

On Compression-Based Text Classification

Yuval Marton^{1*}, Ning Wu², and Lisa Hellerstein^{2**}

¹ University of Maryland, Department of Linguistics, 1401 Marie Mount Hall,
College Park, MD 20742-7505 ymarton@umiacs.umd.edu

² Polytechnic University, Department of Computer and Information Science, 5
Metrotech Center, Brooklyn, NY, 11201 wning@cis.poly.edu hstein@cis.poly.edu

Abstract. Compression-based text classification methods are easy to apply, requiring virtually no preprocessing of the data. Most such methods are character-based, and thus have the potential to automatically capture non-word features of a document, such as punctuation, word-stems, and features spanning more than one word. However, compression-based classification methods have drawbacks (such as slow running time), and not all such methods are equally effective. We present the results of a number of experiments designed to evaluate the effectiveness and behavior of different compression-based text classification methods on English text. Among our experiments are some specifically designed to test whether the ability to capture non-word (including super-word) features causes character-based text compression methods to achieve more accurate classification.

1 Introduction

Text classification is the task of taking a set of input documents that are labeled according to some classification (e.g. by topic, author, or style) and using that information to classify other, unlabeled documents. Many different methods have been used for text classification, including support vector machines (SVM), logistic regression, boosting, Naive Bayes, nearest neighbor (kNN), and language modeling (cf.[30, 39, 38, 23]).

Compression-based classification is a non-standard approach to classification. It was discovered independently by different researchers, and has been explored by proponents as well as opponents [16, 34, 1, 13, 8, 33, 11, 19]. Compression programs build a model or dictionary of the files they process. Thus compression can be used to “train” classifiers on the labeled documents for each class. Classification of a new document is done by compressing it multiple times, each time using a different class model or dictionary obtained during “training”. The new document is assigned to the class that yielded the highest compression rate. This procedure can be viewed from an information-theoretic perspective: the

* Part of this research was performed while Y. Marton was studying at Polytechnic University.

** Research of L. Hellerstein and her coauthors partially supported by NSF Grants CCR-9877122 and ITR-0205647 and by the Othmer Institute for Interdisciplinary Studies at Polytechnic University

compression rate measures the cross-entropy between the training text and the new document, and the new document is assigned to the class whose training text minimizes that cross-entropy (see e.g. [33]).

A main attraction of compression-based methods for classification is that they are extremely easy to apply. They require virtually no preprocessing of the input documents. Moreover, the compression-based classification procedures used by Khmelev, in [16], and by Benedetto et al. [1] enable average computer users with access to off-the-shelf compression programs to easily perform classification. These procedures run quite slowly, however, and thus are not suitable when speed is important.

Most text classification methods are word-based; they treat a text document as a collection of words (or stems). In contrast, nearly all research on compression-based classification has been done using byte/character-based compression methods ([34] is an exception). Researchers have noted that character-based classification methods have a potential advantage over word-based methods, in that they are able to automatically capture document features other than words (cf. [24, 8]). Such non-word features include subword features such as stems, superword features that span more than one word, and punctuation.

In this paper, we present a variety of experiments using compression-based classification methods on English text. For simplicity, we restrict our experiments to problems in which each document belongs to exactly one class. We measure performance in terms of micro-averaged accuracy (total number of correct classifications over total number of tests), which is a useful measure for problems with single-class labels, and classifiers with no tuning parameters [38]. We perform our experiments using three standard off-the-shelf compression methods, RAR, gzip, and LZW, on topic classification and authorship attribution tasks. We compare the procedure of Kukushkina et al. [16] (which builds one model or dictionary per class) to the procedure of Benedetto et al. [1] (which is a nearest neighbor approach). We present novel experiments designed to address the question of whether compression methods do, in fact, benefit from their potential to capture non-word features. We believe these experiments are also relevant to the study of other character-based classification methods. We also examine the change in classification accuracy as the amount of training data increases, and explore the effect of imbalanced class size.

We begin with the history of compression-based text classification in Section 2; we then discuss three classification procedures (SMDL, AMDL, and BCN) in Section 3, the compression programs used here for classification in Section 4, and the corpora they were tested on in Section 5; we present our experiments in Section 6, and our conclusions in Section 7.

2 Related Work

It is difficult to determine who first suggested using compression for classification (cf. [8, 11, 19]). Here we review previous experimental work on the approach. Khmelev [16] performed experiments using a large variety of compres-

sion methods to classify a large corpus of Russian literary works by author. Thaper used LZ78, character-based PPM, and word-based PPM to classify English literary works by author [34]. Frank et al. applied PPM to topic classification [8]. They concluded that compression methods are handicapped by their inability to exploit a handful of highly informative terms. Subsequently, Teahan and Harper [33] applied variants of PPM to topic classification. They concluded that compression could, in fact, be an effective method of classification.

