Natural Language Engineering **1** (1): 1–19. Printed in the United Kingdom

Abbreviated Text Input Using Language Modeling

STUART M. SHIEBER and RANI NELKEN
Division of Engineering and Applied Sciences, Harvard University,
33 Oxford Street,
Cambridge, MA 02138 USA
{*shieber,nelken*}*@deas.harvard.edu*

(*Received 12 December 2005*)

Abstract

We address the problem of improving the efficiency of natural language text input under degraded conditions (for instance, on mobile computing devices or by disabled users), by taking advantage of the informational redundancy in natural language. Previous approaches to this problem have been based on the idea of *prediction* of the text, but these require the user to take overt action to verify or select the system's predictions. We propose taking advantage of the duality between prediction and *compression*. We allow the user to enter text in compressed form, in particular, using a simple stipulated abbreviation method that reduces characters by 26.4%, yet is simple enough that it can be learned easily and generated relatively fluently. We decode the abbreviated text using a statistical generative model of abbreviation, with a residual word error rate of 3.3%. The chief component of this model is an n-gram language model. Because the system's operation is completely independent from the user's, the overhead from cognitive task switching and attending to the system's actions online is eliminated, opening up the possibility that the compression-based method can achieve text input efficiency improvements where the prediction-based methods have not. We report the results of a user study evaluating this method.

1 Introduction

The problem of text input with devices under degraded conditions is not new; disabled users, for instance, have had to interact with computers using sometimes severely degraded means, including mouth sticks, symbol-scanning systems, eye-gaze tracking, and so forth. The problem has renewed currency, however, because of the increased prevalence of small and embedded computing systems (handheld computers, cell phones, digital video recorders, and the like) for which traditional text input and verification modalities (keyboard and monitor) are impractical.

Natural language text is highly redundant; Shannon's estimates (1951) place the entropy of English text at below a bit and a half per character. Theoretically, this invites the possibility that the redundancies could be used to allow more efficient text entry. The traditional approach to take advantage of this redundancy relies on *prediction* of the user's text. For instance, many cell phones have the technology to

predict the most likely word based on the initial letters typed by the user. The user is required to merely verify the prediction rather than typing the remaining characters. Other methods dynamically predict the next character. A paradigm example is the *Reactive Keyboard* of Darragh and Witten (1992) though the approach arose as early as the early 1970's.

Though intuitively plausible, prediction suffers in practice from severe problems: Because users must take overt action to verify or select, they must be constantly attending to the system's predictions. Typing moves from a fluent, unconscious task to one in which each keystroke requires a significant cognitive load. Previous research (Goodenough-Trepagnier et al., 1986) has shown that the overheads involved swamp any advantages in speed gained unless the keystroke rate is extremely slow. For this reason, these predictive methods are only useful and have only found acceptance among severely disabled users.[1]

Our approach is based on the duality of prediction and compression (Bell et al., 1990). A good statistical model of language, one that can generate good predictions, can inherently be used for compression as well. If we can have the user enter compressed text, the compression of which is based on a good predictive model, we can then use that model to decode the compressed text into the intended full text. The advantage of the compression approach over the previous prediction approach is clear: The generation of the (compressed) text is not an interactive task that requires task switching, verification of system proposals, selection of options, and so forth. The cognitive load increase is limited to that induced by the ability to fluently generate compressed text.

Because a person must generate the compressed text fluently, we require a *human-centered compression method*. As a *reductio*, imagine choosing a standard "computer-centered" method, say, some Lempel-Ziv (LZ) variant, as used in the standard gzip compression facility. We might expect to obtain a two to one reduction in keystrokes or more, at the cost of requiring a user to compute the LZ compression of the original text mentally, an obvious absurdity. The question arises, then, as to how to devise a human-centered compression method to limit this cognitive load.

Conceptually, there are two possibilities.

Stipulated compression First, we can conform the user's behavior to a particular model by *stipulating* a compression method, so long as the stipulated method (unlike LZ) is simple and easily learnable. In practice, the learnability requirement means that the compressed forms of words must be abbreviations of some sort. In fact, the literature has traditionally distinguished prediction approaches from abbreviation approaches (Vanderheiden and Kelso, 1987), which have been taken to be of this stipulated variety.

Natural compression Alternatively, we can try to conform the model to the

[1] The exception that proves this rule is the use of auto-completion for very specific tasks, such as entering long URLs into web browsers, which can be seen as a kind of dilute version of predictive typing. In this application the payoff in terms of keystrokes saved may be so large that the overheads can be tolerated.

> user's natural behavior by allowing a *natural* compression method, one that users would naturally turn to when compressing text.
> As it turns out, there seems to be a more or less standard compression method, a kind of ad hoc abbreviation form, well understood by average writers of English, and best exemplified by the old advertising slogan "If u cn rd ths, u cn gt a gd jb".[2]

In this paper we report our experiments with a human-centered simple stipulated word abbreviation method. A method relatively well matched to the natural method, is simply to drop all vowels.[3] (We always treat "y" as a consonant.) Noting that letters early in the word are most predictive of the remainder, we retain the first letter even when it is a vowel. (This solves the problem of what to do with words consisting of only a single vowel as well.) In addition, we drop consecutive duplicate consonants. Thus, the word "association" would be abbreviated "asctn" under this method, and the sentence

We have conducted a thorough evaluation of this disabbreviation method.

would be abbreviated as

W hv cndctd a thrgh evltn of ths dsbrvtn mthd.

with 24 fewer characters, 33.8% of the 71 in the original.

We describe the abbreviation method in Section 2, including a presentation of the basic method, its implementation, its evaluation, and several extensions. Section 3 details the user study. Finally, a review of related research is given in Section 4.

2 Method

To decode text that has been abbreviated using the stipulated method, we have created a generative probabilistic model of the abbreviation process as a weighted finite state transducer (Pereira and Riley, 1997). The model transduces word sequences, weighted according to a language model, to the corresponding abbreviated character sequences. The model is explicitly constructed by composing a language model, representing the probability of a word-sequence, $p(W)$, and an abbreviation model (or "channel" model), representing the conditional probability of the abbreviation, A, given W, $p(A \mid W)$. The composed model therefore models the joint probability, $p(W, A) = p(W)p(A \mid W)$.

