Text mining:
A new frontier for lossless compression

Ian H. Witten, Zane Bray, Malika Mahoui, Bill Teahan
Computer Science, University of Waikato, Hamilton, New Zealand
email {ihw,zcb,mmahoui,wjt}@cs.waikato.ac.nz

1 Introduction

Data mining, a burgeoning new technology, is about looking for patterns in data. Likewise, text mining is about looking for patterns in text. It may be defined as the process of analyzing text to extract information that is useful for particular purposes. Compared with the kind of data stored in databases, text is unstructured, amorphous, and difficult to deal with. Nevertheless, in modern Western culture, text is the most common vehicle for the formal exchange of information. The motivation for trying to extract information from it is compelling—even if success is only partial.

Analysis of natural language text is commonly thought of as a problem for artificial intelligence. And ever since extravagant claims for mechanical translation in the 1960s prompted an "AI winter" of despair and disillusionment, mainstream computer scientists have—understandably—been skeptical of claims for automatic natural language understanding. The most advanced efforts still rely on tightly-focused domains, small vocabularies, and quantities of specialist domain knowledge, painstakingly programmed in—and still the resulting systems are distressingly brittle. Whether contemporary attempts to codify "common-sense knowledge" (e.g. Lenat, 1995) will make much of a difference remains to be seen. Although corpus-driven, statistical language analysis (e.g. Garside *et al*., 1987) represents a promising approach for producing robust parsers, it does not help in putting the structures that are extracted to any use.

Text mining is possible because you do not have to understand text in order to extract useful information from it. Here are four examples. First, if only names could be identified, links could be inserted automatically to other places that mention the same name—links that are "dynamically evaluated" by calling upon a search engine to bind them at click time. Second, actions can be associated with different types of data, using either explicit programming or programming-by-demonstration techniques. A day/time specification appearing anywhere within one's email could be associated with diary actions such as updating a personal organizer or creating an automatic reminder, and each mention of a day/time in the text could raise a popup menu of calendar-based actions. Third, text could be mined for data in tabular format, allowing databases to be created from formatted tables such as stock-market information on Web pages. Fourth, an agent could monitor incoming newswire stories for company names and collect documents that mention them—an automated press clipping service.

This paper aims to promote text compression as a key technology for text mining.

2 Structured information in text

Looking for patterns in text is really the same as looking for structured data inside documents. According to Nardi *et al*. (1998), "a common user complaint is that they cannot easily take action on the structured information found in everyday documents ... Ordinary documents are full of such structured information: phone numbers, fax numbers, street addresses, email addresses, email signatures, abstracts, tables of contents, lists of references, tables, figures, captions, meeting announcements, Web addresses, and more.

In addition, there are countless domain-specific structures, such as ISBN numbers, stock symbols, chemical structures, and mathematical equations."

As an example, Table 1 shows information items extracted (manually) from a 4-page, 1500-word, weekly electronic newsletter; items of the kind that readers might wish to take action on. They are 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. Identifying these types is rather subjective. For example, 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—indeed it may be impossible for a person (let alone a computer) to distinguish them. The methodology we develop here accommodates all these options: unlike AI approaches to natural language understanding it is not committed to any particular ontology.

The sheer quantity of different information items in Table 1 is impressive: 175 items, in ten categories, from a mere four pages. The volume of such information items is highly dependent on the kind of text—one would expect far less in a novel, for example—but this example is not atypical of the factual, newsy, writings that information workers frequently have to scan on a regular basis.

Nardi *et al*. (1998) define structured information as "data recognizable by a grammar," and go on to describe how such data can be specified by an explicit grammar and acted on by "intelligent agents." Despite the intuitive appeal of being able to define data detectors that trigger particular kinds of actions, there are practical problems with this approach.

First, grammars presuppose that the input is somehow divided into lexical tokens, and although "words" delimited by non-alphanumeric characters provide a natural tokenization for many of the items in Table 1, such a decision will turn out to be restrictive in particular cases. For example, generic tokenization would not allow the unusual date structure in this particular document (e.g. *30Jul98*) to be recognized. In general, any prior division into tokens runs the risk of obscuring information.