Benedetto et al. used gzip and a nearest neighbor procedure for authorship classification [1]. Their paper received media attention (e.g., [28]), and generated some controversy [14, 3, 11, 2]. In this paper we examine one question raised by a critic of the paper, and contested by the authors – whether gzip and the nearest neighbor procedure do, in fact, perform accurate classification [2, 11].

Recently, Khmelev and Teahan tested a number of classification methods on the Reuters corpus, including both SVM and compression-based classification (with gzip and RAR) [13]. RAR outperformed all other methods, including SVM. (However, the authors noted that their method of using SVM for multi-class problems might not be optimal). Khmelev and Teahan hypothesized that SVM and other methods using “bag-of-words” models are handicapped by their inability to capture word sequences.

Peng et al. proposed the use of character-based language modeling methods for text classification and achieved excellent results [23, 24]. As they noted, their work is related to Teahan’s work on classification with PPM compression [32, 33]. Both PPM compression and language modeling work by building n-gram Markov models of the text. Both calculate the degree of match between the learned model and the test document via a cross-entropy calculation.

Other previous approaches to character-based text classification include the work of Damashek [6], Cavnar and Trenkle [4], and Khmelev and Tweedie [15].

3 Classification procedures

We discuss three different compression-based classification procedures from the literature. We refer to them as the *standard MDL* (minimum description length), the *approximate MDL*, and the *best-compression neighbor* procedures.

The standard MDL procedure (SMDL) was used in [8, 34, 33]. It is analogous to the procedures used in Multinomial Naive Bayes text classification (e.g., [18]) and language-modeling methods [23], but typically does not allow the use of off-the-shelf compression methods. Given training documents for categories C_1, \dots, C_n SMDL forms for each C_i a single file A_i consisting of all documents in that category. It then runs the compression algorithm on each A_i to obtain a model (or dictionary) M_i . Then, for each M_i , it runs the compression algorithm “statically” on a test file T , i.e., it uses model M_i as input and doesn’t update it as T is processed. Finally, it assigns test document T to the class i whose model M_i achieves the best compression of T . We define SMDL here for completeness, but in this paper we do not present any experimental results using SMDL.

In most of our experiments, we use the approximate MDL procedure (AMDL) proposed by Khmelev [16] and used by Khmelev and Teahan [13]. Suppose the input documents are from categories C_1, \dots, C_n . AMDL, like SMDL, forms for each C_i a single training file A_i consisting of all training documents in C_i . It then runs the compression program on each A_i to produce a compressed file \mathcal{A}_i of length $|\mathcal{A}_i|$. Given a test file T , AMDL appends T to each A_i , producing A_iT . It then runs the compression program on each A_iT to produce a compressed file \mathcal{A}_iT . Finally, it assigns T to the class C_i that minimizes the compressed size difference $v_i = |\mathcal{A}_iT| - |\mathcal{A}_i|$. The value v_i can be viewed as an estimate of the cross-entropy of text T with respect to text A_i . AMDL can be viewed as an attempt to approximate SMDL with off-the-shelf compression methods, which are “adaptive” with regard to test documents, rather than “static” as above.

The best-compression neighbor (BCN) procedure was developed by Benedetto et al. [1]. They use a similar approach to AMDL, but instead of concatenating all the training documents in a class into a single input file, they keep each training document D in a separate file. They concatenate test document T to each D , forming DT , and calculate $v_{DT} = |\mathcal{DT}| - |\mathcal{D}|$, the difference between the sizes of the compressed versions of DT and D . Then they assign T to the class containing the document D that minimizes v_{DT} . Their procedure is thus a nearest-neighbor approach, using v_{DT} as the distance measure. As we discuss in Section 6.2 the BCN procedure can require significantly more running time than AMDL.

SMDL will typically be faster than both AMDL and BCN in classifying a new file, because it uses *saved* models or dictionaries for each class. In AMDL and BCN, the new file is concatenated to the training files, causing models or dictionaries for the training files to be recomputed. Note that if one can alter the compression program’s source code, runtime can be dramatically decreased by not actually writing compressed files to disk. One can instead just calculate the number of bytes that would be written into in each compressed file.

4 Classification Methods

Gzip [12] is a compression program available on most UNIX systems. It uses Lempel-Ziv compression (LZ77), a dictionary-based scheme. We used the command line option “-9fc”, for best compression. Its efficacy in classification is limited by its use of a sliding window. Although we were not able to definitely ascertain the size of our gzip version’s window, we assume it is of size 32K; this is a typical size of a gzip sliding window, and our experimental results are consistent with that size.

LZW is a well-known, dictionary-based compression method. We used a straightforward implementation of LZW, based on published source code [21], which we modified slightly, to increase the dictionary size from 14 to 16 bits. On large datasets, even the increased 16-bit dictionary becomes full, usually after processing approximately 300KB of text.

