Applications of lossless compression in adaptive text mining

Ian H. Witten
Department of Computer Science
University of Waikato
Hamilton, New Zealand
e-mail: ihw@cs.waikato.ac.nz

I. Introduction

Text mining is about looking for patterns in natural language text, and may be defined as the process of analyzing text to extract information for particular purposes. Compared with the kind of data stored in databases, text is unstructured, amorphous, and contains information at many different levels. Nevertheless, the motivation for trying to extract information from it is compelling—even if success is only partial. Despite the fact that the problems are difficult to define clearly, interest in text mining is burgeoning because it is perceived to have enormous potential practical utility.

This paper argues that lossless compression, operating within the standard training/testing paradigm of machine learning, is a key technology for text mining. Research in compression has always taken the pragmatic view that files need to be processed whatever they may contain, rather than the normative approach of classical language analysis which generally assumes idealized input: a sequence of sentences, comprising words all of which appear in the dictionary, delimited by single spaces, with punctuation and perhaps numbers but no other extraneous symbols. In practice text—particularly text gathered from the Web, the principal source of material used today—is messy, and many useful clues come from the messiness.

To avoid making prior assumptions about the input, adaptive techniques are standard in modern text compression. Adaptation is exactly what is required to deal with the vagaries of text universally encountered in the real world, typified by the Web.

This paper describes four areas in which compression has been used for text mining: generic entity extraction, text categorization, segmentation into tokens, and acronym extraction, and concludes with more speculative material on structure recognition. Throughout this work, a PPM text compression scheme was used [3], with order 5 (except where otherwise mentioned) and escape method D [7]. However, the results should not be particularly sensitive to the compression method used, although character-based prediction is assumed.

II. Generic entity extraction

Business and professional documents are packed with loosely structured information: phone and fax numbers, street addresses, email addresses and signatures, tables of contents, lists of references, tables, figures, captions, meeting announcements, URLs. In addition, there are countless domain-specific structures—ISBN numbers, stock symbols, chemical structures, and mathematical equations, to name a few.

The information extraction research community has studied these tasks. "Named entities" are defined as proper names and quantities of interest, including personal, organization, and location names, as well as dates, times, percentages, and monetary amounts [2]. The standard approach is manual: tokenizers and grammars are hand-crafted for the particular data being extracted. Commercial text mining software includes IBM's *Intelligent Miner for Text* [10], which uses specific recognition modules carefully programmed for the different data types, Apple's *Data Detectors* [8], which uses language grammars, and the *Text Tokenization Tool* of [6].

An alternative approach to generic entity extraction is to use training instead of explicit programming to detect instances of sublanguages in running text. Character-based language models provide a promising way to recognize lexical tokens. Tokens can be compressed using models derived from different training data, and classified according to which model supports the most economical representation [1].

An Example

In order to assess the power of language models to discriminate tokens, experiments were conducted with information items extracted (manually) from twenty issues of a 4-page, 1500-word, weekly electronic newsletter. Items of the kind that readers might wish to take action on were classified into generic types: people's names; dates and time periods; locations; sources, journals, and book series; organizations; URLs; email addresses; phone numbers; fax numbers; and sums of money. These types are subjective: dates and time periods are lumped together, whereas for some purposes they should be distinguished; personal and organizational names are separated, whereas for some purposes they should be amalgamated. The methodology we describe accommodates all these options: there is no committment to any particular ontology.

Discriminating Isolated Tokens

The first experiment involved the ability to discriminate between different token types when the tokens are taken in isolation. Lists of names, dates, locations, etc. in twenty issues of the newsletter were input to PPM separately to form ten compression models. Each issue contained about 150 tokens, unevenly distributed over token types. In addition, a plain text model was formed from the full text of all these issues. These models were used to identify each of the tokens in a newsletter that did not form part of the training data, on the basis of which model compresses them the most. Although the plain text model could in principle be assigned to a token because it compresses it better than all the specialized models, in fact this never occurred.

Of the 192 tokens in the test data, 40% appeared in the training data (with the same label) and the remainder were new. 90.6% of the total were identified correctly and the remaining 9.4% incorrectly; all errors were on new symbols. Three of the "old" symbols contain line breaks that do not appear in the training data: for example, in the test data `Parallel␣Computing\nJournal` is split across two lines as indicated. However, these items were nevertheless identified correctly. The individual errors are easily explained; some do not seem like errors at all. For example, the place names `Norman` and `Berkeley` were "mis"-identified as people's names, time periods like `Spring␣2000` were misidentified as sources (because of confusion with newsgroups like `comp.software.year-2000`), people's names were confused with organizational names, and so on.