Second, grammars make definitive judgements: they paint the world black and white. Yet the situation exemplified by Table 1 reeks of ambiguity. A particular name might be a person's name, a place name, a company name—or all three. A name appearing at the beginning of a sentence may be indistinguishable from an ordinary capitalized word that starts the sentence—and it is not even completely clear what to "start a sentence" means. Determining what is and what is not a time period is not always easy—the notion of "time period" in natural language text is ill-defined. Text, with its richness and ambiguity, does not necessarily support hard and fast distinctions. Email addresses and URLs are exceptions: but, of course, they are not "natural" language.

Third, recognition should degrade gracefully when faced with real text, that is, text containing errors. Traditional grammars decide whether a particular string is or is not recognized, whereas in this application it is often helpful to be able to recognize a string as belonging to a particular type even though it is malformed, or a name even though it is misspelled. (The newsletter used for Table 1 is unusual in that it is virtually error free.)

Fourth and most important, text mining will require incremental, evolutionary development of grammars. The problems are not fully defined in advance. Grammars will have to be modified to take account of new data. This is not easy: the addition of just one new example can completely alter a grammar and render worthless all the work that has been expended in building it.

Finally, many of the items in Table 1 cannot be recognized by conventional grammars. Names are a good example. Some will have been encountered before; for them, table

People's names (*n*)

Al Kamen
Barbara Davies
Bill Park
Bruce Sterling
Ed Royce
Eric Bonabeau
Erricos John Kontoghiorghes
Heather Wilson
John Holland
John R. Koza
Kung-Kiu Lau
Lakhmi C. Jain
Lashon Booker
Lily Laws
Maria Zemankova
Mark Sanford
Martyne Page
Mike Cassidy
Po Bronson
Randall B. Caldwell
Robert L. Park
Robert Tolksdorf
Sherwood L. Boehlert
Simon Taylor
Sorin C. Istrail
Stewart Robinson
Terry Labach
Vernon Ehlers
Zoran Obradovic

Sums of money (*m*)

$1K
$24K
$60
$65K
$70
$78K
$100

Dates/time periods (*d*)

30Jul98
31Jul98
02Aug98
04Aug98
05Aug98
07Aug98
08Aug98
09Aug98
10Aug98
11Aug98
13Aug98
14Aug98
15Aug98
August 18, 1998
01Sep98
15Sep98
15Oct98
31Oct98
10Nov98
01Dec98
01Apr99
Nov97
Jul98
Aug98
Mar99
August
July
Spring 1999
Spring 2000
1993-4
1999
120 days
eight years
eight-week
end of 1999
late 1999
month
twelve-year period

Email addresses (*e*)

amin@cse.unsw.edu.au
bonabeau@santafe.edu
booker@mitre.org
cbd-support@gpo.gov
erricos.kontoghiorghes@info. unine.ch
espaa@soc.plym.ac.uk
hermes@iway.fr
koza@cs.stanford.edu
koza@genetic-programming. org
kung-kiu@cs.man.ac.uk
l.jain@unisa.edu.au
mzemanko@nsf.gov
rbcaldwell@delphi.com
ricos@dcs.qmw.ac.uk
s.l.robinson@aston.ac.uk
scistra@frodo2.cs.sandia.gov
simon.taylor@brunel.ac.uk
thclax00@ukcc.uky.edu
tolk@cs.tu-berlin.de
zoran@eecs.wsu.edu

Organizations (*o*)

ACM
Austrian Research Inst. for AI
Bureau of Labor Statistics
CRC Press
Case Western Reserve U.
Fraunhofer CRCG
Ida Sproul Hall
Kluwer Academic Publishers
NSF
Nohital Systems
Oregon Graduate Inst.
Permanent Solutions
Random House
Santa Fe Institute
UOklahoma
UTrento

Locations (*l*)

Beaverton
Berkeley
Britain
Canada
Cleveland
Italy
Montreal
NM
Norman
Providence, RI
Quebec
Silicon Valley
Stanford
US
Vienna
the Valley

Phone numbers (*p*)

650-941-0336
(703) 883-7609
+44 161 275 5716
+44-1752-232 558

Fax numbers (*f*)

650-941-9430 fax
+44 161 275 6204 Fax
(703) 883-6435 fax
+44-1752-232 540 fax

Sources, journals, book series (*s*)