RAR is a proprietary shareware program [25]. In the reported experiments, we used the default mode (no command line options). Current versions of RAR, such as the one we experimented with, can use either LZ (Lempel-Ziv) based or PPM based compression, and chooses between them based on the input data (early versions of RAR used only LZ compression) [27]. For text, it usually uses PPM based compression, in particular, a version of the PPMII algorithm due to Shkarin [29]. In classification experiments performed by Khmelev and Teahan, the performance of RAR was similar to the performance of PPMC [13].

5 Corpora

As mentioned in the introduction, for simplicity we restricted our experiments to problems in which each document belongs to exactly one class. We used the following corpora.

20 Newsgroups (20news)³ – This widely-used corpus consists of approximately 20,000 postings that are labeled by the Usenet discussion groups to which they were posted (around 1000 postings in each of the corpus’ 20 newsgroups). We used J. Rennie’s version of the corpus⁴, in which duplicate postings to more than one newsgroup were removed, and in each posting most headers were removed, while Subject and From fields were retained. This subset contains 18828 documents. We used 5 random splits of 80/20 training/testing. We also used **10 Newsgroups (10news)**, a subset of the 20news corpus, consisting of the 10 categories containing the most documents (and also the most bytes), as in [33]. This subset contains 8998 documents, many of which are 2-6K bytes long, although some are as short as 100 bytes or as long as 51K bytes. No class has fewer than 700KB of training data. We used one random split: 80% training, 20% testing.

Industry Sector⁵ – This dataset consists of approximately 6000 company web pages classified by industry sector. Many of the web pages are 10-20K long, although some are as small as 370 bytes, and others as long as 128K. Training data per sector varies from about 100KB to over 700KB. The 105 sectors are arranged in a 2-level hierarchy, which we ignored. This corpus has been used previously in other text classification experiments (cf. [18, 38, 9, 26]). According to [9], 15 of the web pages in this corpus appear in more than one category. We did not remove these. We used one random 80% training, 20% testing split, and excluded the 0.3% of the test documents that were empty.

Reuters-10 (R10) – This corpus is a subset of the popular Reuters-21578 corpus⁶. We used the ModApte split for dividing the documents into training and test sets. We removed any articles appearing in more than one class. We then

³ Some researchers use this corpus for what they call “genre classification” - a coarser resolution than the usual topic classification problem [18, 32]

⁴ <http://people.csail.mit.edu/people/jrennie/20Newsgroups/20news-18828.tar.gz>

⁵ <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-4/text-learning/www/sector-data.tar.gz>, made available by Market Guide Inc. (www.marketguide.com)

⁶ At <http://www.daviddlewis.com/resources/testcollections/reuters21578>

removed all articles from classes that lacked either training or testing articles. Finally, we selected the 10 classes containing the most documents (and bytes). The resulting corpus has 5444 training documents (4MB), and 2150 tests (1.4MB). Document size ranges from 84 bytes to 6.4K. Training data per class varies from 91KB to 1.27MB.

Reuters-Author (R9) – This is another subset of the Reuters-21578 corpus, but with articles labeled by author, rather than topic. We chose the 9 authors who contributed at least 12 articles, and then included the first 12 articles from each author. We used 6-fold cross-validation, with 10 training and 2 testing documents per class in each fold. Training data per author varies from 24KB to 40KB.

Gutenberg-10 (Gu-10) – This dataset, used in experiments by [34] consists of 4 works of each of 10 well known authors (40 works in total), all taken from the Gutenberg Project⁷. We used 4-fold cross-validation, with 3 training and 1 test document per class in each fold. Some works are as short as 123KB, and some as long as 1.1MB (many are novels). Training data per class ranges from 416KB to 2.5MB.

Federalist Papers (Fed.) – This is a classic corpus in authorship identification literature (cf. [20, 33]), and is also available at the Gutenberg Project. We arbitrarily removed the first of two slightly different versions of Paper 70. We trained on six Hamilton papers (#65-70), six of Madison’s (#40-45), and tested on twelve disputed papers (#49-58, 62-63) and on three undisputed papers of each (#46-48, and #59-61, respectively). Each Paper is 7-21KB long. The training data sizes for the two classes are 82KB and 109KB.

6 Experiments

6.1 A comparison of RAR, gzip, and LZW on full corpora

We compared RAR, gzip, and LZW on a variety of corpora, using AMDL (described in Section 3). Our experimental results are presented in Table 1.

RAR is the best performing method on all but the small Reuters-9 corpus, in which it correctly classified only 2 test documents fewer than gzip (out of 108). RAR’s superiority confirms results obtained by Khmelev and Teahan [13].

The poor performance of gzip on Gutenberg-10 is due both to its 32K sliding window, and to how that window interacts with AMDL. Under AMDL, the training file for each Gutenberg-10 author consists of the concatenation of three of the author’s works. After processing the training file, gzip’s sliding window has information only from the last 32K bytes of the file. Since works in Gutenberg-10 are longer than 32K bytes, gzip is effectively trained on only 32K bytes from each author, and those 32K bytes are drawn from only one of the author’s three training works. We discuss the effect of gzip’s window on its accuracy again in Sections 6.2 and 6.6.

