Using compression-based language models for text categorization

William J. Teahan and David J. Harper
School of Computer and Mathematical Sciences
The Robert Gordon University, Aberdeen AB25 1HG, Scotland, UK
{wjt,djh}@scms.rgu.ac.uk

1. BACKGROUND

Text categorization is the problem of assigning text to any of a set of pre-specified categories. It is useful in indexing documents for later retrieval, as a stage in natural language processing systems, for content analysis, and in many other roles [7].

The traditional machine learning approach to text categorization is based on a four step process [4]: first, performing word and sentence segmentation on the training files; second, performing feature selection on the word counts; third, applying a machine learning algorithm; and last, applying the learned model to the test data produced after performing the same feature selection process that was applied in step two. A wide variety of machine learning algorithms have been applied based on this approach, including: probabilistic Bayesian classifiers, decision trees, neural networks, multivariate regression models, symbolic rule learning and support vector machines [4].

Traditional approaches to text categorization share several problems: the need to perform feature extraction before processing; the need to define where word boundaries occur; and the need to deal uniformly with morphological variants of words [5]. The problem of word and sentence segmentation is an especially vexing one, and indeed for some languages, the Western notion of a word may be inappropriate [13].

We wish to use language models developed for text compression as the basis of a text categorization scheme and potentially for other applications in IR (although the latter will not be discussed here). The motivation for using these models is that they are well grounded in information theory, and that they also have proven performance at many IR-related tasks such as word segmentation [13], text mining [14], language/dialect identification and authorship ascription [12].

For text categorization, our compression-based language modeling approach has several advantages. By using models that are character-based, not word-based, we can avoid the segmentation issue altogether. Additionally, we can sidestep to some extent the issue of which features to use by examining potential features within the framework of standard Markov-based approximations commonly used in language modeling.

Anecdotal evidence indicates that the idea of using compression for text categorization has been re-invented many times (Frank *et al.* [5]). However, compression-based approaches to date have produced inferior results compared to state-of-the-art schemes based on traditional machine learning techniques more reliant on feature extraction [5] [12]. Indeed, Frank *et al.* state that it is their belief that compression-based approaches will not be capable of competing with the machine learning approaches.

In this paper, we seek to allay this mis-apprehension. We show how a compression-based approach can in fact produce competitive results compared with the traditional machine learning approaches, and in some cases outperform them. The next two sections gives the theoretical background to our approach showing its basis in information theory and Bayesian classification. Section 4 briefly describes the particular text compression scheme (PPM) that we use in our experiments with the results discussed in Section 5. Further discussion is provided in the final section.

2. COMPRESSION MODELS

As stated, we wish to use compression-based language models as the basis of a text categorization scheme. The crux of our reasoning hinges on the notion of *entropy* as a measure of the "information content" of a message [10], and importantly that text compression can be used directly to estimate an upper bound to it [2]. Various treatments of entropy are given in [10] [2] [8] and these are drawn upon in the following discussion.

Formally, suppose we have a language defined by an alphabet of k symbols s_i, and a probability distribution over these symbols $p(s)$. The fundamental coding theorem [10] states that the lower bound on the average number of bits per symbol needed to encode messages of the language (i.e. sequences of the symbols) is given by the entropy of the probability distribution (and often referred to as the "entropy of the language"):

$$H(\text{language}) = H(p) = -\sum_{i=1}^{k} p(s_i)\log_2 p(s_i). \qquad (1)$$

The information content of a particular symbol is given by $-\log_2 p(s_i)$, and is the number of bits required to encode that symbol. Thus, $H(p)$ can be viewed as an expectation over the number of bits per symbol. Hence, we can obtain minimum code lengths, or alternatively maximal compression rates, by encoding symbols in accordance with the actual probability distribution $p(s)$.