Given a particular abbreviated form, A, we seek the most likely word-sequence, W, that could have generated it, i.e., $argmax_W\ p(W \mid A)$. By Bayes' rule,

[2] This was a marketing slogan for a shorthand technique called "Speedwriting" that incorporates, in part, a stipulated abbreviation model with a small set of rules that include, among others, dropping silent letters, replacing letters with phonetic equivalents (k for c in "cat" for instance), dropping short vowels unless at the beginning of the word, using special symbols for frequent words, and so forth. Though more complex and difficult to learn than the abbreviation methods we discuss below, the system bears some similarities.

[3] Something like this has been proposed by Tanaka-Ishii.(2001) for Japanese.

$$argmax_W\ p(W \mid A) = argmax_W\ \frac{p(W, A)}{p(A)} \quad . \tag{1}$$

Since A is given, and is the same for all disabbreviations, we can ignore the denominator. Hence, finding the most likely sequence of words W through the model yields the most likely disabbreviation. We use Viterbi decoding, a standard algorithm for efficiently computing the best path through an automaton, to find W.

2.1 Component transducers

Weighted finite-state transducers constitute a simple general technology for modeling probabilistic string-to-string transformations. Their nice closure properties, especially closure under composition, make them ideal for the present application in that the model can be composed as a cascade of simpler transducers in an elegant fashion. These include:

An n-gram language model (LM) The language model, which implements the $p(W)$ component of the generative model, was trained on a text of size 1.8 million words from Wall Street Journal articles (from July 1994), and implemented as a finite-state acceptor. Numbers and unknown words are replaced by special tokens.

A spelling model (SP) This transducer serves the purely technical purpose of converting words into the sequence of characters that compose them. This change in token resolution is required since the language model operates on word tokens and the following transducers in the cascade operate on character tokens. This transducer is constructed by creating a separate path of states for each word, in which the word is first transduced to the null symbol, ϵ, followed by the transduction of ϵ to each of the word's letters, as illustrated for two words in Figure 1. To complete the loop, there is an added transition from the final state to the initial state that generates a space (represented as $\sqcup$ in Figure 1). To compact the transducer, we determinize it on the input symbols.[4]

A compression model (CMP) This transducer implements the stipulated abbreviation model, removing non-initial vowels and doubled consonants. The transducer has a unigram memory of the last character seen. Starting from the second letter, any vowel is transduced to ϵ. A consonant is transduced to ϵ, if it is the same as the previous letter. This is illustrated in Figure 2, for an alphabet restricted to two letters—a vowel (a) and a consonant (b). Special symbols for unknowns and numbers, as well as punctuation marks are left intact. CMP implements the $p(A \mid W)$ component of the model, and is deterministic, i.e., for any W and *abbrev*, it is either 0 or 1, depending

[4] For large training data sets (see below) determinization became infeasible due to memory usage, in which case we used the undeterminized transducer.

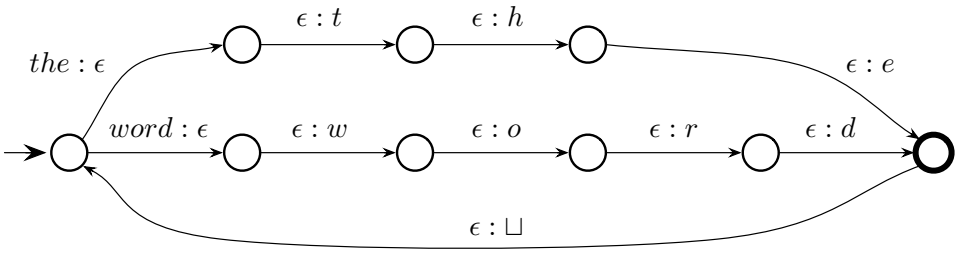

Fig. 1. Spelling model

on whether that sequence of words can be abbreviated as that sequence of letters.

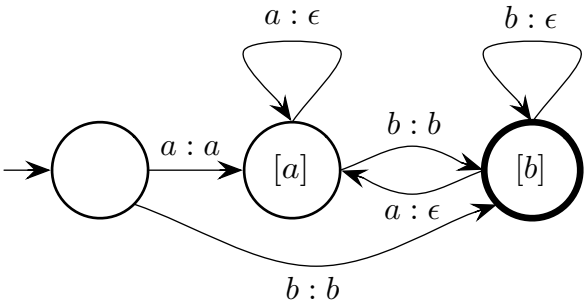

Fig. 2. Compression model

An unknowns model (UNK) This transducer replaces the special tokens for unknowns and numbers with sequences of characters or digits, according to a simple generative model: reading the token ⟨NUM⟩ or ⟨UNK⟩ as input, it enters a loop emitting arbitrary digits or characters, respectively.

The composition of these four transducers forms the entire abbreviation model as illustrated in Figure 3 (but see below for extensions). The composed transducer is deterministic in the forward direction (with the exception of UNK), i.e., a given sequence of words has a single abbreviation. It is nondeterministic in the backward direction; multiple word sequences may yield the same abbreviation. Viterbi decoding chooses the most probable of these.

For instance, the string of words "⟨*an*⟩ ⟨*example*⟩ ⟨*of*⟩ ⟨NUM⟩ ⟨*words*⟩" would be successively assigned a probability according to the language model (LM); converted to the sequence of characters "`an␣example␣of␣`⟨NUM⟩`␣words`" (SP); abbreviated to the sequence "`an␣exmpl␣of␣`⟨NUM⟩`␣wrds`" (CMP); and completed by instantiation of the special token ⟨NUM⟩ to, e.g., "`an exmpl of 5 wrds`" (UNK).