Distinguishing Tokens in Context

When tokens appear in text, contextual information provides additional cues for disambiguating them. Identification must be done conservatively, so that strings of plain text are not misinterpreted as tokens—since there are many strings of plain text, there are countless opportunities for error.

Context often helps recognition: e.g., email addresses in this particular newsletter are always flanked by angle brackets. Conversely, identification may be foiled by misleading context: e.g., some names are preceded by `Rep.`, which reduces the weight of the capitalization evidence for the following word because capitalization routinely follows a period.

The second experiment evaluated the effect of context by assuming that all tokens have been *located* in the test issue, and the task is to *identify* their types *in situ*. If a stretch of text is identified as a token of the appropriate type it will compress better using the specialized model; however, begin- and end-token markers must be coded to indicate this fact. To investigate this, all tokens in the data were replaced by a surrogate symbol that was treated by PPM as a single character (different from all the ASCII characters). A different surrogate was used for each token type. A new model was generated from the modified training data, and the test article was compressed by this model to give a baseline entropy of e_0 bits. Then each token in turn, taken individually, was restored into the test article as plain text and the result recompressed to give entropy e bits. This will (likely) be greater than e_0 because the information required to represent the token itself (almost certainly) exceeds that required to represent its type. Suppose e_m is the token's entropy with respect to model m. Then the net space saved by recognizing this token as belonging to model m is

$$e - (e_0 + e_m) \text{ bits.}$$

This quantity was evaluated for each model to determine which one classified the token best, or whether it was best left as plain text. The procedure was repeated for each token.

When context is taken into account the error rate per token actually increases from 9.4% to 13.5%. However, almost all these "errors" are caused by failure to recognize a token as different from plain text, and the rate of actual mis-recognitions is only 1%—or just two mis-recognitions, one of which is the above-mentioned `Berkeley` being identified as a name.

To mark up a string as a token requires the insertion of two extra symbols: begin- and end-token, and it is this additional overhead that causes the above-noted failures to recognize tokens. However, the tradeoff between actual errors and failures to identify can be adjusted by using a non-zero threshold when comparing the compression for a particular token with the compression when its characters are interpreted as plain text. This allows a small increase in the number of errors to be sacrificed for a larger decrease in identification failures.

Locating Tokens in Context

Tokens can be located by considering the input as an interleaved sequence of information from different sources. Every token is to be bracketed by *begin-token* and *end-token* markers; the problem is to "correct" text by inserting such markers appropriately. The markers also identify the type of token in question—thus we have *begin-name-token*, *end-name-token*, etc., written as `<n>`, `</n>`. Whenever *begin-token* is encountered, the encoder switches to the compression model appropriate to that token type, initialized to a null prior context. Whenever *end-token* is encountered, the encoder reverts to the plain text model that was in effect before, replacing the token by a single symbol representing that token type.

The algorithm takes a string of text and works out the optimal sequence of models that would produce it, along with their placement. It works Viterbi-style, processing the input characters to build a tree in which each path from root to leaf represents a string of characters that is a possible interpretation of the input. The paths are alternative output strings, and *begin-token* and *end-token* symbols appear on them. The entropy of a path can be calculated by starting at the root and coding each symbol along the path according to the model that is in force when that symbol is reached. The context is re-initialized to a unique starting token whenever *begin-token* is encountered, and the appropriate model is entered. On encountering *end-token*, it is encoded and the context reverts to what it was before.

What causes the tree to branch is the insertion of *begin-token* symbols for every possible token type, and the *end-token* symbol—which must be for the currently active token type so that nesting is properly respected. To expand the tree, a list of open leaves is maintained, each recording the point in the input string that has been reached and the entropy value up to that point. The lowest-entropy leaf is chosen for expansion at each stage. Unless the tree and the list of open leaves are pruned, they grow very large very quickly. A beam search is used, and pruning operations are applied that remove leaves from the list and therefore prevent the corresponding paths from growing further.

To evaluate the procedure for locating tokens in context, we used the training data from the same issues of the newsletter as before, and the same single issue for testing. The errors and mis-recognitions noted above when identifying tokens in context (rates of 1% and 12.5% respectively) also occur when locating tokens. Inevitably there were a few incorrect positive identifications—2.6% of the number of tokens—where a segment of plain text was erroneously declared to be a token. In addition, 8% of tokens suffered from incorrect boundary placement, where the algorithm reported a token at approximately the same place as in the original, but the boundaries were slightly perturbed. Finally, a further 4.7% of tokens suffered discrepancies which were actually errors made inadvertently by the person who marked up the test data.

Discussion

We find these initial results encouraging. There are several ways that they could be improved. The amount of training data—about 3000 tokens, distributed among ten token types—is rather small. The data certainly contains markup errors, probably at about the same rate—4.7% of tokens—as the test file. Many of the mistakes were amongst very similar categories: for example, fax numbers contained embedded phone numbers and were only distinguished by the occurrence of the word *fax*; several times they were confused with phone numbers and this counted as an error. Some of the mistakes were perfectly natural—`Norman` as a name instead of a place, for example. In addition, improvements could likely be made to the pruning algorithm.

III. Text categorization

A central feature of the above approach to generic entity extraction is the basic assumption that a token can be identified by compressing it according to different models and seeing which produces the fewest bits of output. We now examine whether this extends to text categorization—the assignment of natural language texts to predefined categories based on their content. Already-classified documents, which define the categories, are used to build a model that can be used to classify new articles.

Text categorization is a hot topic in machine learning. Typical approaches extract "features," generally words, from text, and use the feature vectors as input to a machine learning scheme that learns how to classify documents. This "bag of words" model neglects word order and contextual effects. It also raises some problems: how to define a "word," what to do with numbers and other non-alphabetic strings, and whether to apply stemming. Because there are so many different features, a selection process is applied to determine the most important words, and the remainder are discarded.

Compression seems to offer a promising alternative approach to categorization, with several potential advantages:

- it yields an overall judgement on the document as a whole, and does not discard information by pre-selecting features;
- it avoids the messy problem of defining word boundaries;
- it deals uniformly with morphological variants of words;
- depending on the model (and its order), it can take account of phrasal effects that span word boundaries;
- it offers a uniform way of dealing with different types of documents—for example, files in a computer system;
- it minimizes arbitrary decisions that inevitably need to be taken to render any learning scheme practical.

We have performed extensive experiments on the use of PPM for categorization using a standard dataset [5]. Best results were obtained with order 2; other values degraded performance in almost all cases—presumably because the amount of training data available is insufficient to justify more complex models.

The Benchmark Data

All our results are based on the Reuters-21578 collection of newswire stories, the standard testbed for the evaluation of text categorization schemes. In total there are 12,902 stories averaging 200 words each, classified into 118 categories. Many stories are assigned to multiple categories, and some are not assigned to any category at all. The distribution among categories is highly skewed: the ten largest—*earnings*, *corporate acquisitions*, *money market*, *grain*, *crude oil*, *trade issues*, *interest*, *shipping*, *wheat*, and *corn*—contain 75% of stories, an average of around 1000 stories each.

Pairwise Discrimination

Applying a straightforward compression methodology to the problem of text categorization quickly yields encouraging results. In the two-class case, to distinguish documents of class A from documents of class B we form separate models M_A and M_B from the training documents of each class. Then, given a test document (different from the training documents), we compress it according to each model and calculate the gain in per-symbol compression obtained by using M_A instead of M_B. We assign the document to class A or B depending on whether this difference is positive or negative, on the principle that M_A will compress documents of class A better, and similarly for M_B. Encouraging results are obtained.

Building Positive and Negative Models

To extend to multiply-classified articles, we decide whether a model belongs to a particular category independently of whether it belongs to any other category. We build positive and negative models for each category, the first from all articles that belong to the category and the second from those that do not. For a particular category C, call these models M_P and M_N.

Given a new article A, denote its length when compressed according to these models by $L[A|M_P]$ and $L[A|M_N]$. The article's probability given the categories C and $\bar{C}$ can be estimated as:

$$Pr[A|C] = 2^{-L[A|M_P]} \qquad Pr[A|\bar{C}] = 2^{-L[A|M_N]}$$

Bayes' formula gives the probability that a particular article A belongs to category C:

$$Pr[C|A] = \frac{Pr[A|C]Pr[C]}{Pr[A|C]Pr[C] + Pr[A|\bar{C}]Pr[\bar{C}]}$$

The prior probability $Pr[C]$ is the proportion of articles belonging to that category, and the denominator is the prior probability of article A.

