general, to be words with low information content in the Reuters collection such as “national”, “reuter”, “future”, “total”, “annual”, and “increase”; we plan to investigate optimisations for these lists as future work.

Overall, the effect of our bin pruning strategy is fast, accurate categorisation. Pruning two bins marginally improves the accuracy of categorising the Reuters test data into the Reuters categories from an F_1 of 79.5% without pruning to an F_1 of 79.7% with two bins omitted. Moreover, the index for the AP collection is reduced from 54.52 Mb without pruning (but without category numbers) to 4.02 Mb with two bins omitted; the index size, therefore, for the 1,830 category index is only 1.3% of the size of the collection and fits entirely into main-memory. Most importantly, categorisation is fast: AP documents are processed in 2.857 milliseconds on average, a saving of around 60% over the original 8-bit index without pruning. In addition, the time cost of categorising into a 1,830 category index is only around 2.3 times more than into a 20 category index.

6 Conclusion

Categorisation is an important technique for managing large document collections. In this paper, we have proposed indexing techniques for efficient and accurate categorisation of large collections, and shown

experimentally that these techniques are effective in practice. Our techniques can be applied directly to areas as diverse as web document categorisation, network packet snooping, newswire filtering, and defence applications.

In addition to showing that inverted indexing can be successfully applied to categorisation tasks, we have proposed quantisation techniques that significantly reduce the size of the index without affecting the accuracy of categorisation. We have also shown that a compressed index can be further reduced in size by discarding small-valued term-category weights that do not affect the categorisation process. Overall, we have found that a quantised index where weight values are stored in 256 bins and then compressed reduces index size by a factor of around 20, while removing only the first three bins further reduces the size by a factor of 18. Most importantly, these reductions in index size have no effect on accuracy, and permit faster and more scalable online categorisation of large document collections.

We plan further investigation of categorisation index design, including alternative weight selection schemes — such as a fixed number of category weights per term — as well as design of indexes for adaptive categorisation tasks, such as topic tracking. Additionally, we plan to validate these results on other collections with different language properties, such as email or journal abstracts. With modification, the methods described in this paper could also be applied to more accurate categorisers; we are currently investigating application of these methods to Support Vector Machines [14]. In addition, our schemes can be applied to problems where there are large vocabularies or indexed terms are compound words; we plan to investigate this as future work.

7 Acknowledgments

This work was supported by the Australian Research Council and the Australian Defence Science and Technology Organisation, and the Australian Defence Signals Directorate. We thank the referees for their comments.

References

- [1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In W.B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 35–42, New Orleans, LA, 2001.
- [2] V. N. Anh and A. Moffat. Impact transformation: effective and efficient web retrieval. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 3–10, 2002.
- [3] C. Apte, F. Damerou, and S. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.
- [4] K. M. A. Chai, H. L. Chieu, and H. T. Ng. Bayesian online classifiers for text classification and filtering. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 97–104, 2002.
- [5] W.W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. In H. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 298–306, Zurich, Switzerland, August 1996.
- [6] K. Crammer and Y. Singer. A new family of online algorithms for category ranking. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 151–158, 2002.
- [7] I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. In C. Cardie and R. Weischedel, editors, *Proc. Conference on Empirical Methods in NLP*, pages 55–63, Providence, RI, 1997.
- [8] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21(2):194–203, March 1975.
- [9] M. Franz and J. S. McCarley. How many bits are needed to store term frequencies? In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 377–378, 2002.
- [10] S.W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, IT-12(3):399–401, July 1966.
- [11] D. Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing & Management*, 31(3):271–289, 1995.
- [12] W. Hersh, C. Buckley, T. J. Leone, and D. Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In W. B. Croft and C. J. van Rijsbergen, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 192–201, Dublin, Ireland, 1994.
- [13] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In D. H. Fisher, editor, *Proc. International Conference on Machine Learning*, pages 143–151, Nashville, TN, July 1997. Morgan Kaufmann.
- [14] T. Joachims. Text categorisation with support vector machines: Learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *Proc. European Conference on Machine Learning*, pages 137–142, Chemnitz, Germany, 1998. Springer.
- [15] D. Lewis, R. Schapire, J. Callan, and R. Papka. Training algorithms for linear text classifiers. In H. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 296–306, Zurich, Switzerland, August 1996.
- [16] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. Available at: <http://www.cs.cmu.edu/~mccallum/bow>, 1996.

- [17] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.
- [18] M. Persin. Document filtering for fast ranking. In W.B. Croft and C.J. van Rijsbergen, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 339–348, Dublin, Ireland, 1994.
- [19] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10):749–764, 1996.
- [20] S. E. Robertson, S. Walker, and M. Beaulieu. Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track. In E. M. Voorhees and D. K. Harman, editors, *Proc. Text REtrieval Conference*, pages 253–264, Gaithersburg, MD, November 1998. NIST publication number SN003-003-03614-5.
- [21] J.J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall, 1971.
- [22] G. Salton, editor. *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.
- [23] G. Salton. *Automatic Text Processing*. Addison Wesley, Massachusetts, 1989.
- [24] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 222–229, 2002.
- [25] H. Schütze, D. A. Hull, and J. O. Pedersen. A comparison of classifiers and document representations for the routing problem. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 229–237, Seattle, WA, 1995.
- [26] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.
- [27] V. Shanks and H. E. Williams. Fast categorisation of large document collections. In *Proc. International Symposium on String Processing and Information Retrieval*, pages 194–204, San Rafael, Chile, 2001. IEEE Computer Society.
- [28] A. Singhal, G. Salton, M. Mitra, and C. Buckley. Document length normalisation. *Information Processing & Management*, 32(5):619–633, 1996.
- [29] A. Turpin. *Efficient Prefix Coding*. PhD thesis, The University of Melbourne, Melbourne, Australia, 1999.
- [30] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, second edition, 1979.
- [31] E. M. Voorhees and C. Buckley. The effect of topic set size on retrieval experiment error. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 316–323, 2002.
- [32] H.E. Williams and J. Zobel. Compressing integers for fast file access. *Computer Journal*, 42(3):193–201, 1999.
- [33] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical machine learning tools and techniques with Java implementations. In N. Kasabov and K. Ko, editors, *Proc. ICONIP/ANZIIS/ANNES'99 International Workshop: Emerging Knowledge Engineering and Connectionist-Based Information Systems*, pages 192–196, Dunedin, NZ, 1999.
- [34] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Los Altos, CA, second edition, 1999.
- [35] Y. Zhang, J. Callan, and T. Minka. Novelty and redundancy detection in adaptive filtering. In K. Järvelin, M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 81–88, 2002.
- [36] J. Zobel and A. Moffat. Exploring the similarity space. *ACM SIGIR Forum*, 32(1):18–34, Spring 1998.