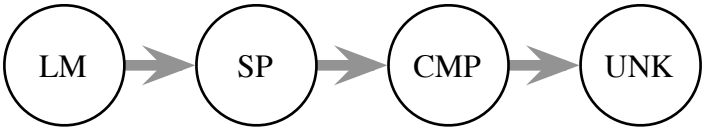

Fig. 3. Abbreviation model

Through this transduction, then, the model associates the word sequence "⟨*an*⟩ ⟨*example*⟩ ⟨*of*⟩ ⟨NUM⟩ ⟨*words*⟩" as the underlying source for the abbreviation: "`an exmpl of 5 wrds`". Of course, other word sequences may also be transduced to the same character sequence, for instance, "⟨*an*⟩ ⟨*example*⟩ ⟨*off*⟩ ⟨NUM⟩ ⟨*wards*⟩". The transducer, through the probabilities manifest in the submodels, most importantly LM, assigns different probabilities to the various sources of the abbreviated string. Viterbi decoding efficiently selects the most likely source. Once the proposed source for the string is computed by this method, the final decoded string is generated by a simple post-processing step that replaces the special tokens ⟨NUM⟩ and ⟨UNK⟩ with the corresponding tokens from the abbreviated form.

The model uses lower case for all text. To handle input text that includes capital letters, we fold its case as a pre-processing step, and then restore the capitals by comparison with the original input, as a post-processing step.

2.2 Implementation

The system is implemented using the AT&T FSM and GRM libraries (Mohri et al., 2000; Allauzen et al., 2003). The FSM library provides a collection of tools for constructing weighted finite-state transducers, including their specification, compilation, composition, and Viterbi decoding. The GRM library provides tools for constructing finite-state language models. Additional code for gluing together the library processes for transducer construction, decoding and evaluation was implemented as a series of Perl scripts.

We trained the language model on a training set of Wall Street Journal articles, after performing several preprocessing steps, including

- stripping any markup information (such as headers, article identifiers, paragraph separation markers, etc.);
- splitting the text into sentences using the Alembic workbench (Aberdeen et al., 1995);
- replacing numbers with the special token ⟨NUM⟩.

We limit the vocabulary of the model to about 97,000 most frequent words. Words are counted using the CMU-Cambridge Statistical Language Modeling Toolkit (Clarkson and Rosenfeld, 1997). All other words in the model are considered unknown and automatically replaced by the ⟨UNK⟩ token. Increasing vocabulary size improves decoding accuracy but increases the language model size and consequently decoding time.

After preprocessing, we train an n-gram model (up to trigrams) using the AT&T GRM library. We use Katz backoff discounting (Allauzen et al., 2003) for smoothing.[5] The other models (SP, CMP, UNK) are all straightforwardly implemented using the AT&T FSM package.

In order to run Viterbi decoding on an abbreviated text, we represent the text as an automaton, TXT, which consists of a linear sequence of states, one per character instance. Composing the transducer cascade with TXT yields a stochastic generative model of this abbreviation text. In theory, it would seem more effective to compile the composed cascade in advance, and then vary the input abbreviated text, as follows: $(LM \circ SP \circ CMP \circ UNK) \circ TXT$. In practice, however, the size of the composed transducer cascade quickly becomes prohibitive. Instead, we found it much more tractable to incrementally compose the transducers in pairs in reverse order: $(LM \circ (SP \circ (CMP \circ (UNK \circ TXT))))$, constructing the composition on-the-fly for each new input text. We start from the last transducer in the cascade, UNK, and compose it with TXT, then compose the previous transducer, CMP, with the result, and so forth. This approach prunes a large number of states and transitions corresponding to character sequences that are never manifested in the abbreviated text input. This is most apparent for words and word n-grams in the language model that do not appear in abbreviated form in the input (i.e., sequences W, such that $p(A \mid W) = 0$). Composing the cascade in the forward direction, we would have to carry the states for these words, and compose them with their spellings, and abbreviations. Composing in the reverse direction, these words are eliminated, since they are never represented in TXT.

2.3 Evaluation

We performed evaluation studies on a held-out corpus of 10 sections of Wall Street Journal text (from August 1994) of about 80,000 words each, for a total of roughly 840,000 words. We abbreviated each text by running CMP in the forward direction. We then ran the disabbreviation procedure, comparing the resulting decoding with the original text. We report two main dimensions of evaluation: keystroke reduction and error rate.

The stipulated abbreviation model achieves 26.4% reduction at 4.57% error rate, measured as percentage of word instances incorrectly decoded averaged over the 10 text sections (and standard deviation of 0.25% in error rate). As a reference upper bound, Lempel-Ziv 77 compression on this corpus (in its entirety) provides a 62.7% reduction and is lossless (though of course, this is not a realistic text-entry method). Traditional predictive methods, such as ANTIC, ANTICIPATOR, PAL, and, PREDICT, have reported maximal keystroke savings of 20 to 50%. See the discussion by Soede and Foulds (1986) and references cited therein.

The benefits of language modeling can be clearly seen by comparing performance against cascades using simpler language models. Table 1 provides the performance

[5] We have also experimented with other smoothing methods such as Kneser-Ney with only negligible variation in accuracy.

Table 1. *Performance of the disabbreviation method using a variety of language models*

Model	*Average error rate* percent	*Standard deviation of error* percent
uniform	51.36%	0.36%
unigram	8.49%	0.25%
bigram	4.67%	0.25%
trigram	4.57%	0.25%

of the system under increasingly complex language models, from uniform to unigram, bigram, and trigram. Of particular importance is the improvement of the bi- and trigram models over the unigram model, demonstrating that this approach is likely to have application to any abbreviation method that ignores context, as prior methods do.

The high success rate of the method illustrates the effectiveness of using local context for reconstructing the words. Conversely, many of the errors can be attributed to cases where the local context is not informative enough. Here are some examples:

- "just a guy, a dog and a couple of *boris* ...." (should have been: beers),
- "skilled in the art of *copper*, they remove your shirt and prepare to revive you" (should have been: cpr),
- "hold on to your *bats*" (should have been: boots),
- "dancers do too, but if they're lucky, they get old *frost*" (should have been: first),
- "the agency named several *beer* heads" (should have been: bureau).

