On Tag Insertion and its Complexity

Stuart Yeates and Ian H. Witten

Department of Computer Science,
University of Waikato,
Hamilton, New Zealand
{s.yeates, i.witten}@cs.waikato.ac.nz

Abstract

Many text mining operations can be recast as tag insertion problems—we illustrate several. The size of the search space of the tag insertion problem is explored, using a number of proofs and heuristics: Viterbi Search, One-Tag-at-a-Time and Automatic Tokenisation. These greatly reduce the size of the search space from approximately 10^{400} to approximately 10^{11} nodes. Properties of the SGML standard are also used to reduce the complexity of the search. We work through several examples taken from bibliographies, showing search space size and possible errors.

1 Introduction

Many text mining operations can be recast as tag insertion problems—particularly functions of identification and information extraction. A simple example is word segmentation: to locate word boundaries in natural language text in which all the words are run together [12]. Western languages such as English are always written with explicit spaces between the words, the only ambiguities being hyphenated words and contractions.[1] However, languages such as Chinese and Thai are written without spaces, which makes segmentation a significant practical problem for information retrieval. Figure 1 shows a Chinese language string and two distinct segmentations.

Word identification is a low-level lexical process—although semantic considerations are occasionally required in order to resolve subtle ambiguities. More interesting linguistically is the question of identifying parts of speech. This is also a tag insertion problem: it requires tags to be inserted around each word in a text to denote the particular syntactic role that the word plays in its surrounding context [12]. Figure 2 shows an English language string, decorated with appropriate syntactic tags.

[1] This has not always been the case. Many medieval and classical manuscripts had no explicit inter-word spaces.

Raw text	我喜欢新西兰花
Interpretation 1	我 喜欢 新西兰 花
Interpretation 2	我 喜欢 新 西兰花

Fig. 1. Two different segmentations of the same piece of text. Interpretation 1 is "I like New Zealand flowers," while interpretation 2 is "I like fresh broccoli" (example adapted from [12]).

text	. . . to be or not to be. . .
tagged text	. . . <to>to</to> <vb>be</vb> <cc>or</cc> <rb>not</rb> <to>to</to> <vb>be</vb>. . .

Fig. 2. Text with parts of speech tagged using the Penn Treebank tag set (example adapted from [10]).

Another text mining problem that can be recast as tag insertion is "generic entity extraction." For example, "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 [1]. The information extraction research community has studied such tasks and reported results at annual Message Understanding Conferences (MUC). Metadata detection in scientific articles [6] or news items [7], and acronym detection [16], are among the many other problems that can naturally be recast as tag insertion.

2 The tag insertion problem

Briefly stated, the general tag insertion problem is this:

> Given a sequence of characters $C_0 \ldots C_{n-1}$ and a set of tags $T_0 \ldots T_{t-1}$, how should the tags be inserted into the sequence, so that the tags reveal the metadata implicit in it?

Tags can be inserted by using a set of hand-crafted heuristics, as a by-product of publishing and other activities, or by a human using prior knowledge.

Tagged text, rare a decade ago, has become ubiquitous with the advent of the world wide web, and is a natural way to represent the result of many text mining tasks. Generally, some kind of SGML [3] tagging is used—typically HTML or XML. While these

last are not, strictly speaking, subsets of SGML, they do share a common subset of properties that are relevant for text mining purposes—principally the notion of "balanced" tags (like `<tag>` and `</tag>`). In fact, in this paper the basic properties common to all three markup languages are used to prune the search space.

When applied to the word segmentation problem, the statement of the tag insertion problem is modified slightly.

> Given a sequence of characters $C_0 \ldots C_{n-1}$ in which word boundaries are not explicitly represented, and two tags T_{word} and $T_{/word}$, how should tags be inserted so that each word is surrounded by a T_{word} and $T_{/word}$ pair?

Several natural assumptions can be made that greatly reduce the number of possible ways to insert tags into the sequence.