Autonomous Agents and Multi-Agent Systems Journal
Commerce Business Daily (CBD)
Computational Molecular Biology Series
DAI-List
ECOLOG-L
Evolutionary Computation Journal
Genetic Programming book series
IRList
International Series on Computational Intelligence
J. of Complex Systems
J. of Computational Intelligence in Finance (JCIF)
J. of Symbolic Computation (JSC)
J. of the Operational Research Society
Parallel Computing Journal
Pattern Analysis and Applications (PAA)
QOTD
SciAm
TechWeb
WHAT'S NEW
Washington Post
Wired
comp.ai.alife
comp.ai.doc-analysis.ocr
comp.ai.genetic
comp.ai.neural-nets
comp.simulation
dbworld
sci.math.num-analysis
sci.nanotech

URLs (*u*)

http://cbdnet.access.gpo.gov/
http://ourworld.compuserve.com/homepages/ftpub/call.htm
http://www.ccic.gov/ac/interim/
http://www.cs.man.ac.uk/~kung-kiu/jsc
http://www.cs.sandia.gov/~scistra/DAM
http://www.cs.tu-berlin.de/~tolk/AAMAS-CfP.html
http://www.elsevier.nl/locate/parco
http://www.santafe.edu/~bonabeau
http://www.soc.plym.ac.uk/soc/sameer/paa.htm
http://www.wired.com/wired/5.11/es_hunt.html

Table 1 Generic data items extracted from a 4-page electronic newsletter

lookup is appropriate—but the lookup operation should recognize legitimate variants. Others will be composed of parts that have been encountered before, say *John* and *Smith*, but not in that particular combination. Others will be recognizable by format (e.g. *Randall B. Caldwell*). Still others—particularly certain foreign names—will be clearly recognizable because of peculiar language statistics (e.g. *Kung-Kui Lau*). Others will not

	computed label											
	d	*n*	*s*	*o*	*l*	*m*	*e*	*u*	*p*	*f*	*t*	
d	47		3									50
n		27	1		2							30
s			29	1			1					31
o			3	15	1							19
l		5			11							16
m						8						8
e			1				19					20
u								10				10
p									4			4
f										4		4
total	47	32	37	16	14	8	20	10	4	4	0	192

(a) Tokens in isolation

	computed label											
	d	*n*	*s*	*o*	*l*	*m*	*e*	*u*	*p*	*f*	*t*	total
d	43										7	50
n		28			1						1	30
s			26								5	31
o				14							5	19
l		1			10						5	16
m						8						8
e							19				1	20
u								10				10
p									4			4
f										4		4
total	43	29	26	14	11	8	19	10	4	4	24	192

(b) Tokens in context

Table 2 Confusion matrix: tokens are identified by compression models

be recognizable except by capitalization, which is an unreliable guide—particularly when only one name is present.

3 Using language models to recognize tokens

Character-based language models provide a good way to recognize lexical tokens. Tokens can be compressed using models derived from different training data, and classified according to which one supports the most economical representation. To test this, several issues of the same newsletter used for Table 1 were analyzed manually to extract all 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. Various experiments were carried out to determine the power of language models to discriminate these tokens, both out of context and within their context in the newsletter. Throughout this work, we use a PPM text compression scheme (Cleary and Witten, 1984), with order 5 (except where otherwise mentioned), escape method D (Howard, 1993), and a further improvement for deterministic scaling (Teahan, 1997).

3.1 Discriminating isolated tokens

Lists of names, dates, locations, etc. in 19 issues of the newsletter were input to PPM separately to form ten compression models labeled *n*, *d*, *l*, *s*, *o*, *u*, *e*, *p*, *f*, *m*. In addition, a plain text model, *t*, was formed from the full text of all 19 issues. These models were used to identify each of the tokens in Table 1 on the basis of which model compresses them the most. The results are summarized in the form of a confusion matrix in Table 2a, where the rows represent the correct label and the columns the computed one. Notice that although *t* never appears as the correct label, it could be assigned to a token of a different type because it compresses it best.

Of the 192 tokens in Table 1, 174 are identified correctly and 18 incorrectly. In fact, 69 of them appear in the training data (with the same label) and 123 are new; all of the errors are 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 still identified correctly. The 18 errors are easily explained; some are quite understandable.

