Transform

***Reverse***

***transform***

**(b)**

**(a)**

***Search for the***

***most compressed***

***target text***

Decode

Compressed Target Text

Channel (Noiseless)

Receiver

Sender

Compressed Target Text

***Search for the***

***most probable target text***

Target Text

Encode

Target Text

Source Text (Unknown)

Source Text (Known)

Decode

Observed Text

Channel (Noisy)

Receiver

Sender

***Noisy Channel***

Observed Text

***Search for most***

***probable text***

Decode

Target Text

Source Text (unknown)

**Figure 1a.**

The Decoding Model.

**Figure 1b.**

The Encoding Model.

***Search for most***

***compressed text***

***Noiseless Channel***

Decode

Compressed Text

Encode

Transform

Target Text

***Reverse***

***transform***

Source Text

(known)

**TXT** Library

(for text based operations)

**TLM** Library

(for language modelling operations)

**TTM** Library

(for text transformation operations)

“Noiseless Channel Model” Architecture

**codelength**

**classify**

**decode**

**encode**

Applications

**train**

**segment**

**markup**

**align**

**Tawa Toolkit for Text Processing**

0.000

0.000

**8.807**

24.275

22.948

13.218

15.998

31.398

39.471

26.255

27.055

42.455

25.907

**12.496**

5.385

**5.141**

54.385

50.390

40.262

36.916

42.477

38.482

**30.676**

30.795

39.455

36.398

28.173

**22.447**

26.109

**22.447**



***Search***

***Search***

***Search***

**Figure 1a.**

The noisy channel model.

*Noisy*

*channel*

Target

text

Decode

Source

text

Target text

Source

text

Classify

(Insert tags)

Encode

*Noiseless*

*channel*

Compressed text

Decode

*Remove*

*tags*

**Figure 1b.** Tag insertion model.

Source

text

Generate

Target text

Encode

*Noiseless*

*channel*

Compressed text

Decode

**Figure 1c.** Dual encoding model.

The shaded boxes in Figure 1 (labeled “Decode“, “Classify” and “Generate” respectively) describe the transformation processes being performed in each model. Each perform repeated corrections of the source text combined with search to arrive at the target text. It can be argued that this correction process for 1b & 1c can be modeled using the noisy channel model. However, …

Our paper is organized as follows. In the next section, we detail the design of the information extraction system that we use.

1. An encoding framework

In this section, a framework is proposed that simplifies the conceptualization for a broad range of text mining and NLP applications involving textual transformations. It is is based on the key insight as described in Section 2 that each application performs a search to find the best encoding of the source and target messages.

Cleary & Teahan (1999) defined an Application Programming Interface (API) for modelling sequential text based on text compression models. The intention of the API was to shield the user from the details of the modelling and estimating process, with the primary motivation being to simplify the design of applications where textual models are needed, such as text compression, categorization, and segmentation (by language or by inserting spaces). During the evolutionary process accompanying the development of the API, however, it became apparent that broadening its scope to encompass a much wider range of text operations would be beneficial. For example, it was found that many of the API-based programs that were devised had remarkable similarities in the underlying source code – despite the programs being complex, only a few changes were needed to define the key differences between the applications. It was felt that these similarities could be characterised in some manner to create a much more powerful specification.

These ideas have now been implemented in the Text Mining Toolkit (TMT). The toolkit is a comprehensive extension of the original Cleary & Teahan API and is now available for download at <http://aiia.cs.bangor.ac.uk/software/TMT-1.0.tgz>. The toolkit provides a powerful method for transforming text. The essential differences between quite diverse applications can be characterised by a few transformations as illustrated by Table 3. Listed in the left hand column of the table are a number of applications; a terse representation of the transformations that uniquely characterise each application is given in the right hand column. These are the transformations that are performed when the search for the most probable target text (the one with the best encoding) given the source text is being performed during the encoding phase.