1. Tags must be balanced—there must be the same number of $T_{/word}$ as T_{word} tags.
2. In any prefix of the tagged text, the number of end tags $T_{/word}$ cannot exceed the number of begin tags T_{word}.
3. T_{word} tags (which start words) and $T_{/word}$ tags (which end words) must alternate.
4. At least one character must occur between a T_{word} and a $T_{/word}$.
5. Tags have no attributes.

Assumptions 1 and 2 are built into the SGML standard; assumptions 3, 4 and 5 are easily encoded in an SGML Document Type Definition (DTD).

Assuming that all elements of the tagged text belong to a word, and because the concept of a "word" is non-recursive, the end tags can be implicit. In other words, only the start tag is specified and the tag is considered to run until the next start tag is encountered. Note that the SGML standard accommodates implicit tagging—consider the `<p>` tag in HTML. Implicit end tags allow the $T_{/word}$ tag to be dropped entirely. In the general case, where there is more than one tag type, a tag is considered to run until either the start tag goes out of scope, another start tag of the same type is seen, or a mutually exclusive tag is encountered.

Table 1 shows how various tag insertion problems can be represented in terms of tags to be inserted and the typical lengths of the tags, in characters. The maximum length of tags has important performance implications, as we shall see (Section 3.1). The first two domains have implicit end tags (i.e. the alphabetic character before the start of the current word is the last character of the previous word in word segmentation) so the tags do not need to be included. The last two domains need explicit end tags (because documents can have text other than document metadata or acronyms), but these are not shown.

These assumptions transform the word segmentation problem into a binary search space of size 2^n (n is the sequence length), with the possibility of inserting a T_{word}

Problem	Typical tags	Typical tag length
Chinese segmentation	T_{word}	1–5 chars
Part of Speech Tagging	$T_{noun}, T_{verb}, \ldots$	3–10 chars
Metadata Extraction	$T_{title}, T_{author}, \ldots$	10–250 chars
Acronym Definition Detection	$T_{acronym}$	50–200 chars

Table 1. Typical tag insertion problems, the tags inserted to solve the problem and typical lengths of the tags.

tag between each pair of characters in the sequence. These assumptions, or very similar ones, are used throughout our work.

The tag insertion problem involves searching this 2^n space. Of course, this must be done in conjunction with an evaluation function that specifies when one putative tagging is better than another. Treating text mining as tag insertion provides a uniform interface to a range of interesting problems expressed in terms of the common subset of the widely used markup standards.

2.1 Sizes of search space for tag insertion

In the general tag insertion problem, there are

$$\sum_{r=0}^{t} \frac{t!}{(t-r)!}$$

way of selecting zero or more tags from a set of t tags, so if the sequence has n characters and tags may be inserted between any two characters, before the first and after the last character then there are approximately

$$\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^n$$

possible encodings of the sequence with a tag set of size t. This rests on Assumptions 1-5 above, and is suitable for hierarchical markup (tags may be nested within tags of different types) but not recursive markup (tags nested within tags of the same type). Full recursive markup (revocation of Assumption 3) effectively doubles t.

In terms of a search space, this is represented as a tree with a branching factor of $\sum_{r=1}^{n} \frac{t!}{(t-r)!}$ and a depth of n. This is a very large space, especially considering that real world tag sets (HTML, XML etc) commonly have $t \geq 50$ and that real world documents can have $n \geq 5,000,000$.

Indeed, searching of spaces of this order is intractable using computers as we know them.

3 Text Mining using Viterbi Search

Viterbi search is an alternative to machine learning in text mining. Rather than being grounded in statistics as machine learning is [14], Viterbi search is grounded in information theory [8], which defines the entropy of a text or message in relation to a model. By calculating the entropy of various texts in relation to PPMD [5, 15] models initialized on training text the "goodness" of the markup in the texts can assessed in relation to the markup in the training text [4]. Entropy calculations are, however, an expensive operation and need to be reduced wherever possible.