Beaverton , a location, was mis-identified as a name. Compressed as a name, it occupies 3.18 versus 3.25 bits/char as a location. *Norman* (!), *Cleveland*, *Britain* and *Quebec* were also identified as names. Conversely, the name *Mark Sanford* was mis-

identified as a location. Although *Mark* appears in five different names, this is outweighed by eighteen appearances of *San* in locations (e.g. *San Jose*, *San Mateo*, *Santa Monica*, as well as *Sankte Augustin*). The name *Sorin C. Istrail* was also identified as a location, as was the organization *Fraunhofer CRCG*.

ACM, an organization, was mis-identified as a source. In fact, the only place these letters appear in the training data is in *ACM Washington Update*, which is a source. Sources are the most diverse category and swallow up many foreign items: the dates *eight-week*, *eight years* and *Spring 2000* (note: *comp.software.year-2000* is a source); the name *Adnan Amin*; organizations *Commerce Business Daily (CBD)* and *PAA*; and the email address *koza@genetic-programming.org* (because *genetic-programming* appears in the training data as both a source and plain text). Some sources were also mis-identified: *ECOLOG-L* was identified as an organization, and *sci.nanotech* as an email address.

3.2 Distinguishing tokens in context

Our quest is to identify tokens in the newsletter. Here, contextual information is available which provides additional help in disambiguating them. However, identification must be done conservatively, so that strings of plain text are not misinterpreted as tokens—and since there are many strings of plain text, there are countless opportunities for error.

For example, email addresses in the newsletter are always flanked by angle brackets. Many sources are preceded by a [or ,• and followed by a ,•. (Bullets are used to make spaces visible.) These contextual clues often help in identifying tokens. Conversely, identification may be foiled in some cases by misleading context. For example, some names are preceded by *Rep.*•, which reduces the weight of the capitalization evidence because capitalization routinely occurs following a period. But by far the most influential effect of context is that to mark up any string as a token requires the insertion of two extra symbols: *begin-token* and *end-token*. Unless the compression models are good, tokens will be misread as plain text to avoid the overhead of these extra symbols.

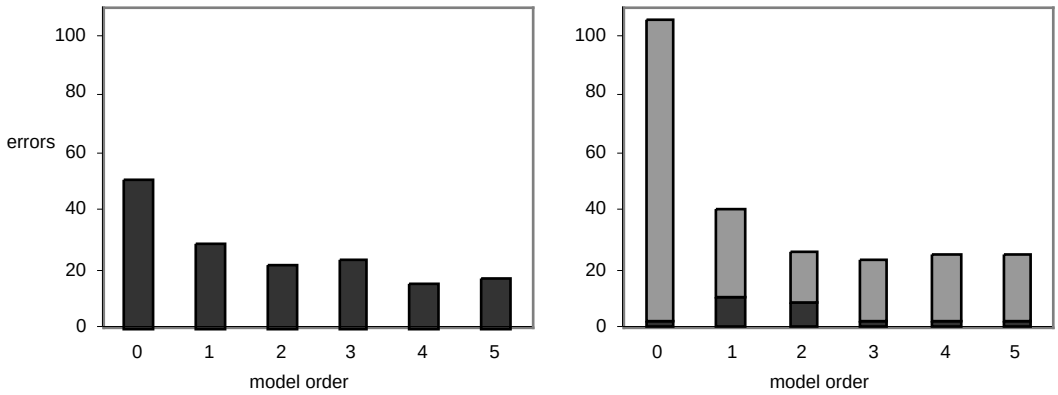
In order to evaluate the effect of context, all tokens were replaced by a symbol that was treated by PPM as a single character. The training data used for the plain-text model was transformed in this way, a new model was generated from it, and the text article was compressed by this model to give a baseline entropy figure of e_0 bits. The first token in Table 1 was restored into the test article as plain text and the result recompressed to give entropy e bits. The net space saved by recognizing this token as belonging to model m is

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

where e_m is the entropy of the token with respect to model m. This was evaluated for each model to determine which one classified the token best, or whether it was best left as plain text. The entire procedure was repeated for each token individually.

Table 2b shows the confusion matrix that was generated. The number of errors has increased from 18 to 26; however, 24 of these "errors" are caused by failure to recognize a token as being different from plain text, and only two are actual mis-recognitions (*Berkeley* is identified as a name and *Mark Sanford* as a location).