.

|  |  |
| --- | --- |
| **Application** | **TMT Transformations** |
| *Chinese word segmentation* | • → •; • → • ; |
| *English word segmentation* | • → •; α → α ; |
| *Mobile phone texting* | 2 → [abcABC]; 3 → [defDEF]; 4 → [ghiGHI]; 5 → [jklJKL];  6 → [mnoMNO]; 7 → [pqrsPQRS]; 8 → [tuvTUV]; 9 → [wxyzWXYZ] |
| *Spelling correction* | • → •; α → [a..zA..Z]; α → α [a..zA..Z]; … |
| *OCR spelling correction* | • → •; c → e; e → c; m → n; m → ni; n → m; ni → m; El → El; El → H; H → El; … |
| *Language segmentation* | • → •; • → {English, Welsh} • |

**Figure 2.** Some NLP applications and their TMT transformations.

The form *source text* → *target text* is used to denote a transformation from the source text to the target text; for example, El → H denotes that the bigraph El is transformed to the letter H. A similar convention used for regular expressions is used here to represent (and match) the source and target text. i.e. The symbol • is used to denote the “wildcard” symbol – this will match any symbol in the source text; the α symbol will match just alphanumeric symbols; denotes the space symbol. Ranges of symbols are denoted by the symbols between the square brackets ([…] as used for Unix regular expressions). Special *model symbols* are denoted between braces – {…} – these symbols signal to the transformation process to insert a special *sentinel* symbol into the encoding stream (to terminate the coding of the prior context) after which all subsequent symbols will be coded using the new model (up until the next sentinel symbol is encountered).

As a further explanation of the table, the Chinese word segmentation application is characterized by just two transformations – one which keeps the symbol unchanged, and another which inserts an extra space after it. For English word segmentation, the space insertion only occurs for alphanumeric characters. For the mobile phone texting application, ranges of symbols are used to transform each digit into its corresponding alphabetic equivalent (in either lower or upper case); similarly, for spelling correction, the ranges are used to denote that each alphanumeric character can be replaced or followed by any other alphanumeric character. The transformations listed for the OCR spelling correction application are a small sample of the typical corrections taken from Teahan *et al*. (1998). Finally, the language segmentation application specifies that the language model used to encode future symbols should change for every symbol in the source sequence. Note that for all applications (except for mobile phone texting) the transformation • → • is required to ensure that the unaltered source text sequence and all its subsequences will always be included in the search for the marked up sequence with the best encoding.

1. Search algorithms

The TMT toolkit implements several search algorithms. Three of these are illustrated in Figures 3a, 3b and 3c.

Figure 3a shows the initial search tree for a word segmentation problem. Word segmentation is an issue for Asian languages such as Chinese and Japanese since they are written without word delimitation. This poses a problem for a number of applications, such as information retrieval and compression, where better performance is achieved if the application can apply some automatic method to find where the words in the text occur. Chinese word segmentation is explored more fully in Teahan *et. al.* (2001).}.

In this example, the problem is that the source text contains no spaces (in this example, the source text starts with “th”), and the task is to re-insert them. This is done by finding the most probable correct segmentation of the text by first generating alternative segmentations of the text by inserting spaces after each letter and then searching for the best segmentation i.e. the most probable segmentation which has the best encoding according to some language model trained on representative text. Various language models based on text compression schemes – in particular, variations of different order PPM models such as PPMC and PPMD – have been implemented in the toolkit and are available for the developer to experiment with.

The search space in this problem has a branching factor of two (as shown in the figure), since there are two possibilities for each character – the character by itself, or the character followed by a space. Various search algorithms can be used to prune this search space, such as the Viterbi dynamic programming algorithm (Viterbi, 1967) or the stack decoding algorithm [ref]; these have been implemented in the toolkit and are available for the developer to trade off accuracy, memory and execution speed.

The Viterbi algorithm guarantees that the best possible segmentation will be found by using a trellis-based search – all possible segmentation search paths are extended at the same time and the poorer performing alternatives that lead to the same conditioning context are discarded. The Viterbi algorithm is commonly used with hidden Markov models, for example in part of speech tagging programs to assign the most likely sequence of tags to words in a sentence (Charniak, 1993).