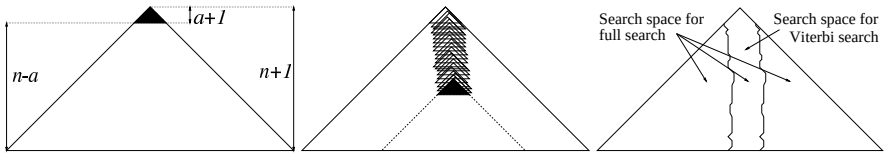
3.1 Viterbi Search

Viterbi search is a technique developed by Andrew J. Viterbi [13] for correcting errors in digital communications. An original message is encoded by an encoder with reference to a model; after transmission (and associated introduction of errors into the message), the decoder decodes the message with respect to its copy of the model.

What differentiates Viterbi search encoder/decoder pairs from any other model-based encoder/decoder pair is that the encoder introduces redundancy (increases the length of a message rather than reduces it as compression does) in such a way that the redundancy can be used by the decoder like a check-sum. The redundancy used in traditional Viterbi search is a specially constructed form of redundancy—the message is commonly compressed to remove normal redundancy before being encoded.

Viterbi search is a non-heuristic entropy-minimizing search, as shown in Figures 3 (a-d). Only a small portion (coloured black) is searched initially (a). The lowest entropy leaf is found and the root node is pruned to leave only the branch connected to the lowest entropy leaf; then, all leaves on that branch are expanded. At each step the root of the active search space is pruned leaving only the branch that is connected to the lowest entropy leaf, and all unpruned leaves are expanded (b). The dashed lines indicate the portions of the search space pruned by this step. When the search reaches the end of the sequence, only a subset (c) of the space has been searched.

Pruning the search space is possible because of a proof outlined in [13], which gives bounds on how much context need be taken into account when considering whether or not to insert a tag. When inserting a tag with a maximum length of l_{max} using a model of maximum order o_{max}, the maximum lookahead is $a = l_{max} + o_{max}$.



(a) (b) (c)

search of a large search space, with the active part of the search space at arked in black.

es the search space from:

$$\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{n}$$

nodes to:

$$(n-a)\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{a+1}$$

nodes.

The reason for the factor $n-a$ is shown graphically in Figure 3(a)—it is the number of steps in which the incremental search space is searched. This reduction makes the tag insertion problem tractable, for sufficiently low order PPMD models (typically 1–5 characters), sufficiently short maximum tag lengths (typically in the range of 5–15) and sufficiently small tag sets (typically 1–3).

The implementation described and used in [10–12] uses a variation on Viterbi search which heuristically prunes the search tree to the c lowest entropy leaves at each level. This approach fails to find optimal encodings that sacrifice a high entropy cost early in the sequence. This effect does not occur in our implementation because we calculate the maximum required look-ahead—the number of characters whose encoding can be effected by the insertion of a tag at the current point.

3.2 One Tag at a Time Search (OTT)

Further reduction in the search space can be achieved using heuristic assumptions. A heuristic we have implemented, which we call One Tag at a Time (OTT), is to perform the search in multiple passes, once for each tag, rather than simultaneously inserting all

the tags. This further reduces the search space from:

$$(n-a)\left[\sum_{r=0}^{t}\frac{t!}{(t-r)!}\right]^{a+1}$$

nodes to:

$$\sum_{i=1}^{t}(n-a+c)2^{a_t+1}$$

nodes by reducing the search space to a set of t binary trees, one per tag, which are explored sequentially using the Viterbi algorithm. c is the number of tags inserted by previous passes and is bounded by $tn^2 \geq c \geq 0$, but can be expected to be $tn \geq c \geq 0$ for reasonable tag sets and documents. a_t is the maximum lookahead for tag t and is the sum of the maximum length for that tag and the maximum order of the model.

An example of the errors this heuristic introduces is shown and discussed in Section 3.5.