There is a small overall improvement in compression through the use of tokens. To code the original test file using a model generated from the original training files takes a total of 28,589 bits for 10,889 characters, an average of 2.63 bits/char. To code the marked-up test file using the appropriate models generated from the marked-up training files takes 841 fewer bits—despite the fact that there are 364 additional *begin-token* and *end-token* symbols). This represents a 2.9% improvement for the file as a whole, or 0.077 bits per character of the original file. Of course, the improvement is diluted by the presence of a large volume of plain text. When the savings are averaged over the affected



(a) Isolated tokens (b) Tokens in context

Figure 3 Effect of model order on token identification (light areas represent tokens mis-identified as plain text)

characters—namely the 2945 characters that are present in the tokens alone—the improvement seems more impressive, 0.28 bits/char.

3.3 Effect of model order

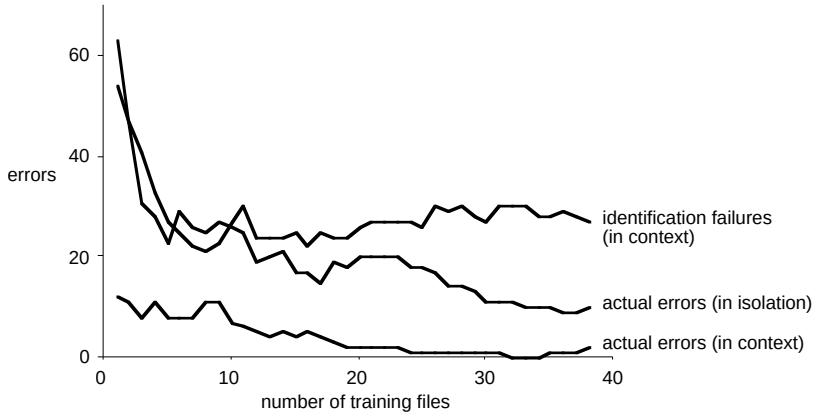
To investigate the effect of model order, the token discrimination exercise was re-run using PPM models of order 0, 1, 2, 3, 4, and 5, using the same training and test data. Figure 3 shows the number of errors in token identification observed, for both isolated tokens and tokens in context. The dark bars show actual errors, and the light ones indicate failure to identify a token as something other than plain text. The number of actual errors is very small when tokens are taken in context, even for low-order models. However, the number of identification failures is enormous when the models are poor ones.

Although the lowest error rates are observed with the highest-order models, the improvement as model order increases beyond 2 is marginal. Note that we have used models of the same order for all information items (including the text model itself); better results may be obtained by choosing the optimal order for each token type individually. For example, order-0 models identify every single sum of money, email address, URL, phone and fax number, with no errors—even in context.

The tradeoff, with tokens in context, between actual errors and failures to identify is not a fixed one. It 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 us to control the error rate, sacrificing a small increase in the number of errors for a larger decrease in identification failures.

3.4 Effect of quantity of training data

Next we examine how identification accuracy depends on the amount of training data. We varied the training data from one to 38 issues of the same newsletter, marked up manually. Figure 3 plots the number of errors in token identification, for the same test file, against the amount of training data. The middle line corresponds to isolated tokens (as in Figure 3a). The other two correspond to tokens in context, the lower one to the number of actual errors made (as in the dark bars of Figure 3b), and the upper one to the number of failures to identify a token as anything but plain text (the light bars). Notice that there is a very satisfactory reduction in errors as the amount of training data increases, but the number of identification failures stabilizes at an approximately constant level. Our choice of 19 training documents for all other tests seems to be a sensible one.



4 Locating tokens in context

We locate tokens in context by considering the input as an interleaved string of information from different sources. This model has been studied by Reif and Storer (1997), who consider optimal lossless compression of non-stationary sources produced by concatenating finite strings from different sources. However, they assume that the individual strings are long, and grow without bound as the input increases; this assumption allows an encoding method to be derived with asymptotically optimal expected length. Volf and Willems (1997) study the combination of two universal coding algorithms using a switching method. They devise a dynamic programming structure to control switching, but rather than computing the probability of a single transition sequence, they weight over all transition sequences.

Our work derives from Teahan *et al*.'s (1997) for correcting English text using PPM models. Suppose every token is bracketed by *begin-token* and *end-token* symbols; the problem then is to "correct" text by inserting such symbols appropriately. *Begin-* and *end-token* will identify the type of the token in question—thus we have *begin-name-token*, *end-name-token*, etc, written as *<n>*, *</n>*. The innovation in the present paper is that whenever a *begin-token* symbol is encountered, the encoder switches to the compression model appropriate to that type of token, initialized to a null prior context. And whenever *end-token* symbol is encountered, the encoder reverts to the plain text model that was in effect before, replacing the token by the single symbol representing that token type.

4.1 Algorithm for model assignment

Our algorithm takes a string of text and works out the optimal sequence of models that would produce it, along with their placement. As a small example, the input string

In•1998,•$2.

produces the output

<t>In•<d>1998</d>,•<m>$2</m>.</t>

Here, *t* is a model formed from all the training text with every token replaced by a single-symbol code. The characters *1998* have been recognized as a date token, because the date model compresses them best; *$2* has been recognized as a money token, because the money model compresses them best. The remainder has been recognized as plain text.

The algorithm works by processing the input characters to build up 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. For example, the character that follows

<t>In•<d>1998</d>

will be predicted by the four characters *In•<d/d>*, where *<d/d>* is the single-symbol code representing the occurrence of a date. This context is interpreted in the *t* model.

What causes the tree to branch is the insertion of *begin-token* symbols for each possible token type, and the *end-token* symbol for the currently active token type (in order 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. Currently, three separate pruning operations are applied that remove leaves from the list and therefore prevent the corresponding paths from growing further. First, if two leaves are labeled with the same character and have the same preceding *k* characters, the one with the greater entropy is deleted. Here, *k* is the model order (default 5). This would be a "safe" pruning criterion that could not possibly eliminate any path that might turn out to be the best one, were it not for the fact that a different sequence of unterminated models might exist in the two paths, some of which, when terminated, might cause the contexts to differ between the paths. At present, we do not check for this eventuality. The second pruning operation is to delete any leaf that (a) has a larger entropy than the best path so far, and (b) represents a point that lags more than *k* symbols behind that best path. This pruning heuristic is not guaranteed to be safe: it may delete a path that would ultimately turn out to be best. The reason is that the price in bits to enter any model is unbounded; yet so is the benefit that may eventually accrue from using the model. The third is to restrict the open-leaves list to a predetermined length.

These pruning strategies cause a small number of identification errors (discussed below); other strategies are under investigation.

4.2 Results when locating tokens in context

To evaluate the procedure for locating tokens in context, we used the training data from the same 19 issues of the newsletter that were used previously, and the same single issue for testing. Error counting is complicated by the presence of multiple errors on the same token. Counting all errors on the same token as one, there are a total of 47 errors:

2	identification errors noted in Section 3.2 for in-context discrimination
24	failures to recognize a token as being different from plain text
5	incorrect positive identifications
9	boundary errors
3	phone/fax absorption errors
4	pruning errors

In addition, a further 9 "errors" occurred which were actually errors in the original markup, made by the person who marked up the test data.

The two identification errors and 24 failures to recognize are those noted in the confusion matrix of Table 2b. The five incorrect positive identifications picked out *bonus* as a date, *son* and *Prophet* as names, *field as* and *Ida* as organizations. Boundary errors are more interesting. Many involved names: *Wilson*, *Kung-Kiu*, *Lashon* and *Sorin C* were identified by the algorithm as names whereas it was *Heather Wilson* , *Kung-Kiu Lau* , *Lashon Booker* and *Sorin C. Istrail* that were marked up. Similar mistakes occurred with other token types: *year* for *twelve-year*, *CBD* for *CBDNet* , *International Series* and *Computational Intelligence* (separately) for *International Series on Computational Intelligence*, *koza* and *.org* (separately) for *koza@genetic-programming.org* , *J. of Symbolic Computation* and "*JSC)*" separately for *J. of Symbolic Computation (JSC)* , and *comp.ai* for *comp.ai.genetic*. Phone/fax absorption errors are a special case of boundary errors involving phone and fax numbers: for example, *+44-1752-232 540* was identified as a phone number whereas it was *+44-1752-232 540 fax* that was marked up, as a fax number. Finally, pruning errors are caused by the pruning strategy described in the previous section.