Generally, we do not have access to the actual probability distribution underlying a language, but rather some model of that language. Suppose we have a language, L, in which sequences of symbols $x_1, x_2, \ldots, x_n$ (denoted x_{1n} hereafter) are generated according to probability distribution $p(x)$, for which we have constructed a model $p_M(x)$. The *cross-entropy* of the language with respect to the model provides a measure of the extent to which the model departs from the actual distribution, and is given by:

$$H(L, p_M) = -\lim_{n\to\infty}\frac{1}{n}\sum_{x_{1n}} p(x_{1n})\log_2 p_M(x_{1n}). \qquad (2)$$

The cross-entropy is the average number of bits per symbol required to encode language L using the model p_M, and thus $H(L, p_M)$ is an upper bound on the entropy of the language, and $H(L) \leq H(L, p_M)$. $H(L, p_M)$ can be approximated by observing that Equation 2 is simply the expected value of $-\log_2 p_M(x_{1n})$ over an infinite sample of language examples, and we approximate it using one very large sample as follows:

$$H(L, p_M) \approx -\frac{1}{n} \log_2 p_M(x_{1n}), \quad n \text{ very large.} \tag{3}$$

For a given document, D, we can compute the *document cross-entropy*, the average number of bits per symbol to encode the document using the model as:

$$H(L, p_M, D) = -\frac{1}{n} \log_2 p_M(D), \quad D = x_{1n}. \tag{4}$$

Suppose that p_M is a good model for L, then the lower the document cross-entropy, the more likely it is that the document D is generated by the language. This key observation lies at the heart of using text compression for text categorization. Categories are viewed as languages, and documents are compressed using models constructed over the categories.

3. BAYES CLASSIFIERS

We would like to establish the relationship between Bayes classifiers, and using cross-entropy for categorization, and in that context discuss feature selection for compression models.

Suppose we want to classify documents such that they are assigned independently to one or more categories. Assuming we have a training collection of categorized documents, then for any given category, C, we can construct a model for that category p_C, and model for its complement p_N (hereafter the "negative" model). Bayes formula gives us a mechanism for deciding whether a particular document, D, belongs to category C, and using the more familiar notation $P(D|C) = p_C(D)$ and $P(D|\overline{C}) = p_N(D)$, decide D belongs to C if:

$$P(D|C)P(C) > P(D|\overline{C})P(\overline{C}) \tag{5}$$

where $P(C)$ is the prior probability of a document belonging to C. Equivalently, the documents can be ranked with respect to the category using the following expression:

$$\log_2 \frac{P(D|C)}{P(D|\overline{C})} \tag{6}$$

and a cutoff is selected, which gives optimal categorization, according to some measure of categorization goodness.

Equation 6 has a dual formulation in the cross-entropy and compression world, namely

$$\begin{aligned}
&\log_2(P(D|C)/P(D|\overline{C})) \\
&\quad \equiv -\log_2 P(D|\overline{C}) - (-\log_2 P(D|C)) \\
&\quad \equiv -\log_2 p_N(D) - (-\log_2 p_C(D)) \\
&\quad \equiv H(\overline{C}, p_N, D) - H(C, p_C, D).
\end{aligned} \tag{7}$$

The inverse relationship between cross-entropy, and probability of a given document, is well-known. Thus, maximizing the code length difference between a document encoded using the negative model and category model respectively, is equivalent to constructing an independent Bayes classifier for each category. Frank *et al* provided a similar analysis in [5]. In applying text compression to categorization, we will determine an optimal cutoff for the code length difference computed in Equation 7 on a category-by-category basis, which will be referred to as the *category cutoff*.

More interestingly, this relationship suggests ways in which feature selection might be incorporated within text compression models when applied to categorization. The need for feature selection was highlighted in [5], although they did not propose doing this within a compression framework. Features are selected for Bayes classification on the basis of their discrimination power, which is computed based on the relative density of the feature in the category compared with its density in the complement (or similar). Likewise, we propose selecting features that exceed some code length difference threshold. Hence, features will be ranked based on their potential contribution to the overall code length difference (as given by Equation 7). Again, we will apply feature selection on an individual category basis, and the resultant cutoff will be referred to as the *feature cutoff*.

A benefit of performing thresholding (i.e. category and feature cutoff in our case) noted in [5] is that it automatically adjusts for the fact that the models p_C and p_N are formed using different amounts of training data. This is a potential problem since in general, one expects to achieve better compression with more training data.

4. PPM-BASED LANGUAGE MODELS

The Prediction by Partial Matching (PPM) text compression scheme has consistently set the standard in lossless compression of text since it was originally published in 1984 by Cleary and Witten [3]. Other methods such as those based on Ziv-Lempel coding are more commonly used in practice, but their attractiveness lies in their relative speed rather than any superiority in compression [1].

Compression models encode a document according to a given model, and the resultant cross-entropy is given by Equation 4. In the case of the PPM family of models, the individual symbols are encoded within the context provided by the preceding symbols appearing in a document. Taking Equation 4 as a starting point, we can achieve optimal compression for the model p_M, by observing that:

$$\begin{aligned}
&H(L, p_M, D) \\
&\quad = -\tfrac{1}{n} \log_2 p_M(D), \quad D = x_{1n} \\
&\quad = -\tfrac{1}{n} \log_2 \textstyle\prod_{i=1}^{n} p_M(x_i|context_i) \quad \text{[by Chain Rule]} \\
&\quad = \tfrac{1}{n} \textstyle\sum_{i=1}^{n} -\log_2 p_M(x_i|context_i)
\end{aligned}$$

where $context_i = x_1, x_2, \ldots x_{i-1}$. Thus each symbol is encoded according to its *information content* within the context provided by all the preceding symbols. In practice, PPM uses a Markov approximation and assumes a fixed order context.

PPM uses an approximate blending technique called *exclusion* based on the *escape* mechanism to exclude lower order predictions from the final probability estimate. This includes an order 0 model which predicts symbols based on their unconditioned probabilities, and a default model which ensures that a finite probability (however small) is assigned to all possible symbols. Prediction probabilities for each context in the model are calculated from frequency counts, and the symbol that actually occurs is encoded relative to its predicted distribution. The maximum context length is a fixed constant, and it has been found that increasing it beyond about five does not generally improve compression [3] [11].

The PPM method is character-based, although the blending mechanism can equally be applied to other classes of symbols such as words and parts of speech. Compression experiments with a wide range of English text [11] show that word-based models consistently outperform the character-based methods, although the difference in performance is only a few percent. A fuller description of the PPM algorithm can be found in [1]; two variants of the algorithm that perform well in compression experiments are PPMC [9] and PPMD [6].

For text categorization purposes, when we apply feature selection using PPM models, we determine on a category-by-category basis an optimal cutoff by only selecting features that exceed some code length difference threshold $h_N - h_C$ where the feature code lengths $h_N = -log_2 p_N(x_i|context_i)$ and $h_C = -log_2 p_C(x_i|context_i)$.

5. EXPERIMENTAL RESULTS

The experiments described here all use the Reuters-21578 collection. (This collection is publicly available at `http://www.research.att.com/~lewis/reuters21578.html`). Using the ModApte split, the collection is separated into a training set, testing set and an unused set (9603, 3299 and 7634 documents respectively). This collection comes with manually assigned categories for all the documents in the training and testing sets.

For the experiments, we have further split the training set into a training subset and a validation subset (6338 and 3266 documents respectively). We use the validation subset for choosing the category and feature cutoff values that maximizes the average of recall and precision for each category based on the code length difference ranking formula defined in Equation 7. Once the cutoff values have been obtained, the models p_C and p_N are rebuilt to take maximum use of the full training data in the same manner as [5].

Although there are 118 categories in the entire Rueters collection, our experiments only concentrate on the ten largest categories (ordered by the size of training data available, in bytes): *earn, acq, money-fx, trade, crude, grain, interest, wheat, ship* and *corn.* This cuts the validation subset and testing set down to 2152 and 3019 documents respectively. Table 1 lists the number of training and testing documents available for each category; also listed is the total size of the training data in bytes after concatenating all the training files together once irrelevant text (including tags) has been removed. The average size per document is also shown in the last column. The figures show that the distribution of the collection is highly skewed—there is a varying amount of training data available from nearly 1.5 Mb for the most frequent category *earn* down to just over 200 Kb for the smallest category *corn*; the size of the documents also varies noticeably, with just over 500 bytes for *earn*, to over three times that for *trade.*

Table 2 shows the average precision/recall values that are achieved on the Reuters collection at the testing stage when using the category and feature cutoffs obtained from the validation stage. The italic values indicate the orders that performed the best at the validation stage. Shown at the bottom of the table is the average of the top ten categories weighted according to their frequency of occurrence in the testing set. From Table 2, we can see that no one context order is particularly predominate at achieving the best average precision/recall, with the best values scattered across all orders. These results show that each category has features that make

Category	Number Training Documents	Number Testing Documents	Training size (bytes)	Avg. size of document (bytes)
earn	2877	1087	1488519	517.4
acq	1650	719	1330793	806.5
money-fx	538	179	623599	1159.1
trade	369	118	579607	1570.8
crude	389	189	532919	1370.0
grain	433	149	490372	1132.5
interest	347	131	335260	966.2
wheat	212	71	239843	1131.3
ship	197	89	218093	1107.1
corn	182	56	217246	1193.7

Table 1: Various statistics for the Reuters collection.

a particular context order most effective for it. This is not too surprising when we consider the language used in some of these categories. Considering *earn, acq, money-fx* and *trade*, these categories quite often use acronyms that are only two or three characters long, or contain numerical data, and therefore are best served with models that have a context order different from the order 5 models found most effective for general English text. Also, higher orders (4 to 6) feature strongly, which closely match well-documented results from standard compression experiments on text-like data [1].

Table 3 compares the order 3 model results from Table 2 (reproduced in the last column) with results published in [4] for Naïve Bayes and linear support vector machines (LSVMs), and with previously published results for PPM. Results in the last two columns are new results using our models. (The best result for each category is shown in bold font). The results show that our approach is uniformly better than Naïve Bayes (except for *ship*), and performs better than LSVMs in three out of the ten categories (*acq*, *money-fx* and *trade*) and competitively in a further four categories (*crude*, *earn*, *grain* and *interest*). Lack of training data clearly affects performance for the three poorest performing categories, *corn*, *ship* and *wheat*, which correspond to the three smallest categories from Table 1.

Shown in the third column of Table 3 are the results reported by Frank *et al.* who used order 2 PPMC models [5]. In their method, they used a single category cutoff for optimizing precision/recall; this is clearly improved across nearly all categories using the order 3 models and the feature cutoff approach that we adopt. They also found that models higher than order 2 achieved degraded performance in almost all cases. They attributed this to the amount of training data being insufficient to justify more complex models. This is in direct contrast to the results that we have obtained—we have found that higher order models perform as well (and in some cases better). Unfortunately, Frank *et al.* did not include results for higher order models in their paper, so it is difficult to speculate as to the reasons for the variance between our results and theirs.

Results for order 3 PPMD models without feature cutoff are shown in the second to last column of Table 3. They show a consistent but slight improvement across all categories when using feature cutoff, except for *corn* (a very slight decrease) and *wheat.* Results for other orders show that feature cutoff leads to better performance for two categories in particular, *interest* and *money-fx*, but to inconsistent results for other categories (sometimes better and sometimes worse than when just applying category cutoff without feature cutoff). However, these results are encouraging, and show that the approach is worthy of further investigation.

Category	Order 2	Order 3	Order 4	Order 5	Order 6
earn	0.964	0.970	0.967	0.947	*0.953*
acq	0.942	0.954	0.959	*0.954*	0.953
money-fx	0.838	0.835	*0.816*	0.834	0.810
trade	0.776	0.772	0.759	0.778	*0.772*
crude	0.837	*0.888*	0.878	0.867	0.857
grain	0.870	0.905	0.915	*0.899*	0.911
interest	0.749	0.735	0.723	0.728	*0.759*
wheat	0.721	0.717	0.706	*0.735*	0.757
ship	0.757	*0.783*	0.750	0.771	0.765
corn	0.635	0.666	0.668	0.669	*0.646*
Weighted average	0.899	0.910	0.907	0.900	*0.901*

Table 2: Average recall/precision values for the Reuters collection using PPMD models with feature cutoff.

Category	Naïve Bayes	LSVM	PPMC (Order 2)	PPMD (Order 3)	
				without feature cutoff	*with feature cutoff*
earn	0.959	**0.980**	0.963	0.968	0.970
acq	0.878	0.936	0.910	0.950	**0.954**
money-fx	0.566	0.745	0.763	0.831	**0.835**
trade	0.639	0.759	0.650	0.734	**0.772**
crude	0.795	**0.889**	0.807	0.871	0.888
grain	0.788	**0.946**	0.746	0.883	0.905
interest	0.649	**0.777**	0.604	0.685	0.735
wheat	0.697	**0.918**	0.649	0.744	0.717
ship	0.854	**0.856**	0.819	0.749	0.783
corn	0.653	**0.903**	0.542	0.667	0.666
Weighted average	0.848	**0.919**	0.863	0.902	0.910

Table 3: Average recall/precision values for various schemes on the Reuters collection.

To investigate the effect of applying feature cutoff further, we analyzed the percentage of occurrences of the symbol (i.e. character) code length differences that were observed during coding of the relevant (category) and non-relevant (non-category) documents in the testing set. Table 4 summarizes the results for the two categories *interest* and *wheat* (these were chosen because they had the biggest positive and negative difference in values between the last two columns shown in Table 3). The columns in the table correspond to the category model's code length ranges ($h_C < 1, 1 \leq h_C < 2, \ldots$), and the rows to the negative model's code length ranges ($h_N < 1, 1 \leq h_N < 2, \ldots$) where h_C and h_N are the character code lengths (in bits) obtained when coding using the category and negative models respectively. Similar results were found for all categories, and not just the two shown in the table.

The results highlighted in bold and italics font in the table show a marked difference between the relevant and non-relevant documents in the observed code lengths. The results in italics are when the category models are performing very well (i.e. $h_C < 1$)—as shown, this occurs much more frequently for the relevant documents. The figures in bold font show a similar contrast—these occur when the category models are performing poorly (h_C is high) and the negative models are doing relatively well (h_N is low). This is likely to occur due to lack of training when a feature is unobserved in the relevant training documents but is common in the non-relevant ones (such as a sequence of text that occurs frequently in standard English, say). These results are likely to have a significant impact on the overall decision, even though the number of occurrences of these cases ranges between 5 and 10% in total. This is because these code length values are high, whereas for a substantial proportion of the values, the difference in code lengths is small (i.e. the table shows that 30 to 40% of the values occur when both h_C and h_N are < 1). Of particular interest is the over one per cent difference in percentage for *interest* observed for the case when $h_N \leq 8$ and $h_N < 1$. In general, we found that the feature cutoff values chosen for each category varied between -4 and -8 (i.e. when the category model was performing very poorly compared to the negative model). The primary effect of this, as evidenced by the results in the table, is to discard features which are "suspiciously" rare in the non-relevant documents.

6. DISCUSSION

In this paper, we have presented the results of a preliminary study in the use of text compression models for text categorization. Compression models are firmly grounded in information theory, and we have exploited this theoretical underpinning in applying text compression to categorization. We have demonstrated that our compression-based approach, namely maximization of the recall/precision average based on the code length difference, is equivalent to a two class (category, complement of category) Bayes classification.

We have argued that the PPM-family of compression models, and in particular the character-based variants, have a number of advantages over word-based approaches. Problems of text segmentation (determining word boundaries), normalization such as stemming, and to a lesser extent word sense disambiguation, can be largely avoided. Instead, the PPM model discovers the important character contexts, which capture word boundary, morphology and word sense. Furthermore, we can apply our approach to languages where word segmentation is far more problematic. And, models can be constructed over very diverse texts, which may include coded and numeric data.

Our preliminary experimental results indicate that, for text categorization, PPM character-context compression models perform significantly better than word-based Naïve Bayes

		Relevant Documents					*Non-relevant documents*				
		(Percentage of occurrences)					(Percentage of occurrences)				
	$h_C \rightarrow$	< 1	1–2	2–4	4–8	≥ 8	< 1	1–2	2–4	4–8	≥ 8
	< 1	*43.91*	3.94	2.98	**0.91**	**0.07**	*33.95*	6.33	7.43	**5.01**	**1.25**
	< 2	*3.96*	4.23	3.01	**0.95**	**0.06**	*1.67*	3.11	3.90	**2.30**	**0.45**
interest	< 4	*2.26*	3.13	3.01	4.53	0.29	*0.68*	1.85	3.90	7.16	0.95
	< 8	*0.67*	0.75	3.01	9.03	1.18	*0.10*	0.30	3.90	9.30	2.15
	≥ 8	*0.06*	0.05	3.01	0.73	0.76	*0.00*	0.01	3.90	0.84	1.12
	< 1	*37.56*	4.09	3.07	**1.05**	**0.07**	*32.15*	7.52	8.08	**5.63**	**0.88**
	< 2	*3.50*	3.10	2.84	**0.95**	**0.05**	*1.59*	3.05	4.05	**2.32**	**0.49**
wheat	< 4	*2.28*	3.01	2.84	4.82	0.37	*0.54*	1.69	4.05	7.06	0.84
	< 8	*1.16*	1.24	2.84	11.83	1.79	*0.10*	0.31	4.05	9.15	1.84
	≤ 2	*0.06*	0.10	2.84	1.76	1.71	*0.00*	0.01	4.05	0.90	0.97
	$h_N \uparrow$										

Table 4: Percentage of occurrences within various ranges for symbol code lengths obtained when coding documents for the *interest* and *wheat* categories in the Reuters testing set.

classifiers, and approach the performance of the linear vector support machine text classifiers. Unlike earlier work presented in [5], we have shown that the performance of various PPM n-order models are comparable (for $n \geq 2$), although overall order 3 models are best.

Feature selection in the context of compression models has shown small improvements (see final two columns of Table 3). We based our feature selection on selecting features which will maximize recall/precision based on the code length difference. Our analysis of the distribution of feature code lengths over relevant (category) and non-relevant (non-category) documents indicates that our thresholding strategy was effective in removing some evidently poor features, but that many features are retained with poor discrimination values. Currently, features are ranked and selected based on the "local" code length difference for an individual symbol (and context). We conjecture that ranking features by the likely "global" effect they would have on code length difference may result in improved feature selection.

Clearly, an important issue is the size of the training data, and it is possible this will remain an impediment to compression-based approaches to text categorization. However, it may be possible to use a hierarchy of models, whose purpose is to progressively select features (and hence text), for subsequent compression-based categorization. For example, one could imagine a hierarchy of models for this application comprising: English model, Reuter's model (and we have evidence to suggest such a model exits), and Category models.

The idea of using text compression models is not new. Text compression models have been used for various categorization problems: language identification, dialect identification, and authorship ascription, to name but some. Recently, they have been applied to the arguably more demanding problems of subject/genre categorization, as exemplified by the Reuter's categorization task. However, unlike [5], we remain optimistic about compression-based text categorization, both because of the ease of applying the approach, and because our results indicate performance is comparable to other well-performed approaches, and capable of being still further improved. Lastly, it seems highly likely that new applications will emerge in which the combination of text compression and text categorization, when used in tandem, will be a potent mixture, especially given the huge amounts of uncategorized text which is stored on the Internet, and on Intranets.

7. REFERENCES

[1] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text compression.* Prentice Hall, New Jersey, 1990.

[2] P. F. Brown, S. A. D. Pietra, V. J. D. Pietra, J. C. Lai, and R. L. Mercer. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1):31–40, 1992.

[3] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[4] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings International Conference on Information and Knowledge Management*, pages 148–155, 1998.

[5] E. Frank, C. Chui, and I. H. Witten. Text categorization using compression models. In *Proceedings IEEE Data Compression Conference*, Snowbird, Utah, 2000.

[6] P. G. Howard. *The design and analysis of efficient lossless data compression systems.* PhD thesis, Brown University, Providence, Rhode Island, 1993.

[7] D. D. Lewis. Guest editorial. *ACM Transactions on Information Systems*, 12(3):231, 1994.

[8] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing.* MIT Press, Cambridge, Massachusetts, 1999.

[9] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[10] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656,1948, 1948.

[11] W. J. Teahan. *Modelling English text.* PhD thesis, Waikato University, Hamilton, New Zealand, 1998.

[12] W. J. Teahan. Text classification and segmentation using minimum cross-entropy. In *Proceedings RIAO'2000*, volume 2, pages 943–961, Paris, France, April 2000.

[13] W. J. Teahan, Y. Wen, R. McNabb, and I. H. Witten. Using compression models to segment Chinese text. *Computational Linguistics*, 26(3):375–393, 2000.

[14] I. H. Witten, Z. Bray, M. Mahoui, and W. J. Teahan. Using language models for generic entity extraction. In *ICML-99 Worskhop: Machine learning in text data analysis*, 1999.