3.3 Automatic Tokenisation

Automatic Tokenisation (AT) is a heuristic based on the observation that tags in text often occur in certain places in the text—names, titles and dates do not begin in the middle of words but on word boundaries. By noticing during training that no tag occurs between a pair of lowercase letters nor between a pair of digits, as is the case for all the examples mentioned in this paper, the search can be pruned by not considering inserting tags between pairs of lowercase letters or digits. If the text being encoded is in Unicode [2] or can be converted to Unicode (i.e. all web pages), then a number of character classes are defined, including letter (with the partial subsets uppercase-letter, lowercase letter), digit, punctuation and white space.

In practical terms, the Java `java.lang.Character` class provides a fast, stable, language and platform independent interface to these character classes.

The search-space reduction given by AT is difficult to model exactly. Assume that tags are only inserted either side of non-alphanumeric characters and that words average five characters with one white space character and one punctuation character between each word. Then the depth of the search tree is reduced by a factor of $\frac{3}{7}$ and the search space from:

$$\sum_{i=1}^{t}(n-a+c)2^{a_t+1}$$

nodes to:

$$\sum_{i=1}^{t}(n-a+c)2^{\frac{3}{7}(a_t+1)}$$

nodes.

AT has interesting implications for initials and names. For example the string `John McMahon and Francis J. Smith` has approximately the same sized search space as the string `J. McMahon and F. J. Smith`, because while the extra characters could not have had tags inserted between them, the added "`.`" could have a tag inserted before it. More generally AT often works as well or better on long terms as on abbreviations of those terms, since the long terms are effectively abbreviated by the tokeniser.

Because there are only 16 contexts in which to see a tag, the chances of encountering all correct tag contexts when training on enough training data to build effective order 4 PPMD models is extremely high. While it is possible to construct artificial situations in which this heuristic hails, the authors are not currently aware of any naturally occurring circumstances in which AT will fail.[2]

3.4 Worked Example: Bibliographies

(a) Raw text

```
References
[1] B. J. R. Bailey. Alternatives to Hastings' approximation to the inverse
of the normal cumulative distribution function. Applied Statistics,
30(3):275-276, 1981.
[2] L. C. Dinneen and B. C. Blakesley. Definition of Spearman's footrule.
Applied Statistics, 31(1):66-66, 1982.
```

(b) Marked up text

```
References
[1] <author>B. J. R. Bailey</author>. <title>Alternatives to Hastings'
approximation to the inverse of the normal cumulative distribution
function</title>. <journal>Applied Statistics</journal>, 30(3):275-276,
<year>1981</year>.
[2] <author>L. C. Dinneen</author> and </author>B. C. Blakesley </author>.
<title>Definition of Spearman's footrule </title>. <journal>Applied
Statistics</journal>, 31(1):66-66, <year> 1982</year>.
```

Fig. 4. Extract from a bibliography taken from `http://liinwww.ira.uka.de/bibliography/-index.html` showing (a) the text as it appears in the References section of an article that cites the two papers and (b) the same two citations marked up with four tags.

[2] We are currently implementing a version which uses the full 28 Unicode character classes and are exploring how much training text is needed to see all categories. Categories appear to be distributed in a hyper-geometric distribution as predicted by Zipf's law.

Using the sample of text shown in Figure 4(a), randomly selected from a large bibliography collection, we will show the relative difficulty of marking up text using the methods discussed earlier. Four tags are inserted, `<author>`, `<title>`, `<year>` and `<journal>`, as shown in Figure 4(b); of the four, all but `<title>` are typically quite short. Titles tend to be long because there is a need for the title to be precise and expressive of the topic and nature of the article. Fortunately this same need also leads to the use of long words and precise terminology—and as already discussed, the AT works best on very long words which have long runs of alphabetic characters are not candidates for tag insertion.

	C_{i+1} Character Class			
C_i Character Class	Upper	Lower	Digit	Other
Upper	no	no	no	yes
Lower	no	no	no	yes
Digit	no	no	no	yes
Other	yes	yes	yes	yes

Table 2. The AT tokeniser table for the text shown in Figure 4. Entries are marked "yes" whenever any tag is seen between a pair of characters C_i and C_{i+1} in their respective classes.

The automatic tokeniser builds the table shown in Table 2 from the training data (not shown). The effect of the table is that the prospect of inserting a tag between adjacent letters, adjacent digits or between a letter and a digit is discounted. In fact, the search space is pruned to consider inserting tags only before and after non-alphanumeric characters. This heuristic can be confirmed by looking at the marked-up text above—no tag occurs between a pair of alphanumeric characters.

Table 3 shows the calculated sizes of the four search spaces, initially for the full problem, then for the problem with the longest tag, `<title>`, omitted. Because the `<title>` tag had l_{max} about twice that of the other tags, it requires a much larger search space. The difference between the two halves of the table strongly suggests that l_{max}, the maximum tag length, is a critical factor for determining performance. The last factor of the formula column is the size of the incremental search space (the small black triangle in Figure 3 (b) and (c)).

3.5 Worked Example: Names

Figure 5 shows part of a (contrived) bibliography: a list of names, as used in lists of authors or editors. (a) shows the raw text without any tags; (b) shows the correctly

Search Space	Formula		Size
Full	$\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{n}$	$= 66^{289}$	$\approx 10^{527}$
Viterbi	$(n-a)\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{a+1}$	$= 136 \times 66^{154}$	$\approx 10^{282}$
OTT	$\sum_{i=1}^{t}(n-a+c)2^{a_t+1}$	$\approx 4 \times 136 \times 2^{154}$	$\approx 10^{49}$
AT	$\sum_{i=1}^{t}(n-a+c)2^{\frac{3}{7}(a_t+1)}$	$\approx 4 \times 136 \times 2^{66}$	$\approx 10^{22}$
Full	$\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{n}$	$= 24^{289}$	$\approx 10^{400}$
Viterbi	$(n-a)\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{a+1}$	$= 226 \times 24^{64}$	$\approx 10^{90}$
OTT	$\sum_{i=1}^{t}(n-a+c)2^{a_t+1}$	$\approx 3 \times 226 \times 2^{64}$	$\approx 10^{22}$
AT	$\sum_{i=1}^{t}(n-a+c)2^{\frac{3}{7}(a_t+1)}$	$\approx 3 \times 226 \times 2^{27}$	$\approx 10^{11}$

Table 3. The sizes of the four search spaces using an order 3 PPMD model. The top half shows the full problem ($t = 4, n = 289, l_{max} = 150, o_{max} = 3, a = 153, n-a = 136$ and $\sum_{r=0}^{t} \frac{t!}{(t-r)!} = 66$), the lower half with the long `<title>` tag removed ($t = 3$, $n = 289, l_{max} = 60, o_{max} = 3, a = 63\, c = 0, n-a = 226$ and $\sum_{r=0}^{t} \frac{t!}{(t-r)!} = 24$).

(a) Raw text
`Stuart L. Stewart, Stewart Z. Beagleman and T. Franchi-Zannettacci.`
(b) Correctly marked up text
`<name><forename>Stuart L.</forename> <surname>Stewart</surname></name>, <name> <forename>Stewart Z.</forename> <surname>Beagleman</surname></name> and <name> <forename> T.</forename><surname>Franchi-Zannettacci</surname>/<name>.`
(c) OTT Marked up text—forename first
`<forename>Stuart L. Stewart</forename>, <forename>Stewart Z. </forename> Beagleman and <forename>T.</forename> Franchi-Zannettacci.`
(d) OTT Marked up text—surname first
`Stuart L. <surname>Stewart</surname>, <surname>Stewart </surname> Z. <surname> Beagleman</surname> and T. <surname> Franchi-Zannettacci</surname>.`

Fig. 5. An extract from a (contrived) bibliography showing the difficulties raised when a string can be marked in one of two tags. (a) shows the raw text and (b) the correctly marked up text. (c) shows the text marked up incorrectly when the `</forename>` is sought first. (d) shows the text marked up incorrectly when the `</surname>` is sought first.

marked up text using the tags <forename>, <surname> and <name>. This breakdown of names into parts of names is common in bibliography systems as a preliminary to abbreviation of forenames to initials.

Figure 5 (c) and (d) show the sample of text marked up using OTT; (c) has <forename> marked up and (d) has <surname> marked up. A problem arises with the OTT heuristic, in that the string Stewart can be both a surname and a forename, and the results will depend on the order in which the tags are sought.

An error occurs whether we mark up <surname> or <forename> first. This small chance of confusion is the price exacted by the OTT heuristic for pruning the search space. It may be partially offset by increasing the size of context used by the model, to try to influence the results by more of the surrounding context, but the effect of this is likely to be limited and leads to inordinately large model sizes.

4 Conclusion

We have recast several important problems in text mining as tag insertion problems, showing that properties of the SGML standard enable strong assumptions to be made about the nature of markup. The search space of the tag insertion problem is examined. Viterbi search, One-Tag-at-a-Time search and Automatic Tokenisers are introduced as search techniques for the tag insertion problem and their performance evaluated.

Worked examples are shown drawn from a bibliography and the size of the four search spaces calculated for a short piece of text. Performance is found to be very strongly linked to l_{max}, the maximum allowable length of tag. An example of the possible errors introduced by the one-tag-at-a-time search is given and explained.

An implementation of the ideas described in this paper is currently underway. Once implementation is compete and results obtained for non-trivial examples, the relative utility of tag insertion to express and resolve a broad class of text mining problems will be shown.

Acknowledgments

Thanks to Yingying Wen for assistance with Chinese text issues.

References

1. Nancy A. Chinchor. Overview of MUC-7/MET-2. In *Proc Message Understanding Conference MUC-7*, April 1998.

2. The Unicode Consortium. *The Unicode Standard—Worldwide Character Encoding.* Addison-Wesley, 1992.
3. Charles F. Goldfarb. *The SGML Handbook.* Oxford, 1990.
4. Chris Heegard and Stephen B. Wicker. *Turbo Coding.* Kluwer Academic Publishers, 1999.
5. Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Department of Computer Science, Brown University, June 1993. Also appeared as Technical Report CS-93-28.
6. Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.
7. Wendy Lehnert. A performance evaluation of text analysis technologies. *AI Magazine*, pages 81–94, Fall 1991.
8. Claude Elwood Shannon and Warren Weaver. *The Mathematical Theory of Communication.* The University of Illinois Press, Urbana, 1964.
9. James A. Storer and Martin Cohn, editors. *Proceedings of the Data Compression Conference (DCC)*, Snowbird, Utah, 28–30 March 2000 2000. IEEE Computer Society, IEEE.
10. William J. Teahan. *Modelling English Text*. Ph.D., Department of Computer Science, University of Waikato, Bag 3105, Hamilton, New Zealand, May 1998.
11. William J. Teahan and John G. Cleary. Tag based models of english text. In Storer and Cohn [9], page 582. http://www.cs.waikato.ac.nz/ wjt/papers/DCC98b.ps.gz.
12. William J. Teahan, Yingying Wen, Roger McNab, and Ian H. Witten. A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, In Press.
13. Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.
14. Ian H. Witten and Eibe Frank. *Data Mining: Practical Mechine Learning Tools and Techniques with Java Implementations.*, chapter WEKA: The Waikato Environment for Knowledge Analysis. Morgan Kaufmann, San Francisco, 1999.
15. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes — Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, 115 Fifth Avenue, New York, New York, 1994.
16. Stuart Yeates, David Bainbridge, and Ian H. Witten. Using compression to identify acronyms in text. In Storer and Cohn [9], page 582. A short version of this appears in Working Paper 00/01, Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand, January 2000.