As can be expected, accuracy is maximized when the test texts stay within typical standard Wall Street Journal topics and vocabulary. Texts that stray towards less typical topics, such as the occasional art review, yield more errors.

The confusion of "*these*" and "*this*" is a recurring error, as the system has no knowledge of grammatical factors such as number. In addition, unknown words are a major source of errors. We discuss improved handling of them below.

2.4 Extensions

The remarkable simplicity and modularity of the finite-state architecture enable modifications and extensions to the basic model described above to be easily performed. We have experimented with the following changes. Evaluation results are given in Table 2.

"Forgiving" abbreviation model Informal user experimentation has shown that whereas the stipulated model is fairly simple to learn, users will sometimes forget to drop all of the vowels or repeated consonants. Unfortunately, this leads to a failure to decode, as the basic model assumes strict adherence to deterministic letter dropping rules. A minimal change to the original compression

model makes it nondeterministic in the forward direction by allowing a small probability, δ, of not dropping the required vowels and repeated consonants. Graphically, this requires changing the transducer in Figure 2 by adding a self transition of the form $a : a$ in state $[a]$ with probability δ and setting the probability of dropping the vowel, $a : \epsilon$ to the complementary $1 - \delta$. Likewise for dropping the b in state $[b]$. The result is shown in Figure 4.

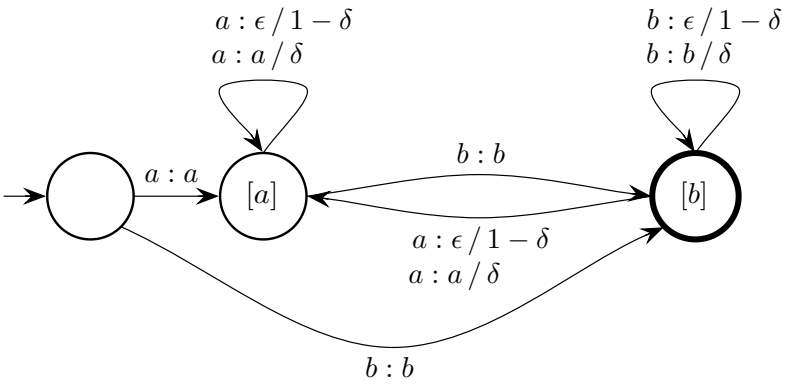

Fig. 4. "Forgiving" abbreviation model

Note that the forgiving model's nondeterminism has a subtle effect on decoding. For instance, consider the abbreviated sequence of characters "ths", which may be generated by the words "this", and "these" (inter alia). In the original model, abbreviation is deterministic, hence $p(ths \mid this) = p(ths \mid these) = 1$. Thus, the difference between the likelihood of the two options depends solely on the relative probabilities of the two words. Here, we have to take into account the probabilities of dropping the vowels too, $p(ths \mid this) = 1 - \delta \neq p(ths \mid these) = (1 - \delta)^2$. Thus, there is a slight bias towards shorter words. Evaluation shows that this leads to only a negligible degradation in decoding accuracy, as shown in Table 2.

In addition to allowing users to inadvertently retain some of the vowels and repeated consonants, the forgiving model also allows user to purposely keep some of these letters. A user may want to do so to ensure that rare or highly ambiguous words and phrases would be correctly decoded with only a small loss in compression rate.

Keypad Hashing As an additional compression method, we allow users to replace letters by the standard digit equivalent on a 12-key telephone keypad (that is, the letters 'a', 'b', and 'c' with the digit '2', the letters 'e', 'f', and 'g' with '3', etc. punctuation marks are replaced by '$*$', and spaces by '#') to support cell-phone text input. Since this mapping is many-to-one, most methods for cell phone text entry require multiple keystrokes per character. By contrast, we allow hashed input using a single keystroke per character. Dehashing is performed using the same method, relying on the language model to disam-

Table 2. *Performance of the disabbreviation method using a variety of language models*

Model	*Average error rate* percent	*Standard deviation of error* percent
trigram	4.57%	0.25%
forgiving	4.61%	0.24%
keypad hashing standalone	5.61%	0.22%
keypad hashing and abbreviation	13.89%	0.45%
capitalization	4.39%	0.21%
7-gram standalone letter	7.10%	0.32%
full model	4.01%	0.17%

Table 3. *Keypad hash sample decodings*

hashed form	*decoding*	*correct decoding*	*correct?*
22253	cable	cable	$\surd$
38	eu	38	$\times$
786733	pumped	stored	$\times$
*	,	.	$\times$

biguate. Keypad hashing is straightforwardly implemented as a transducer, KEY. We allow hashing to be used either in isolation (by replacing CMP with KEY) or on top of abbreviation (by composing KEY and CMP). Results for both configurations are given in Table 2 and some example decodings (without abbreviation) are shown in Table 3.

Whereas in the original model, the ambiguity in the input stems from missing letters, when keypad hashing is used, the actual identity of characters becomes a major source of ambiguity. For instance, as shown in the table, the model confused "38" with "EU". Likewise, punctuation marks, which are uniformly replaced by '$*$', are a cause of many errors. Finally, the post-processing step of replacing $\langle$NUM$\rangle$ and $\langle$UNK$\rangle$ with the original input string is also complicated. For numbers, we reconstruct the digits from the input, and use a heuristic to choose between commas (as in 3,500) and decimal points (as in 3.500). Unknown words, however, cannot be reconstructed from the hashed input. Unsurprisingly, when hashing and abbreviation are combined, the results are severely degraded.

Capitalization In the basic model we use lower case for all text. A more refined model can be obtained by using the true case of the words, for instance distinguishing between "exchange" (the action) and "Exchange" (the institution). A straightforward approach to adding case distinctions would simply use the true case of the letters in both the training and the input. This method, however, runs into data-sparseness problems with respect to the first word of the sentence. We wish to be able to decode first words even if they do not appear capitalized in the training set. For instance, consider an input sentence beginning in "Cbl", which should be decoded as "Cable". We wish to

Table 4. *Letter model sample decodings*

abbreviated form	*decoding*	*correct decoding*	*correct?*
pltyps	platypus	platypus	√
sympthzng	sympathizing	sympathizing	√
mscmnctn	miscommunication	miscommunication	√
mrspl	marsupial	marsupial	√
rclctrnc	recalcitrance	recalcitrance	√
antltst	antilatest	antielitist	×
strptkns	stripteakens	streptokinase	×
mldns	mouldens	mildness	×
Aljndr	Alojoinder	Alejandro	×
McWrld	MacaWirled	MacWorld	×

decode this word correctly even if the training set contains the word "cable" in lower case, but not the capitalized version. To handle this, we fold the case of the sentence-initial word in both the training and the input texts. This is done using an additional transducer, CAP, which is added at the end of the cascade.

Letter model for out-of-vocabulary words A major source of errors in the basic system is the occurrence of unknown words in the text to be abbreviated. Clearly, if a word is not included in the language model's training text, the system will not be able to correctly disabbreviate it. Increasing the vocabulary helps mitigate the problem, but cannot solve it completely. We have therefore added a letter model as an alternative generative model of the abbreviated sequences. The letter model is constructed very similarly to the basic cascade described in Section 2. The n-gram word model is replaced by an n-gram character model. In addition, the need for SP is obviated. We train the character model on (character sequences that form) words in the vocabulary extracted from the same Wall Street Journal training texts. After running an abbreviated text through the Viterbi decoder on the main transducer, any remaining unknown words (decoded as ⟨UNK⟩) are run through the letter model.

Our usage of the letter model is restricted to out-of-vocabulary words, and so we only consider character sequences corresponding to words. Alternatively, we could use a letter model in isolation, replacing the word model altogether. This requires training the letter model not only on single words, but on character sequences transcending word boundaries. Using n-grams of high enough order, a letter model can cover on average the same span as a bigram or trigram word model. Experimental results with a 7-gram standalone model, also smoothed using Katz backoff, are given in Table 2. Taking word tokens into account, however, leads to more parsimonious models and improved accuracy. Some examples of decodings of the letter model are given in Table 4. These examples illustrate that the letter model reflects a good approximation of the letter patterns of English. In particular, the model tries to force names of foreign origin into these patterns.

Table 5. *Performance of the the full model as a factor of training dataset size*

Training data size million words	*Average error rate* percent	*Standard deviation of error* percent
1.80M	4.01%	0.25%
3.68M	3.30%	0.22%
24.81M	2.67%	0.19%

We achieved the best accuracy results by combining several of these extensions, denoted by "full model" in Table 2. This model incorporates a trigram model, capitalization, and a 10-gram letter model for unknown words.

Finally, we experimented with enlarging the training dataset size for the word language model, using additional Wall Street Journal text as shown in Table 5. As can be expected, accuracy improves with training data size. At 3.68 million words, the full model achieves 3.30% accuracy rather than the original simple trigram's 4.57%. Despite appearances, this is a relatively large reduction in error of 27.8%. Enlarging the training dataset size to 24.81 million words reduces the error even further. Since there is no shortage of plain English text, the only bounds on the training data sizes are dictated by performance considerations. At 3.68M words, the system still disabbreviates at rates well below 1 second per sentence. At 24.81M words, disabbreviation takes several seconds per sentence, which is an unreasonable time for a user to wait.

3 User study

To assess the practicality and efficiency of our abbreviation method, we conducted a user study in which human subjects (with no prior knowledge of either the project or the underlying technology) were asked to abbreviate sentences, and their typing speeds were recorded. As a control, we also asked the participants to copy sentences without dropping any vowels, and compared the average typing speeds.

3.1 Experimental design

The experiment proceeded in two stages, and was directed from a specially designed Web site. In the first stage, devoted to training, we first gave subjects a set of instructions explaining the stipulated abbreviation method, and then allowed them to experiment with the abbreviation and copying procedures. The Web site presented participants with sentences and asked them to abbreviate them or to copy them fully. We chose sentences from the Wall-Street Journal corpus, for which the decoding error rate was similar to the average error rate.[6] Once a user submits an abbreviated sentence, the system automatically calls the decoding procedure, to find the most likely disabbreviation of that sentence. Crucially, this usage method

[6] A 3.3% error rate corresponds roughly to either one or no error for an average length sentence.

is quite different from the constant task-switching required by prediction-based methods. The result is compared with the original sentence and checked for discrepancies, which may arise either from user typing errors, or from system decoding errors. Subjects are displayed the decoded text with the errors highlighted, and are asked to correct the errors and resubmit the text. This process is repeated until the submitted text agrees completely with the original. Likewise, for copying, once the user submits the copied text, any copying errors are highlighted, and the user is asked to correct them. Users were given one set of sentences to abbreviated and one set to copy. The choice of which set of sentences was abbreviated and which was copied was randomly varied between participants as were the order of abbreviation and copying as well as the order of the sentences within each set.

Since our method is geared towards user input with devices that lack an ordinary keyboard, we disabled the regular computer keyboard in our experiments, replacing it with a software-based on-screen keyboard, controlled by mouse clicks. Using a virtual keyboard also eliminates any bias towards copying offered by users' familiarity with touch typing. Since users might be accustomed with quickly typing letter combinations corresponding to regular words but not their abbreviations, such bias might arise with a regular keyboard.

After participants successfully completed several rounds of copying and abbreviation for the training sentences, they were given a chance to ask clarifying questions, and then moved on to the evaluation stage. In this stage, participants were presented with two sets of ten sentences to abbreviate and to copy, respectively (again, we randomly varied the order between subjects), and the Web site kept track of the typing and correction times. We report results only for the evaluation stage. The entire interaction took roughly one hour on average.

3.2 Results

We ran experiments with 16 subjects. We define the copying speed as the number of characters in the original sentence divided by the time in minutes it took the subject to submit the typed sentence. Similarly, we define the abbreviation speed as the number of original characters divided by the time it took the user to enter the abbreviation.

Averaged over the 16 participants, and without taking correction times into account, abbreviation yields an average speed-up of 12.24% (standard deviation = 4.97%). Tracking these results over the ten abbreviation/copying rounds, the average speed-up increases, as illustrated in Figure 5, which shows the average relative speedup of abbreviation over copying for each round. Abbreviation speeds improve over the rounds, reaching a maximum of 18.63%. A closer look at the raw results shows that whereas copying speeds remain relatively constant, abbreviation speeds improve from round to round, as shown in Figure 6.

Theoretically, the "Power law of learning" (Newell and Rosenbloom, 1981) would predict the logarithm of the abbreviation speed at round n, which we denote by AS_n to increase linearly with the logarithm of n:

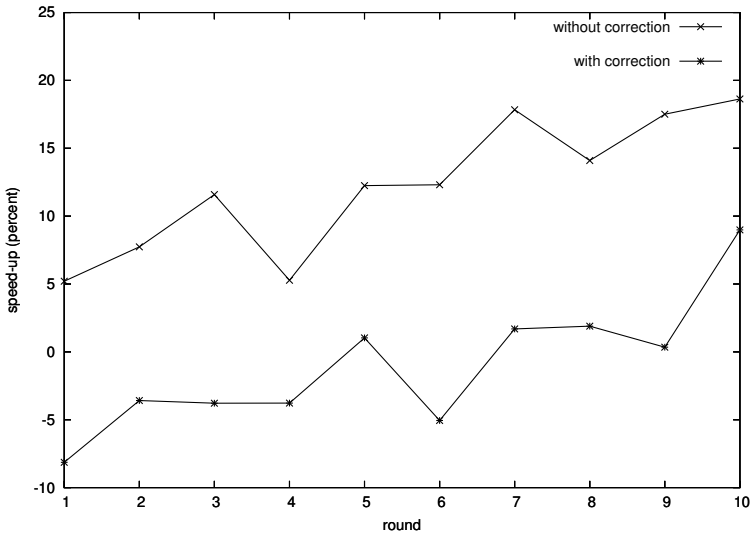

Fig. 5. Average speedup of abbreviation over copying

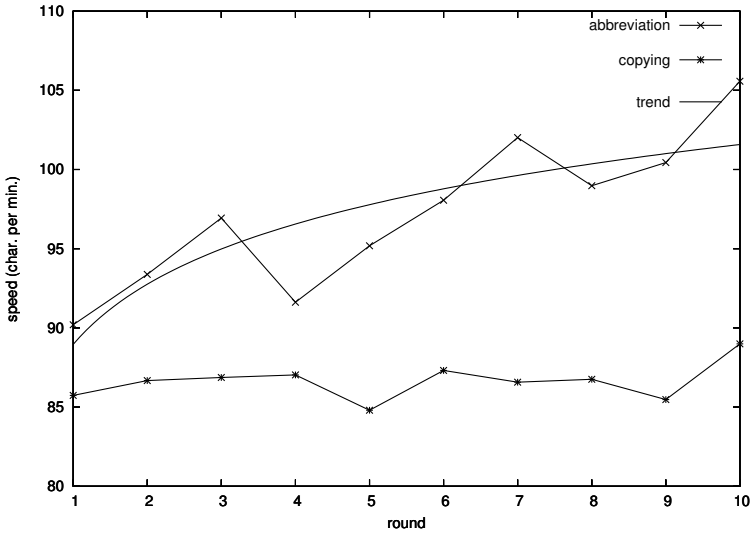

Fig. 6. Copy vs. abbreviation speeds without correction

$$\log(AS_n) = A\log(n) + B$$

where A and B are parameters. The best fit parameters are $A = 5.4821, B = 88.956$, with $r^2 = 0.7034$, a reasonable though imperfect fit.

As a yardstick to assess these results, note that the abbreviated sentences contained 28.3% less characters than the original. Thus, a user who would assimilate the abbreviation method perfectly can be expected to achieve this level of speed-up.

Once we take correction times into account, both for abbreviation and for copying,

the abbreviation speed-ups become much more modest. Averaging over all rounds, abbreviation with correction is in fact slightly slower by 1.04% than copying with correction. Abbreviation speeds improve over the rounds for this setting as well, reaching a maximum improvement of 8.98% in the last round.

Correction clearly impedes the overall performance gains of the compression system. Note however that the simplistic correction method we use—asking the user to correct highlighted errors—is far from optimal (and of course unrealistic in that we cannot automatically identify errors in user-generated text). We are currently exploring improved correction methods that would decrease the user's cognitive load and take advantage of previously computed information. One approach in this direction would present the user not only with the best path through the transducer, but with the top n paths for some suitable value of n. Figure 7 shows the average error rate for the top n most probable decodings, where we count an error only if it is erroneous in *all* top n decodings where n ranges from 1 to 10 (1 corresponds to the single best path as above).[7] Clearly, increasing the range of accepted possibilities leads to improved accuracy. Thus, there is hope that this information could be utilized in an interface that will improve correction times over the naïve approach we used in the user study.