⁷ <http://gutenberg.net>

LZW performs only slightly better than gzip on Gutenberg-10. Like gzip, it is unable to use all three training documents in each class. Since LZW’s dictionary becomes full after reading approximately 300KB of text, and no changes to the dictionary are made after that point, LZW uses information from only the first 300KB of each class training file in doing classification. The shortest documents in Gutenberg-10 are about 150KB, and many are over 500KB, so for most classes LZW benefits only from the first (and sometimes the second) document in each class training file. Note that gzip and LZW effectively use different training documents, since gzip’s information comes from the *third* document in each class training file. In Section 6.6 we present results suggesting that the limited dictionary size may not be the only reason for LZW’s mediocre performance.

We are not sure what the proprietary RAR does, but as mentioned in Section 4, it usually defaults to PPMII when run on text. PPM methods typically use data to compute estimates for the transition probabilities of the relevant Markov chains. In contrast to gzip and LZW, RAR has the potential to benefit from *all three* training documents in each Gutenberg-10 class.

Despite its 32K window, gzip does perform almost as well as RAR on the Reuters-9 corpus. One reason may be Reuter-9’s small class sizes; gzip can use all the class training data for about half the classes, and most of the training data for the rest. But in fact, as can be seen in gzip’s results for Reuters-10, gzip doesn’t need a high percentage of the data to perform well in all tasks. It seems that on some corpora, a 32K window is a limitation for gzip (e.g., Sector; see Section 6.6), but on others, a 32K window is enough (e.g., Reuters-10). Note that the particular documents that end up in gzip’s sliding window may happen to be especially “good” or “bad” ones, potentially making its accuracy unstable.

Our 90.5% result for RAR on 20news is competitive with some of the best results reported in the literature, such as the 89.23% accuracy reported by Peng et al. using language modeling techniques [23], the 82.1% obtained by Teahan and Harper [33], and the 86.2% reported by Rennie et al. using an extended version of Nave Bayes [26]. The 94.8% accuracy figure reported by Zhang and Oles [38] should not be compared to the above results, because “Newsgroup:” headers were not removed in their experiment [37].

Our 89.6% result for RAR on Sector can be compared with the 64.5% obtained by Ghani using Multinomial Nave Bayes [9], but is not as high as the 93.6% reported by Zhang and Oles using SVM [38], or the 92.3% by Rennie et al. using their extended version of Nave Bayes [26].

6.2 A comparison of AMDL and BCN procedures

In Table 2 we present the results of experiments with gzip and RAR comparing the AMDL procedure to the BCN procedure.

On Gutenberg-10 and on the small Federalist Papers corpus, gzip performs better under BCN than it does under AMDL. It performs only slightly worse on the small Reuters-9 corpus. We also applied gzip to 10news (not shown in the table), and its accuracy increased dramatically from 0.56 to 0.89 when we used BCN instead of AMDL. On Gutenberg-10, BCN allows gzip to make use of 32KB

Table 1. RAR, LZW, and gzip using AMDL

Corpus	RAR	LZW	GZIP
Federalist	0.94	0.83	0.67
Gutenberg-10	0.82	0.65	0.62
Reuters-9	0.78	0.66	0.79
Reuters-10	0.87	0.84	0.83
10news (20news)	0.96 (0.90)	0.66	0.56 (0.47)
Sector	0.90	0.61	0.19

from *each* of an author’s three works, as opposed to 32KB from *one* of those works. Similarly, on 10news, gzip can use up to 32KB of each training document, rather than just a handful of them. On Reuters-9, gzip was already competitive with RAR in AMDL, and remains competitive with RAR in BCN. From these results, it seems that gzip’s sliding window size is *not* a severe handicap under BCN (as opposed to what is suggested in [13]). In fact, the performance of RAR and gzip are very similar across these corpora under BCN.

BCN is a 1-nearest-neighbor method. It is well known that 1NN is highly sensitive to noise. BCN might be improved by using a k-nearest-neighbor approach, for some $k > 1$ (see e.g. [30, 35] for uses of k -NN in text categorization).

In our experiments, BCN ran much more slowly than AMDL. This is not surprising, because in BCN, each byte of a test file is compressed as many times as there are training *documents*, because the test file is concatenated to each training file before compression. In contrast, in AMDL, each byte of a test file is compressed as many times as there are training *classes*. Thus, for example, if a Reuters-10 experiment takes several hours using AMDL, it can easily take over a month using BCN. All remaining experiments reported in this paper use AMDL.