One of the drawbacks of the algorithm is that it is an exhaustive search rather than a selective one, unlike sequential decoding algorithms commonly used in convolutional coding which rely on search heuristics to prune the search space (Anderson \& Mohan, 1984). Teahan et al.’s (2001?) Chinese segmentation method used a variation of a variation of the sequential decoding algorithm called the *stack* algorithm – an ordered list of the best paths in a segmentation tree is maintained, and only the best path in terms of some metric is extended while the worst paths are deleted. The metric they used was the compression codelength of the segmentation sequence in terms of the order 5 PPMD model.

In the stack algorithm's ordered list, paths vary in length (essentially, it is like a depth first search whereas the Viterbi search is more like a breadth-first search). For deleting the worst paths, the traditional method is that the list is set to some maximum length, and all the worst paths longer than this length are deleted. For Teahan et al.’s PPM word segmenter, another variation was used instead. All the paths in the ordered list except the first one were deleted if the length of the path plus *m* was still less than the length of the first's path, where *m* is the maximum order of the PPM model. The reasoning behind this pruning heuristic is that it is extremely unlikely (at least for natural language sequences) that the bad path could ever perform better than the current best path as it still needs to encode a future *m* characters despite being already worse in codelength. One further pruning method was also found to significantly improve execution speed. As for the Viterbi algorithm, poorer performing search paths that lead to the same conditioning context are discarded. However, unlike the Viterbi algorithm, search paths vary in length, so only poorer performing search paths with *both* the same length and conditioning context are discarded.

These heuristics have been implemented in the toolkit. Some experimental results using these different search variations are discussed in the next section.



**Figure 3b.** Search tree for language segmentation.

**Figure 3a.** Search tree for word segmentation.

*Maori stream*

<M>

<M>

<M>

<E>

*English stream*

<W>

<W>

*Welsh stream*

**Figure 3c.** Search paths for the source text “there” with three models: a Maori model (subscript M); an English model (subscript E); and a Welsh model (subscript W).

Figure 3b shows the initial search tree for a language segmentation problem. Language segmentation concerns the problem of identifying where in the text different languages occur (for example, documents produced by the Welsh Assembly government will often contain text written in two languages – Welsh and English). Teahan (2000) showed how a compression-based method can also be applied successfully to this problem. The number of possible languages determines the branching factor of the search tree. If there are just two possible languages in the text (i.e. English and Welsh as in Figure 3b) then the approach is analogous to the word segmentation problem (compare Figures 3a and 3b). Rather than inserting or not inserting a space for each state, instead the search either switches to use the other language model to encode the text by encoding a sentinel symbol or remains encoding with the same language model. This is shown in Figure 3b by the use of subscripts i.e. the sequence “tWhE” indicates that the letter “t” was first encoded using the Welsh model, then the encoding switched for the letter “h” to using the English model.

The language segmentation problem is easily specified using the TMT transformation shown in Figure 2. Additionally, it is possible to specify Viterbi and stack searching variations, and combined with the variations of language models that are possible, this provides a range of options for the experimenter to choose from to explore the best setup for this type of application.

In contrast, Figure 3c illustrates the switching algorithm (Teahan & Harper, 2004) which uses a very different type of search to the other two examples in Figures 3a and 3b. The diagram shows possible search paths for the five symbol sequence “there”. Each state (oval) is labeled by the current source symbol (in this case “t”, “h”, “e”, “r” and “e”) along with a subscript to indicate which model is being used to encode it – here three models are being combined: the subscripts “M”, “E” and “W” stand for static order 5 PPMD models trained on Maori, English and Welsh text respectively. For example “tW” used for the bottom left state designates that the current symbol “t” is being encoded using the Welsh model. Non horizontal transitions (arcs) represent points in the coding sequence at which a switch between models has occurred. These transitions have been labeled with the appropriate switch symbol for that transition (these are shown in angle brackets e.g. “<W> designates a switch to the English model).

A path through the trellis-based network from left to right follows a possible coding sequence. Thus the state sequence tE, hE, eE, rM, eM across the middle then top right of the diagram represents the coding sequence “<E>tE hE eE<M> rM eM” where the symbols “t”, “h” and “e” are encoded using the English model, then the coding switches models (by encoding the switch symbol “<M>”) before encoding the symbols “r” and “e” using the Maori model.