5 Using language models to recognize structures

So far we have not taken advantage of the potentially hierarchical nature of the tokens that are inferred. We use the term "soft parsing" to denote inference of what is effectively a grammar from example strings, using exactly the same compression methodology. After analyzing the errors noted above, we refined the markup of the training documents to use *forename*, *initial*, *surname* for names; *username*, *domain*, and *top-level domain* for email addresses; and embedded phone numbers for fax numbers. Examples are

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

(Of course, different symbols were used for *f*, *s*, *u*, *d*, and *t* to avoid confusion with the other models.) Then, during training, models are built for each component of a structured item, as well as the item itself, and when the test file is processed to locate tokens in context, these new tags are inserted into it too. The algorithm described above accommodates nested tokens without any modification at all.

The results were mixed. Some errors were corrected (e.g. *Kung-Kiu Lau* and *Sorin C. Istrail* was correctly marked), but other problems remained (e.g. the fax/phone number mix-up) and a few new ones were introduced. Some of these are caused by the pruning strategies used; others are due to insufficient training data.

Despite these inconclusive initial results, we believe that soft parsing will be a valuable technique in situations with stronger hierarchical context (e.g. references and tables).

6 Conclusions

In this paper we have, through an extended example, argued the case that compression techniques are valuable for text mining. Different kinds of tokens in text can be classified because different models compress them better. Good results are achieved for tokens in isolation. Taking each token's context into account reduces the error rate at the expense of an increase in the number of symbols that are mistaken for plain text—a tradeoff that is adjustable. The dynamic programming method allows this technique to be used to identify tokens in running text with no clues as to where they begin and end. The methodology works with hierarchically-defined tokens, where each token can contain subtokens. No explicit programming is required for token identification: rather, machine learning methodology is used to acquire identification information automatically from a

marked-up set of training documents. The result is not just improved compression for semi-structured documents, but automatic location and classification of the items that they contain.

Applications of compression-based text mining are legion; we have just begun to scratch the surface. Identifying references in documents; locating information in tables (such as stock prices) expressed in either html or plain text, inferring document structure, finding names, addresses, phone numbers on Web pages, data detectors of any kind—all of these could be accomplished without any explicit programming.

Great power is gained from the hierarchical nature of the representation. A reference, for example, contains tokens that represent names, year of publication, title, journal, volume, issue number, page numbers, month of publication. These tokens will be separated by short fillers involving spaces, punctuation, quotation marks, the word "and", etc. The fillers will be quite regular, and although the tokens that appear, and the order in which they appear, will vary somewhat, the number of different possibilities is not large.

In order to investigate the application of compression to text mining in a constrained context, the experiments reported here have been self-contained. All training and testing has taken place on issues of a particular electronic magazine—we have eschewed the use of any additional information. However, in practice, it is easy and very attractive to prime models from external sources—lists of names, organizations, geographical locations, information sources, even randomly-generated dates, sums of money, phone numbers. Priming will greatly reduce the volume of training data that needs to be marked up manually, making text mining practical even with small amounts of training data.

Acknowledgments

We are very grateful to Stuart Inglis and John Cleary who provided valuable advice and assistance.

References

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.

Garside, R., Leech, G. and Sampson, G. (1987) *The computational analysis of English: a corpus-based approach*. Longman, London.

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

Lenat, D.B. (1995) "CYC: A large-scale investment in knowledge infrastructure." *Comm ACM*, Vol. 38, No. 11, pp. 32–38.

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

Reif, J.H. and Storer, J.A. (1997) "Optimal lossless compression of a class of dynamic sources." *Proc Data Compression Conference*, edited by J.A. Storer and J.H. Reif. IEEE Computer Society Press, Los Alamitos, CA, pp. 501–510.

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

Teahan, W.J., Inglis, S., Cleary, J.G. and Holmes, G. (1997) "Correcting English text using PPM models." *Proc Data Compression Conference*, edited by J.A. Storer and J.H. Reif. IEEE Computer Society Press, Los Alamitos, CA, pp. 289–298.

Volf, P.A.J. and Willems, F.M.J. (1997) "Switching between two universal source coding algorithms." *Proc Data Compression Conference*, edited by J.A. Storer and J.H. Reif. IEEE Computer Society Press, Los Alamitos, CA, pp. 491–500.