Table 2. RAR, LZW and gzip in AMDL and BCN procedures

Corpus	RAR		LZW		gzip	
	AMDL	BCN	AMDL	BCN	AMDL	BCN
Federalist	0.94	0.78	0.83	0.83	0.67	0.78
Gu-10	0.82	0.75	0.65	0.53	0.62	0.72
R-9	0.78	0.77	0.66	0.49	0.79	0.77

6.3 Testing effect of punctuation

To test whether compression-based classification methods successfully exploit patterns of punctuation usage, we devised the following experiment. For each training file, we created a modified version of the file by removing all punctuation and replacing all white spaces (tab, line, paragraph, and page breaks) with

spaces; we call this preprocessing procedure NOP. This kind of preprocessing is typically done for word-based methods, but not for character-based methods. We compared the performance of the compression methods on the original files to their performance on files processed with NOP.

We show the results of our experiments in Table 6.3. The first three lines of the table correspond to authorship classification tasks, and the last three to topic classification tasks. Intuitively, punctuation usage seems an important factor in writing style, and therefore one might expect removal of punctuation to adversely affect authorship classification accuracy. Although this did happen in some of our authorship experiments, accuracy remained the same, or even increased, in many cases.

It is interesting to note that removal of punctuation had a relatively small effect on the performance of RAR, the best algorithm overall. The only two corpora in which its performance changed, Federalist and Reuters-9, contain relatively a small amount of data.

Table 3. Sensitivity to Punctuation Information (Raw files vs. NOP files)

corpus	gzip		LZW		RAR	
	Raw	NOP	Raw	NOP	Raw	NOP
Fed.	0.67	0.39	0.83	0.83	0.94	0.83
Gu-10	0.62	0.65	0.65	0.70	0.82	0.82
R-9	0.79	0.81	0.66	0.62	0.78	0.81
R-10	0.83	0.83	0.84	0.85	0.87	0.87
10news	0.56	0.54	0.66	0.73	0.96	0.96
Sector	0.19	0.22	0.61	0.69	0.90	0.90

6.4 Exploitation of sub-words in character-based methods

We devised two related experiments to test whether compression-based methods exploit sub-word features. Because we wanted to avoid interaction with punctuation effects, we began both experiments using files preprocessed with NOP, rather than with the original files.

In the first experiment, for each word w in the input corpus, we generated a random string s containing between 3 and 5 characters, and then replaced each occurrence of w in the documents with s . The purpose of this procedure was to destroy subword features. For example, “walk” and “walked” might be replaced by “sxq” and “zvro”, thus eliminating their common stem “walk,” and the suffix “ed.” We call this procedure Random-String Words (RSW).

We were concerned, however, that such a radical transformation might affect the compression algorithms in unexpected ways. Therefore, we performed a second experiment, in which We generated a random permutation of the words in the corpus, thus defining a mapping from each word w in the corpus vocabulary

to a unique word w' also in the vocabulary. We then replaced each occurrence of w in the input with w' . Thus, for example, the words “walk” and “walked” might be replaced by the words “the” and “met.” We call this procedure Word Permutation (WP).

The results of our experiments are shown in Table 4.⁸ Note first that in many experiments, the accuracy obtained with RSW is close to the accuracy obtained with NOP. This suggests that the compression algorithms often behave much like word-based methods, and have relatively little regard for what’s *in* a word. Also, contrary to what one might expect, WP does not consistently achieve higher accuracy than RSW.

In some experiments, accuracy was higher with NOP than with RSW and PW, suggesting that the compression methods may exploit subword features to achieve more accurate classification. However, in other experiments, accuracy in RSW or PW equaled, or even exceeded, accuracy in NOP. Character-based compression methods may indeed benefit from exploiting subword features, but our experiments provide mixed evidence for this phenomenon, and the benefit may depend both on corpus and on compression method.

The experiments in which RSW or WP accuracy exceeded NOP accuracy suggest that exploitation of (some) subword features may sometimes have a negative effect on classification accuracy. This is plausible, since subword relationships can be misleading. For example, “of” is a subword of “offer” and “office”. Both RSW and WP reduce such misleading relationships. In RSW, one random word is unlikely to be a subword of another. In WP, shorter words, which are typically function words or stop-words, tend to appear more frequently, and will likely be replaced by longer words. Hence fewer words in a document will accidentally be substrings of other words.

Table 4. Exploitation of subword and superword information

corpus	gzip				LZW				RAR			
	NOP	WP	RSW	WOS	NOP	WP	RSW	WOS	NOP	WP	RSW	WOS
Fed.	0.39	0.61	0.39	0.56	0.83	0.83	0.83	0.83	0.83	0.89	0.89	0.83
Gu-10	0.65	0.50	0.45	0.68	0.70	0.78	0.78	0.72	0.82	0.75	0.72	0.72
R-9	0.81	0.76	0.80	0.70	0.62	0.53	0.64	0.66	0.81	0.78	0.81	0.71
R-10	0.83	0.81	0.81	0.78	0.85	0.84	0.84	0.83	0.87	0.87	0.87	0.85
10news	0.54	0.48	0.51	0.56	0.73	0.54	0.64	0.72	0.96	0.96	0.95	0.90
Sector	0.22	0.20	0.24	0.21	0.69	0.58	0.69	0.66	0.90	0.89	0.84	0.77

⁸ We also performed additional experiments in which we limited the amount of training data per class to 80K. The pattern of results was similar, although not identical, to the results shown in Table 4, with one exception. On 10-news, RAR performed dramatically worse with RSW than with NOP. This result was especially surprising because in our other experiments, we found RAR’s performance to be relatively stable under changes to the training data size.

6.5 Exploitation of superwords in character-based methods

To test the effect of capturing superword information in character-based methods, we devised the following preprocessing procedure: First we preprocessed all documents with NOP, and then we randomly reordered the words in each document. We call this procedure Word Order Scrambling (WOS).

WOS leaves subword and word information intact, while destroying superword relations. If character-based compression methods rely on superword information such as word-based n-gram information, word order scrambling should result in decreased accuracy. Results for our WOS experiments are presented in Table 4. Comparing WOS and NOP results reveals that RAR’s accuracy did decline in all but one corpus after scrambling (the exception is Fed.). Additional experiments in which we limited the amount of training data to 80K per class (not shown), show RAR’s accuracy declining in all but one corpus (R-9), but but the results for LZW and gzip do not show a consistent decline.

6.6 Effects of variable and unbalanced training data size

Methodology There are different ways to artificially vary the amount of training data available to a class. One option is to concatenate all training documents in a class into a single file, and then to truncate the concatenated file at different points. In preliminary experiments using this procedure, we found that the learning curves exhibited strange behavior. We then realized that for small data sizes, the truncated training file contains only a small number of available training documents. In addition, increasing the amount of training data adds new documents to the end of the training file, resulting in jagged learning curves (especially for gzip, with its sliding window). This was especially problematic for Gutenberg-10, in which documents are long, and different documents by the same author can be diverse.

We decided instead, for each desired amount of training data, to use small chunks from as many class training files as possible, within reason. There are many ways to do this; we used the following procedure. Let t be the original number of training documents in a given class. Let $b = \max(\lceil s/t \rceil, 0.5K)$. Truncate each training document D_i ($i=1..t$) in the class to $b_i = \min(b, |D_i|)$ bytes. If $\sum b_i < s$, compensate by restarting this process with the value of b increased by 1. Concatenate the truncated D_1 with a (file containing a) single space followed by the truncated D_2 , space, D_3 , space, and so on. This procedure results in a single mega-document for the class. Truncate the mega-document after s bytes.

To get meaningful results using this approach, one should not use too small a chunk size b . E.g., one shouldn’t take a single byte from each document. The 0.5K parameter was our choice for a minimum chunk size. (We assumed that no document was smaller than .5K.)

We used this procedure to obtain the results shown in Figures 1- 6. Note that, because of the minimum chunk size, not every training document from a class is necessarily in the mega-document used for a given training size s . More importantly, it is also possible that $\text{length}(\text{mega-document}) < s$, if the

total amount of training data in a class is less than s bytes. The larger s gets, the more imbalance in the amount of training data per class. For $s = 40K$, all corpora except Reuters-9 do get 40K bytes of training data. For $s = 80K$, all corpora except Reuters-9 and Reuters-10 have 80K bytes of training data (Reuters-10 has slightly less).

Discussion of experiments varying amount of training data As seen in Figures 1- 6, more training data does not always lead to higher accuracy. Gzip's increase in accuracy is constrained by the size of its sliding window, and it does not exhibit much increase in accuracy beyond 40K. The curves for LZW flatten out somewhere between 300K and 500K, depending on how soon its dictionary fills up (which varies between corpora). Notice that at 40K (and to some extent at 80K), LZW's accuracy is often low compared to RAR's. Thus LZW's poor performance cannot be attributed solely to its small dictionary size, nor to an inability to handle imbalanced training data.

RAR, on the other hand, seems to improve consistently as the amount of training data increases. This reflects RAR's ability to exploit additional data to obtain more exact probability estimates. At the 20K point, RAR's results are similar to gzip's, but then RAR's results climb up, while gzip's do not.

Fig. 1. Federalist Papers

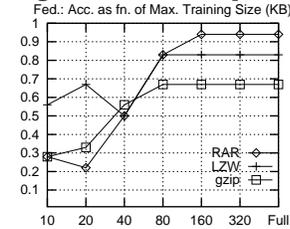


Fig. 2. Gutenberg-10

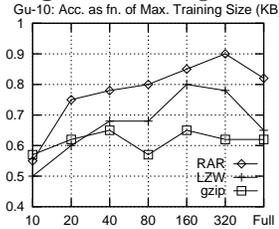


Fig. 3. Reuters-9

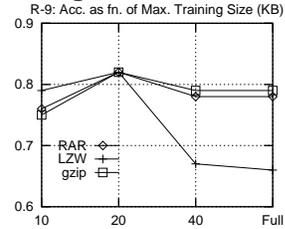


Fig. 4. Reuters-10

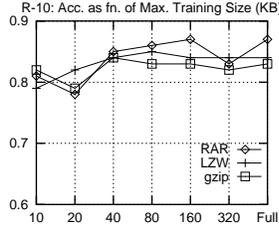


Fig. 5. 10News

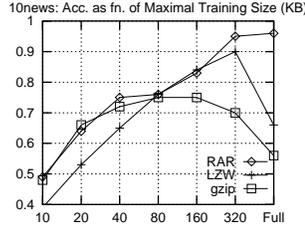
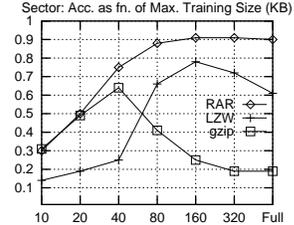


Fig. 6. Sector



The importance of balanced training data The problem of imbalanced training data is well-known, and classification methods are often adversely af-

ected by this imbalance (see e.g. [26]). To test how much training data imbalance affects the accuracy of compression-based classification methods, we performed the following small experiment on Gutenberg-10. We first created a skewed version of the Gutenberg-10 corpus, as follows: we gave classes 1-5 the first 40K of their training data each, and classes 6-10 the first 70K of their training data each. We compared the accuracy obtained on this corpus (with 4-fold cross-validation) to the accuracy obtained on two other versions of the corpus – one in which each class has 40K of training data, and one in which each class has 70K of training data. Results are shown in Table 5. All methods perform worse with the skewed corpus than they do with either the 70K corpus or the 40K corpus. These results suggest that it may be desirable to equalize the amount of training data per class by discarding data from larger classes. Such an approach would be analogous to under-sampling in machine learning.

Table 5. Accuracy of RAR, LZW and gzip on balanced and skewed Gu-10 subset

Method	Skewed-40K/70K	40K	70K
RAR	0.50	.78	.80
GZIP	0.55	.65	.57
LZW	0.53	.65	.72

7 Conclusions and Future Work

We tested three compression methods (RAR, gzip and LZW) under two procedures (AMDL and BCN) on English language topic/genre categorization and authorship attribution problems. RAR almost always produced more accurate classification than LZW and gzip, sometimes by a wide margin. LZW often performed poorly, not always because of its limited size dictionary, and gzip was handicapped by its sliding window on some corpora. We found AMDL to be superior to BCN for RAR, both in runtime and accuracy, We found BCN to be superior to AMDL for gzip’s accuracy, although it runs even more slowly than AMDL. Overall, RAR under AMDL seems to give best results.

There remains a need for careful experiments comparing the current best compression-based classification methods to state-of-the-art classification methods, and to methods similar to the ones explored here, namely PPMC with SMDL (as used by [33]) and the related n-gram language modeling methods of [22, 23]. Further experiments are also needed to determine when artificially balancing class sizes (by throwing away data) is desirable.

We presented a new approach to test whether character-based methods can actually benefit from their ability to capture subword, superword, and other non-word (punctuation) features. Our approach consisted of applying different preprocessing procedures to the corpora, and comparing results between the original corpora and the preprocessed versions. Our subword experiments indicated

that in some cases, compression algorithms may benefit from subword information in performing classification. Our superword experiments provided evidence that RAR (PPM) benefits from information contained in word sequences, and thus successfully exploits an ability not shared by more traditional bag-of-words methods.

References

1. Benedetto, B., Caglioti, E., Loreto, V.: Language trees and zipping. *Physical Review Letters* **88**(048702) (2002).
2. Benedetto, B., Caglioti, E., Loreto, V.: On J. Goodman’s comment, to “Language trees and zipping”. <http://arxiv.org/abs/cond-mat/0203275> July 2004.
3. Benedetto, D. and Caglioti, E.: Benedetto, Caglioti, and Loreto reply. *Physical Review Letters*, **90**(089804) (2003)
4. Cavnar, W. B., Tenkle, J. M.: N-gram-based text categorization. *Proc. of the 3rd Annual Symposium on Document Analysis and Information Retrieval (SDAIR 94)* (1994) 161–175
5. Chen, Stanley F., and Goodman, Joshua: An Empirical Study of smoothing techniques for language modeling. *Proc. of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*. 1998
6. Damashek, M.: Gauging similarity with n-grams: Language-independent categorization of text. *Science* **267**(5199) (1995):843–848
7. Eyheramendy, S., Lewis, D. D., Madigan, D.: On the naive Bayes model for text categorization. *Proc. of the Ninth International Workshop on Artificial Intelligence and Statistics* (2003)
8. Frank, E., Chui, C., Witten, I.H.: Text Categorization Using Compression Models. *Proc. of DCC-00, IEEE Data Compression Conference* (2000) 200–209.
9. Ghani, Rayid: Using Error Correcting Codes for Efficient Text Classification with a Large Number of Categories. KDD project report. Masters Thesis. Center for Automated Learning and Discovery, Carnegie Mellon University (2001)
10. Goodman, Joshua T.: A Bit of Progress in Language Modeling, Extended Version. *Computer Speech and Language*, October 2001, pages 403-434.
11. Goodman, J.: Extended comment on language trees and zipping. <http://arxiv.org/abs/cond-mat/0202383>
12. gzip, a GNU license compression tool, version 1.3.3 (2002-03-08). Copyright 2002 Free Software Foundation Copyright 1992-1993 Jean-loup Gailly
13. Khmelev D., Teahan W.: A repetition based measure for verification of text collections and for text categorization. *Proc. of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (2003):104–110
14. Khmelev, D. V., Teahan, W. J.: Comment: “Language trees and zipping”. *Physical Review Letters*, **90** (2003):089803.
15. Khmelev D., Tweedie F.: Using Markov Chains for Identification of Writers. *Literary and Linguistic Computing* **16**(4) (2001):299–307
16. Kukushkina O. V., Polikarpov A. A., Khmelev D. V.: Using literal and grammatical statistics for authorship attribution. *Problems of Information Transmission* **37** (2001):172–184.
17. Lowenstern D., Hirsh H., Noordwier M., Yianilos P.: DNA Sequence Classification Using Compression-Based Induction. DIMACS Technical Report 95-04 (1995)

18. McCallum, A., Nigam, K.: A Comparison of Event Models for Naive Bayes Text Classification. Proc. of the AAAI-98 Workshop on "Learning for Text Categorization" (1998)
19. Mitchell, T.: Tutorial on machine Learning over natural language documents. <http://www.cs.cmu.edu/~tom/text-learning.ps>
20. Mosteller, F., Wallace, D. L.: Inference and Disputed Authorship: The Federalist. Springer-Verlag, New York, (1964) (Second edition: Applied Bayesian and Classical Inference. (1984)).
21. Nelson, Mark R.: LZW source code. Dr. Dobb's Journal, October, 1989 (Also available at <http://www.dogma.net/markn/articles/lzw/lzw.htm>).
22. Peng, F., Schuurmans, D.: Combining Naive Bayes and n-gram language models for text classification. Proc. of The 25th European Conference on Information Retrieval Research (ECIR03) LNCS 2633 (2003):335-350
23. Peng, F., Schuurmans, D., Wang, S.: Augmenting Naive Bayes classifiers with statistical language models. Information Retrieval **7** (2004):317-345.
24. Peng, F., Schuurmans, D., Wang, S.: Language and task independent text categorization with simple language models. Proc. Human Language Technology Conference of the North American Chapter of the ACL (2003) 189–196
25. RAR compression tool by RAR Labs, Inc. (www.rarlab.com). Version 3.30 (22 Jan 2004). Copyright (c) 1993-2004 Eugene Roshal.
26. Rennie, J. D. M., Shih, L., Teevan, J., Karger, D. R. Tackling the poor assumptions of Naive Bayes text classifiers. Proc. of the Twentieth International Conference on Machine Learning (2003)
27. Rorshal, Eugene (RAR Labs Inc.): Personal communication (2004)
28. Schechter, B.: Fun with your zip program: Sort through texts, and more. New York Times, April 30, 2002.
29. Shkarin, D.: Improving the efficiency of PPM algorithm. Problems of information transmission **34**(3) (2001):44–54 (In Russian. English description available at <http://www.dogma.net/DataCompression/Miscellaneous/PPMILDCC02.pdf>).
30. Sebastiani, F.: Machine learning in automated text categorization, ACM Computing Surveys **34**(1) (2002):1–47
31. Teahan, W.J.: Modelling English Text. PhD thesis, University of Waikato (1998)
32. Teahan, W.J.: Text classification and segmentation using minimum cross-entropy Proc. RIAO-00, 6th International Conference Recherche d'Information Assistee par Ordinateur (2000)
33. Teahan, W. J., Harper, D. J.: Using compression-based language models for text categorization. Proc. of the Workshop on Language Modeling and Information Retrieval (2001)
34. Thaper, N.: Using Compression For Source Based Classification Of Text. Master's Thesis, Massachusetts Institute of Technology (2001).
35. Yang, Yiming: An Evaluation of Statistical Approaches to Text Categorization. Information Retrieval, 1(1/2):67-88. (1999).
36. Yang, Yiming, Pederson, Jan O.: A Comparative Study on Feature Selection in Text Categorization. Proc. of 14th International Conference on Machine Learning (ICML-97).
37. Zhang, Tong: Personal communication (2004).
38. Zhang, Tong, Oles, J. Frank.: Text Categorization Based on Regularized Linear Classification Methods. Information retrieval **4** (2001):5–31.
39. Zhang, J., Jin, R., Yang, Y., Hauptmann, A.G.: Modified Logistic Regression: An Approximation to SVM and Its Applications in Large-Scale Text Categorization. Proc. of the 20th International Conference on Machine Learning (2003):888–895