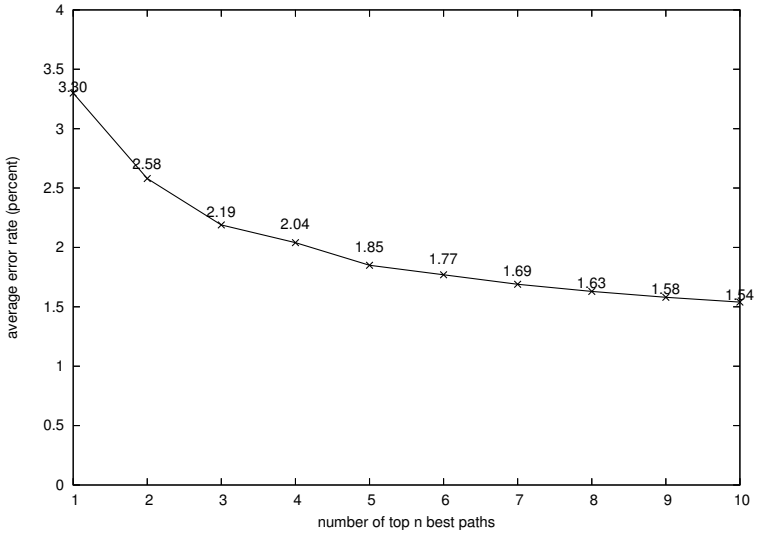

Fig. 7. Performance of the disabbreviation method for the top n readings

As an alternative to actively correcting errors by the sender, one may also envision relying on the recipient's capacity to recognize disabbreviation errors and guess the original words. We performed a second user study to assess whether users would be able to do so. [8] We selected a random contiguous section of Wall Street

[7] In case the decoder can find only n' unique decodings where $n' < n$, we use the top n' decodings instead of the top n.

[8] We thank an anonymous reviewer for suggesting this experiment.

Journal text spanning 150 sentences, abbreviated it and then disabbreviated it. The result included 59 wrongly disabbreviated words. We gave 8 participants this disabbreviated text with a very rudimentary description of the kind of errors they might encounter—explaining merely that while the consonants are always correct, some of the vowels might not be. We asked participants to mark any errors they can find and to suggest a correction for them. On average, the participants correctly identified 71.4% of the errors. 95.2% of their suggested corrections for these errors were the right ones. In addition, they wrongly marked an average of 4.4 correct words as errors (of the whole text). Thus, it is likely that recipients would be able to correctly identify the vast majority of disabbreviation errors.

4 Review of Related Research

As noted above, text input methods based on predicting what the user is typing have been widely investigated; see the work by Darragh and Witten (1992) and references cited therein. Such systems can be found in a variety of tools for the disabled, and some commercial software, most notably the T9 system from Tegic. Methods based on static lookup in a fixed dictionary of codes for words or phrases include Vanderheiden's Speedkey (1987), along with a wide range of commercial keyboard macro tools that require user customization. All rely on the user's memorization of the codes, which must be extensive to provide much compression advantage. Systematic stipulated compression models can be found hidden in stenographic methods such as Speedwriting, though there is no provision for automated decompression.

A recent dynamic prediction approach is used by *Dasher* (Ward and MacKay, 2002), a system in which the predicted characters stream onto the screen towards the constructed sentence, in shaded boxes of sizes proportional to their likelihood, and the user has to choose the next character using a mouse or an eye-tracking device. Dasher's predictions are based on a text compression algorithm called *Prediction by Partial Match* (PPM) (Cleary and Witten, 1984; Moffat, 1990).

Some human factors research on the design of command abbreviations for small vocabularies has been performed. John et al. (1985), for instance, show that vowel-dropping leads to more easily recalled abbreviations but slower throughput than abbreviations based on escaped special characters. Extrapolation of such results to abbreviation of arbitrary text is problematic, but the results are not inconsistent with the possibility of throughput benefits under reasonable conditions.

Study of the structure of natural abbreviation behavior has been limited: Rowe and Laitinen (1995) describe a system for semiautomatic disabbreviation of variable names (such as "tempvar" for "temporary variable") in computer programs, based on their analysis of attested rules for constructing such abbreviations. Stum and Demasco (1992) investigate a variety of rules that people seem to use in generating abbreviations, but do not place the rules in a system that allows the kind of automated disabbreviation we are able to perform.

Abbreviation methods at the sentence level include the "compansion" method of Demasco, McCoy, and colleagues (Demasco and McCoy, 1992; McCoy et al., 1994)

and the template approach of Copestake (1997). These techniques, though bearing their own limitations, are fully complementary to the character-based disabbreviation techniques proposed here, and the user interface techniques for error correction developed for our application may be applicable there as well.

In order to learn a more natural abbreviation model, it would be necessary to collect a corpus of abbreviation patterns in actual use. Willis et al. (2002) performed a study in which participants were given a fixed-length text, and were asked to abbreviate it into progressively shorter texts using whatever abbreviation method they prefer. Several abbreviation patterns emerged, which are compatible with our stipulated abbreviation method. For instance, subjects showed a preference for dropping vowels over consonants, a preference for preserving letters in the beginning of the word, and dropping repeated letters. In addition, subjects also used more large-scale deletions, such as truncating the end letters of a word, using phonetic shorthands (such as replacing "ch" with "k") and omitting phonetically silent letters. Another step in this direction was carried out by How (2004), who has collected some 10,000 SMS messages exchanged by students at the University of Singapore. The corpus contains many abbreviations, but unfortunately not their decodings.

5 Conclusion

Our approach to reducing the effort for natural-language text input by using abbreviation as a human-centered compression method, rather than prediction, provides a simple method to attain both reasonable keystroke (or equivalent) reduction and reduced task-switching cognitive load.

In this study we have focused on a simple stipulated abbreviation method. Even with the forgiving model, our method requires users to follow the stipulated abbreviation method fairly rigidly. Further work is required for gracefully recovering from other deviations from the instructions, such as spelling mistakes. The benefit in user flexibility should be balanced with the increase in the potential disabbreviation search space that would stem from such deviations. Ultimately, one would like to to have a much more flexible abbreviation capability, allowing users to enter free-form abbreviations, and employing a decoder trained on a corpus of such naturally abbreviated text. Alternatively, more sophisticated stipulated abbreviation methods can be tested, which might provide better compression ratios at the cost of learnability and fluency of generation.

The approach to text input described in this paper is an instance of the more general paradigm of collaborative user interfaces (Shieber, 1996). According to this view, interfaces should be designed as means for human users and computers to collaborate towards solving a mutual problem, in this case efficient text entry. Unlike predictive methods, which require a high cognitive load on the user, our approach strives towards an optimized split in responsibilities between the user and the computer.

6 Acknowledgments

A prototype implementation of the disabbreviation system (Shieber and Baker, 2003) was carried out by Ellie Baker with the help of Winston Cheng, Bryan Choi, and Reggie Harris. Winston Cheng and Emily Morgan developed the Web site for the user study. Emily Morgan also conducted the user studies. We thank Min-Yen Kan for providing us with the SMS corpus collected as part of Yijue How's undergraduate thesis under his supervision. We wish to thank Mehryar Mohri for his advice on using the AT&T finite-state tools. This work was supported in part by grant IIS-0329089 from the National Science Foundation.

References

Aberdeen, John, Burger, John, Day, David, Hirshman, Lynette, Palmer, David D., Robinson, Patricia and Vilain, Marc. 1995. Description of the ALEMBIC System Used for MUC-6. *Pages 141–155 of: Proceedings of the Sixth Message Understanding Conference (MUC-6)*. San Mateo: Morgan Kaufmann.

Allauzen, Cyril, Mohri, Mehryar, Roark, Brian. 2003. Generalized algorithms for constructing statistical language models. *Pages 40–47 of: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL'2003)*.

Bell, Timothy C., Cleary, John G. and Witten, Ian H. 1990. *Text Compression*. Englewood Cliffs, NJ: Prentice Hall.

Clarkson, P.R. and Rosenfeld, R. 1997. Statistical Language Modeling Using the CMU–Cambridge Toolkit. *In: Proceedings ESCA Eurospeech 1997*.

Cleary, J. G. and Witten, I.H. 1984. Data Compression using adaptive coding and partial string matching. *IEEE transactions on Communications*, 32(4), 396–402.

Copestake, Ann. 1997. Augmented and Alternative NLP Techniques for Augmentative and Alternative Communication. *Pages 37–42 of: Proceedings of the ACL Workshop on Natural Language Processing for Communication Aids*. ACL, Madrid.

Darragh, John J. and Witten, Ian H. 1992. *The Reactive Keyboard*. Cambridge Series on Human-Computer Interaction. Cambridge, England: Cambridge University Press.

Demasco, Patrick W. and McCoy, Kathleen F. 1992. Generating Text from Compressed Input: An Intelligent Interface for People with Severe Motor Impairments. *Communications of the ACM*, 35(5), 68–78.

Goodenough-Trepagnier, Cheryl, Rosen, Michael J. and Galdieri, Beth. 1986 (June 23-26). Word Menu Reduces Communication Rate. *Pages 354–356 of: Proceedings of the Ninth Annual Conference on Rehabilitation Technology*. RESNA, Minneapolis, MN.

How, Yijue. 2004. *Analysis of SMS Efficiency*. Unpublished Undergraduate Thesis. National University of Singapore.

John, B. E., Rosenbloom, P. S. and Newell, A. 1985. A theory of stimulus-response compatibility applied to human-computer interaction. *Pages 212–219 of: Proceedings of CHI, 1985*. New York: ACM.

McCoy, K.F., Demasco, P.W., Jones, M.A., Pennington, C.A., Vanderheyden, P.B. and Zickus, W.M. 1994. A Communication Tool for People with Disabilities: Lexical Semantics for Filling in the Pieces. *In: In Proceedings of ASSETS '94*.

Moffat, A. 1990. Implementing the PPM data compression scheme. *IEEE transactions on Communications*, 38(11), 1917–1921.

Mohri, Mehryar, Pereira, Fernando and Riley, Michael. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1), 17–32.

Newell, A. and Rosenbloom, P.S. 1981. Mechanisms of skill acquisition and the power law of learning. *Pages 1–55 of:* Anderson, J. R. (ed), *Cognitive Skills and Their Acquisition*. NJ: L. Erlbaum Associates.

Pereira, Fernando C. N. and Riley, Michael. 1997. Speech Recognition by Composition of Weighted Finite Automata. *In:* Roche, Emmanuel and Schabes, Yves (eds), *Finite-State Devices for Natural Language Processing*. Cambridge, MA: MIT Press.

Rowe, Neil C. and Laitinen, Kari. 1995. Semiautomatic Disabbreviation of Technical Text. *Information Processing and Management*, 31(6), 851–857.

Shannon, Claude. 1951. Prediction and Entropy of Printed English. *Bell System Technical Journal*, 30, 50–64.

Shieber, Stuart M. 1996. A Call for Collaborative Interfaces. *Computing Surveys*, 28A (electronic).

Shieber, Stuart M. and Baker, Ellie. 2003 (January 12-15). Abbreviated Text Input. *Pages 293–296 of: Proceedings of the 2003 International Conference on Intelligent User Interfaces*.

Soede, Mathijs and Foulds, Richard A. 1986 (June 23–26). Dilemma of Prediction in Communication Aids and Mental Load. *Pages 357–359 of: Proceedings of the Ninth Annual Conference on Rehabilitation Technology*. RESNA, Minneapolis, MN.

Stum, Gregg M. and Demasco, Patrick. 1992. Flexible Abbreviation Expansion. *Pages 371–373 of:* Presperin, J.J. (ed), *Proceedings of the RESNA International '92 Conference*. RESNA, Washington, D.C.

Tanaka-Ishii, Kumiko, Inutsuka, Yusuke and Takeichi, Masato. 2001 (March). Japanese Input System with Digits: Can Japanese be Input Only with Consonants? *In: Proceedings of the Human Language Technology Conference*.

Vanderheiden, Gregg C. and Kelso, David P. 1987. Comparative Analysis of Fixed-Vocabulary Communication Acceleration Techniques. *Augmentative and Alternative Communication*, 3(4), 196–206.

Ward, D. J. and MacKay, D. J. C. 2002. Fast Hands-free writing by Gaze Direction. *Nature*, 418(6900), 838.

Willis, Tim, Pain, Helen, Trewin, Shari and Clark, Stephen. 2002. Informing Flexible Abbreviation Expansion for Users with Motor Disabilities. *Pages 251–258 of: ICCHP*.