Setting the Threshold

Deciding whether a new article should in fact be assigned to category C or not presents the familiar recall/precision tradeoff between making the decision liberally, increasing the chance that an article is correctly identified but also increasing the number of "false positives"; or conservatively, reducing the number of false positives at the expense of increased "false negatives." To allow comparison of our results with others, we maximize the average of recall and precision—a figure that is called the "breakeven point."

The basic strategy is to compare the predicted probability $Pr[C|A]$ with a predetermined threshold t, and declare A to have classification C if $Pr[C|A] > t$. The threshold is chosen individually, for each class, to maximize the average of recall and precision for that class. To this end the training data is further divided into a new training set and a "validation set," in the ratio 2:1. The threshold t is chosen to maximize the average of recall and precision for the category (the breakeven point) on the validation set. Then maximum utility is made of the information available by rebuilding M_P and M_N from the full training data.

As an additional benefit, threshold selection automatically compensates for the fact that M_P and M_N are based on different amounts of training data. In general, one expects to achieve better compression with more data.

Results

Table 1 shows the breakeven points obtained, and compares them with results reported for the Naive Bayes and Linear Support Vector Machine methods [4]. PPM outperforms Naive Bayes on the six largest categories (*grain* is the only exception) and is worse on the four smallest ones. It is almost uniformly inferior to the support vector method, *money market* being the only exception.

	PPM	Naive Bayes	LSVM
corn	54.2	65.3	90.3
corporate acquisitions	91.0	87.8	93.6
crude oil	80.7	79.5	88.9
earnings	96.3	95.9	98.0
grain	74.6	78.8	94.6
interest	60.4	64.9	77.7
money market	76.3	56.6	74.5
shipping	81.9	85.4	85.6
trade issues	65.0	63.9	75.9
wheat	64.9	69.7	91.8

Tab. 1: Recall/precision breakeven point for three text categorization schemes

Compared to LSVM, PPM produces particularly bad results on the categories *wheat* and *corn*, which are (almost) proper subsets of the category *grain*. Articles in *grain* summarize the result of harvesting grain products—for example, by listing the tonnage obtained for each crop—and all use very similar terminology. Consequently the model for *wheat* is very likely to assign a high score to *every* article in *grain*.

The occurrence of the word "wheat" is the only notable difference between an article in *grain* that belongs to *wheat* and one that does not. The presence of a single word is unlikely to have a significant effect on overall compression, and this is why PPM performs poorly on these categories.

Support vector machines perform internal feature selection, and can focus on a single word if that is the only discriminating feature of a category. In comparison, Naive Bayes performs badly on the same categories as PPM (*money market* is the only exception): like PPM, it has no mechanism for internal feature selection.

Modifications

The results in Table 1 were obtained quickly, and we found them encouraging. We then made many attempts to improve them, all of which met with failure.

To force PPM to build models that are more likely to discriminate successfully between similar categories, we experimented with a more costly approach. Instead of building one positive and one negative model, we built one positive and 117 negative models for each of the 118 categories. Each negative model only used articles belonging to the corresponding category that did not occur in the set of positive articles. During classification, an article was assigned to a category if the positive model compressed it more than all negative models did. Results were improved slightly for categories like *wheat* and *corn*. However, the support vector method still performed far better. Moreover, compared to the standard PPM method, performance deteriorated on some other categories.

We also experimented with several modifications to the standard procedure, none of which produced any significant improvement over the results reported above:

- not rebuilding the models from the full training data;
- using the same number of stories for building M_P and M_N (usually far more stories are available for M_N);
- priming the models with fresh Reuters data from outside the training and test sets;
- priming the models with the full training data (positive and negative articles);
- artificially increasing the counts for the priming data over those for the training data and *vice versa*;
- using only a quarter of the original training data for validation;
- using escape method A for PPM;
- using a word model of order 0, escaping to a character model of order 2 for unseen words.

Discussion

Compared to state-of-the-art machine learning techniques for categorizing English text, PPM produces inferior results because it is insensitive to subtle differences between articles that belong to a category and those that do not. We do not believe our results are specific to PPM. If the occurrence of a single word is what counts, any compression scheme will likely fail to classify the article correctly. Machine learning schemes fare better because they automatically eliminate irrelevant features.

Compared to word-based approaches, compression-based methods avoid *ad hoc* decisions when preparing input text for the actual learning task. Moreover, these methods transcend the restriction to alphabetic text and apply to arbitrary files. However, feature selection seems to be essential for some text categorization tasks, and this is not incorporated in PPM.

IV. Segmentation into tokens

Conventional text categorization is just one example of many text mining methods that presuppose that the input is somehow divided into lexical tokens. Although "words" delimited by non-alphanumeric characters provide a natural tokenization for many items in ordinary text, this assumption fails in particular cases. Indeed, the superior performance of PPM-based categorization for the *money market* category in Table 1 is likely attributable to an unsuitable notion of "word" in the other methods. As another example, generic tokenization would not allow many date structures (e.g. *30Jul98*, which is used throughout the newsletters of Section II) to be parsed. In general, any prior segmentation into tokens runs the risk of obscuring information.

A simple special case of this scheme for compression-based entity extraction can be used to divide text into words, based on training data that has been segmented by hand. An excellent testbed for this research is the problem of segmenting Chinese text, which is written without using spaces or other word delimiters. Although Chinese readers are accustomed to inferring the corresponding sequence of words as they read, there is considerable ambiguity in the placement of boundaries which must be resolved in the process. Interpreting a text as a sequence of words is necessary for many information retrieval and storage tasks: for example, full-text search and word-based compression.

Inserting spaces into text can be viewed as a hidden Markov modeling problem. Between every pair of characters lies a potential space. Segmentation can be achieved by training a character-based compression model on pre-segmented text, and using a Viterbi-style algorithm to interpolate spaces in a way that maximizes the overall probability of the text.

For non-Chinese readers, we illustrate the success of the space-insertion method by showing its application to English text in Table 2, which is due to Teahan [9]. At the top is the original text, including spaces in the proper places, then the input to the segmentation procedure, and finally the output of the PPM-based segmentation method.

In this experiment PPM was trained on a sample of English, and its recall and precision for space insertion were both

text	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using crocidolite in its Micronite cigarette filters in 1956.
input	theunitofNewYork-basedLoewsCorpthatmakesKentcig arettesstoppedusingcrocidoliteinitsMicronitecigarettef iltersin1956.
output	the unit of New York-based LoewsCorp that makes Kent cigarettes stopped using croc idolite in its Micronite cigarette filters in 1956.

Tab. 2: Segmenting words in English text

99.52%. Corresponding figures for a word-based method that does not use compression-based techniques [11] were 93.56% and 90.03% respectively, a result which is particularly striking because PPM had been trained on only a small fraction of the amount of text used for the other scheme.

PPM performs well on unknown words: although *Micronite* does not occur in the Brown Corpus, it is correctly segmented in Table 2c. There are two errors. First, a space was not inserted into *LoewsCorp* because the single "word" requires only 54.3 bits to encode, whereas *Loews Corp* requires 55.0 bits. Second, an extra space was added to *crocidolite* because that reduced the number of bits required from 58.7 to 55.3.

Existing techniques for Chinese text segmentation are either word-based, or rely on hand-crafted segmentation rules. In contrast, the compression-based methodology is based on character-level models formed adaptively from training text. Such models do not rely on a dictionary and fall back on general properties of language statistics to process novel words. Excellent results have been obtained with the new scheme [12].

V. Acronym extraction

Identifying acronyms in documents—which is certainly also about looking for patterns in text—presents a rather different kind of problem. Webster defines an "acronym" as

> a word formed from the first (or first few) letters of a series of words, as *radar*, from *ra*dio *d*etecting *a*nd *r*anging.

Acronyms are often defined by preceding or following their first use with a textual explanation—as in Webster's example. Finding all acronyms, along with their definitions, in a particular technical document is a problem that has previously been tackled using *ad hoc* heuristics. The information desired—acronyms and their definitions—is relational, and this distinguishes it from the text mining problems discussed above.

It is not immediately obvious how compression can assist in locating relational information such as this. Language statistics certainly differ between acronyms and running text, because the former have a higher density of capital letters and a far higher density of non-initial capital letters. However, it seems unlikely that acronym definitions will be recognized reliably on this basis: they will not be readily distinguished from ordinary language by their letter statistics.

We have experimented with coding potential acronyms with respect to the initial letters of neighboring words, and using the compression achieved to signal the occurrence of an acronym and its definition [13]. Our criterion is whether a candidate acronym could be coded more efficiently using a special model than it is using a regular text compression scheme. A phrase is declared to be an acronym definition if the discrepancy between the number of bits required to code it using a general-purpose compressor and the acronym model exceeds a certain threshold.

We first pre-filter the data by identifying acronym candidates: for initial work we decided to consider words in upper case only. Then we determined two windows for each candidate, one containing 16 preceding words and the other 16 following words. This range covered all acronym definitions in our test data.

Compressing the Acronyms

Candidate acronms are coded using a group of models that express the acronym in terms of the leading letters of the words on either side. This group comprises four separate models. The first tells whether the acronym precedes or follows its definition. The second gives the distance from the acronym to the first word of the definition. The third identifies a sequence of words in the text by a set of offsets from the previous word. The fourth gives the number of letters to be taken from each word. Each of these models is an order-0 PPM model with a standard escape mechanism.

After compressing the acronym candidates with respect to their context, all legal encodings for each acronym are compared and the one that compresses best is selected. For comparison, we compress the acronym using the text model, taking the preceding context into account. The candidate is declared to be an acronym if

$$\frac{\text{bits}_{\text{ acronym model}}}{\text{bits}_{\text{ text model}}} \leq t$$

for some predetermined threshold t. Although subtracting the number of bits seems more easily justified than using the ratio between them, in fact far better results were obtained using the ratio method. We believe that the reason for this is linked to the curious fact that, using a standard text model, longer acronyms tend to compress into fewer bits than do shorter ones. While short acronyms are often spelt out, long ones tend to be pronounced as words. This affects the choice of letters: longer acronyms more closely resemble "natural" words.

Experimental Results

To test these ideas, we conducted an experiment on a sizable sample of technical reports, and calculated recall and precision for acronym identification. The operating point on the recall/precision curve can be adjusted by varying t. While direct comparison with other acronym-extraction methods is not possible because of the use of different text corpora, our scheme performs well and provides a viable basis for extracting acronyms and their definitions from plain text. Compared to other methods, it reduces the need to come up with heuristics for deciding when to accept a candidate acronym—although some prior choices are made when deciding how to code acronyms with respect to their definitions.

VI. Structure recognition

We have shown that while compression is a useful tool for many token classification tasks, it is less impressive for document categorization. As a discriminant, overall compression tends to weaken as the size of individual items grows, because a single holistic measure may become less appropriate. Some decisions depend on the occurrence or non-occurrence of a few special words, which makes feature selection essential. Even in token discrimination, different kinds of token may be distinguishable only by the context in which they occur—for example, author's names and editor's names no doubt enjoy identical statistical properties, but are distinguished in bibliographic references by local context.

The size of individual tokens can often be reduced by extending the techniques described above to work hierarchically. This allows more subtle interactions to be captured. Names are decomposable into forenames, initial, surname; email addresses into username, domain, and top-level domain; and fax numbers contain embedded phone numbers. After analyzing the errors made during the generic entity extraction experiments of Section II, we refined the markup of the training documents to use these decompositions. For instance:

```
Name    <n><f>Ian</f>␣<i>H</i>.␣<s>Witten</s></n>
Email   <e><u>ihw</u>@<d>cs.waikato.ac</d>.<t>nz</t></e>
Fax     <f><p>+64-7-856-2889</p>␣fax</f>
```

We use the term "soft parsing" to denote inference of what is effectively a grammar from example strings, using exactly the same compression methodology as before. During training, models are built for each component of a structured item, as well as the item itself. For example, the *forename* model is trained on all forenames that appear in the training data, while the *name* model is trained on patterns like *forename* followed by space followed by *middle initial* followed by period and space followed by *surname*, where each of the lower-level items—*forename*, *middle initial* and *surname*—are treated by PPM as a single "character" that identifies the kind of token that occurs. When the test file is processed to locate tokens in context, these new tags are inserted into it too. The algorithm described in Section II accommodates nested tokens without any modification at all.

Initial results are mixed. Some errors are corrected (e.g. some names that had been confused with other token types are now correctly marked), but other problems remain (e.g. the fax/phone number mix-up) and a few new ones emerge. Some are caused by the pruning strategies used; others are due to insufficient training data. Despite inconclusive initial results, we believe that soft parsing will prove invaluable in situations with strong hierarchical context (e.g. references and tables).

It is possible that the technique can be extended to the other kinds of tasks considered above. For example, we could mark up an acronym, with its definition. Webster's *radar* example above might look like

```
Acronym ... of␣a␣series␣of␣words,␣as␣<a>radar,␣from
        <d>radio␣detecting␣and␣ranging</d></a>.
```

To capture the essential feature of acronyms—that the word being defined is built from characters in the definition—the search algorithm needs to be extended to consider this possibility.

In text categorization, important features could be highlighted. The word "wheat," which distinguishes articles on *wheat* from other articles in the *grain* category, could be marked in the training data—or by an automatic feature selection process—and the markup inferred in the test data. Such techniques may allow compression-based generalization to tackle problems that require feature selection.

VII. Conclusions

Text mining is a burgeoning new area that is likely to become increasingly important in future. One approach is through the use of hand-tailored heuristics. However, adaptive methods offer significant advantages in construction, debugging, and maintenance. While they suffer from the necessity to mark up large numbers of training documents, this can be alleviated by priming the compression models with appropriate data—lists of names, addresses, etc.—gathered from external sources.

This paper has argued, through examples, that compression forms a sound unifying principle that allows many text mining problems to be tacked adaptively. Success has already been demonstrated on some tasks. Others, notably text categorization, seem less well-suited to the holistic approach that compression offers. However, hierarchical decomposition can be used to strengthen context, and perhaps even to incorporate the results of automatic feature selection.

Compression-based techniques for text mining are in their infancy. Watch them grow.

Acknowledgments

Zane Bray, John Cleary, Eibe Frank, Stuart Inglis, Malika Mahoui, Bill Teahan, YingYing Wen and Stuart Yeates have all contributed greatly to the research described here.

References

[1] Witten, I.H., Bray, Z., Mahoui, M. and Teahan, W.J. (1999) "Text mining: a new frontier for lossless compression." *Proc Data Compression Conference*, pp. 198-207. IEEE Press, Los Alamitos, CA.

[2] Chinchor, N.A. (1999) "Overview of MUC-7/MET-2." *Proc Message Understanding Conference MUC-7.*

[3] Cleary, J.G. and Witten, I.H. (1984) "Data compression using adaptive coding and partial string matching." *IEEE Trans on Communications*, Vol. 32, No. 4, pp. 396–402.

[4] Dumais, S., Platt, J., Heckerman, D. and Sahami, M. (1998) "Inductive learning algorithms and representations for text categorization." *Proc Int Conf on Info and Knowledge Management*, pp. 148–155.

[5] Frank, E., Chiu, C. and Witten, I.H. (2000) "Text categorization using compression models." *Proc Data Compression Conference* (Poster paper). IEEE Press, Los Alamitos, CA. Full version available as Working Paper 00/2, Department of Computer Science, University of Waikato.

[6] Grover, C., Matheson, C. and Mikheev, A. (1999) "TTT: Text Tokenization Tool." http://www.ltg.ed.ac.uk/

[7] Howard, P.G. (1993) The design and analysis of efficient lossless data compression systems. PhD thesis, Brown University.

[8] Nardi, B.A., Miller, J.R. and Wright, D.J. (1998) "Collaborative, programmable intelligent agents." *Comm ACM*, Vol. 41, No. 3, pp. 96–104.

[9] Teahan, W.J. (1997) *Modelling English text.* PhD thesis, University of Waikato, NZ.

[10] Tkach, D. (1997) Text mining technology: Turning information into knowledge. IBM White paper.

[11] Ponte, J.M. and Croft, W.B. (1996) "Useg: A retargetable word segmentation procedure for information retrieval." *Proc on Document Analysis and Information Retrieval*, Las Vegas, Nevada.

[12] Teahan, W.J., Yen, Y., McNab, R. and Witten, I.H. (in press) "A compression-based algorithm for Chinese word segmentation." *Computational Linguistics*.

[13] Yeates, S., Bainbridge, D. and Witten, I.H. (2000) "Using compression to identify acronyms in text." *Proc Data Compression Conference* (Poster paper). IEEE Press, Los Alamitos, CA. Full version available as Working Paper 00/1, Department of Computer Science, University of Waikato.