For the next symbol, all possible paths are searched (showed by the dashed lines in the diagram), before the best paths are again chosen (shown by the bold lines). Note that since each models maintains a single coding context, then the number of states in each column is constant and equals the number of models being combined by the algorithm (there are three in this example). As a source symbol is processed, the best paths for each state are found, and the poorer performing paths are rejected. As a consequence, only one transition remains into each state, and some paths terminate. An example of this is shown by “eM” and “hW” states.

1. Some Applications

This section describes how the framework can be applied to three specific applications – word segmentation, language segmentation and POS/Named Entity Tagging. Some results are provided for the first two applications.

Alternative algorithms are also explored based on using heuristic search methods rather than exhaustive search.

A framework is then proposed that substantially simplifies the conceptualization

for a broad range of correction-based applications.

\subsection{PPM based word segmentation---Another case study}

\label{section.segmenter}

Teahan {\em et al.} (1998) describe how a fixed order PPM model can be used

to correct sequences of English text with a high degree of accuracy.

They applied their algorithm to two problems: correction of OCR data, and

word segmentation. The performance of the PPM word segmenter was startling – over 99\%

accuracy.

\subsection{Word segmentation of Chinese text}

Experiments with segmenting Chinese text were performed using Guo Jin's Mandarin Chinese PH corpus containing almost two and a half million words of newspaper stories from the Xinhua news agency of PR China written between January 1990 and March 1991 (Hockenmaier \& Brew, 1998). The corpus contains pre-segmented text encoded using the standard GB coding scheme.

In the experiments, an order 5 PPMD model was trained on the second half of the corpus. This was used to segment part of the first half which was split into fifty sub-sections each containing 10,000 words starting from the beginning of the corpus. Word delimiters were removed from each of the fifty sub-sections

prior to performing the segmentation experiments.

The results obtained are shown in Table 1. The experiments were repeated with the second half of the corpus using the start of the corpus for training instead with similar results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Ave.  edit  distance | Ave.  accuracy  (%) | Time | #Nodes |
| Viterbi | 438 | 98.91 | 109 | 968814 |
| Teahan et al. (2001) | 440 | 98.90 | 94 | 970176 |
| Stack (size=25) | 483 | 98.80 | 212 | 1016780 |
| Stack  (size=50) | 475 | 98.82 | 422 | 1104970 |

**Table 1.** Segmenting words in Chinese text

As shown in the table's leftmost column, four different methods were used to segment each of the test subsections – the Viterbi algorithm, the method used by Teahan *et al*., and two variants of the stack decoding algorithm, one with a maximum stack depth of 25, the other with 50. The next column in the table shows the average edit distance observed between the segmented test sub-sections of the corpus and their corresponding correct form obtained from the pre-segmented corpus. The edit distance can be used as a measure of the accuracy of the segmentation process (Teahan *et al*., 1998). The third column converts these figures into an average edit distance accuracy based on the average size of text contained in each of the test sub-sections. The fourth column lists the average CPU time (in seconds) it took to segment the test sub-sections on a Pentium II 200 MHz processor with 128 MB RAM running Red Hat 5.2 Linux. The final column lists the average number of nodes stored in the paths tree constructed as the algorithms were executed. This figure can be used to gauge the memory consumption of each of the algorithms. From the results it can be seen that the best method in terms of application accuracy is the Viterbi algorithm. This is not surprising as the method guarantees that the most probable segmentation is found unlike the other methods which rely on search heuristics instead to prune the search space. However, the difference in accuracy between the four methods is small, ranging from 98.8% to 98.9% in accuracy, although the average number of errors covered a much wider range, from an edit distance of 438 up to 483. The method used by Teahan {\em et al.} is the nearest to that of the Viterbi algorithm in terms of the number of errors made, but is slightly faster in speed while consuming slightly

more memory resources. In comparison, the performance of the stack algorithm using either of the stack depths is much worse, taking over twice or four times as long in execution speed and also requiring increased memory resources.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503-512.

Ashok K. Chandra, Dexter C. Kozen, and Larry J.Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114-133.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK