Natural Language Engineering **13** (1): 51–74. © 2006 Cambridge University Press
doi:10.1017/S1351324905004080 First published online 15 February 2006 Printed in the United Kingdom

Word-based predictive text entry using adaptive language models

KUMIKO TANAKA-ISHII

Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan
e-mail: kumiko@i.u-tokyo.ac.jp

(*Received 10 September 2004; revised 26 August 2005*)

Abstract

The recent scaling down of mobile device form factors has increased the importance of predictive text entry. It is now also becoming an important communication tool for the disabled. Techniques related to predictive text entry software are discussed in a generalized, language-independent manner. The essence of predictive text entry is twofold, consisting of (1) the design of codes for text entry, and (2) the use of adaptive language models for decoding. Code design is examined in terms of the information-theoretical efficiency. Four adaptive language models are introduced and compared, and experimental results on text entry with these models are shown for English, Thai and Japanese.

1 Introduction

Recent technological advances have enabled various novel means of information entry, such as character or speech recognition. Still, entry with a keyboard remains dominant because of its ease of implementation and its utility, in terms of higher speed, smaller error rate, privacy, and so forth. While various text entry methods can be used even with a keyboard, my concern in this paper is *predictive text entry*.

As considered in this paper, predictive text entry is defined as the process of text entry in three stages:

Stage 1. The user enters a *coded* character sequence of the target text.
Stage 2. The text entry system looks for corresponding target candidates in a dictionary included with the software. It sorts the candidates with regard to the context and displays them for the user.
Stage 3. The user chooses his preferred target from among the candidates.

Japanese kana-kanji conversion, for example, is a predictive text entry method in which the codes are kana and the target words are kanji-based words. Candidates are usually longer than a word length. I use the terms *coding* and *decoding* here, and indeed, such text entry tasks can be regarded as encoding/decoding tasks comparable to text compression. The difference is that the codes here are intended for human use.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

Historically, such predictive text entry systems have been more common in East Asian countries whose languages utilize thousands of characters. The problem with these languages is that the number of characters that must be handled greatly exceeds the number of keys on a keyboard. As a result, predictive text entry was invented to enable the projection of a wide range of characters with a limited number of keys. It is currently the major method of solving this problem.

The problem of too many characters for too few keys has become more international as smaller machines have been developed. Current mobile devices, for example, can be as small as a wristwatch (IBM, 2001), so the number of characters in any language will exceed the number of buttons available for input. As a result, predictive text entry has started to be discussed internationally, in both the academic and commercial domains (Tanaka-Ishii *et al*., 2002; James and Reischel, 2001; Isokoski, 2001; Rau and Skiena, 1994; Tegic 9, 2000; ZI-Corporation, 2003; Sugimoto and Takahashi, 1996; Masui, 1999; MisawaHome, 2002). These systems are overviewed elsewhere (Tanaka-Ishii, 2005a; Masui, 2004).

Another potential application of predictive text entry is in interfaces for the elderly and the disabled. Construction of universal technology is a substantial field with its own long history. Focusing on predictive text entry in this domain, it is well known that keyboard operation is the greatest hurdle to the use of computers by the elderly. The situation is even worse for people with disabilities that limit their mobility or the use of their hands. Today, more and more opportunities for communication are becoming available through predictive text entry. Harbusch and Kuhn (2003) report how it has expanded the communication possibilities for the disabled. Similar work is reported by Jones (1998) in application of predictive text entry for the disabled. There also are commercial products available for both English (Origin Instruments, 2005) and Japanese (Pete, 2005).

In general, the difficulty of predictive text entry can be measured in terms of ambiguity. The more the codes are allowed to be ambiguous, the more difficult the entry process will be. For example, current devices with only four buttons have a high degree of ambiguity. Therefore, the essence of predictive text entry is twofold. The first issue is the design of the codes to be entered during Stage 1. If the codes are badly designed, the entry task will become more ambiguous, and the entry load will be high. The second issue lies in the decoding: how to make the system better predict the user's preferred target from among various candidates.

These two problems can be tackled through an information-theoretical approach, as has been done in other corpus-based statistical language modeling studies. This paper can thus be situated as an application of long-studied statistical language models to predictive text entry systems. My aim is to give an overall picture of the problems involved in predictive text entry, which has not yet been done in the language technology domain. In particular, I emphasize the functioning of adaptive language models when they are applied to predictive text entry. In this paper, I compare four adaptive models, which are based on concepts originally proposed in the text compression domain (Bell *et al*., 1990) and are similar to cache models originally proposed in the speech domain (Kuhn and De Mori, 1990). Although my research approach is thus based on information theory, our second objective is to

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

provide a realistic view of the text entry task, in terms of the number of candidates, candidate ranking, the required number of keystrokes, and actual speed. Also, we have applied the same models to three completely different languages: English, Thai and Japanese.

I first give the background of this field and one example of a predictive text entry system. Then, I proceed to the major discussion in this paper: the design of codes and the use of adaptation methods in language models. Finally, I show experimental results with a predictive text entry system used in English, Japanese and Thai.

2 Predictive text entry

2.1 Background

As noted in the Introduction, I consider predictive text entry methods in terms of a three-stage text entry process aided by computer software. There are related domains in computational linguistics that also employ encoding/decoding models: text compression, and language technology concerning speech recognition and natural language processing (Manning, 1999). In the former case, the process of encoding/decoding is done by machine. The codes are designed for machines and consist of bits, and the main issue is optimizing the codes. Predictive text entry, in contrast, must be designed for humans, who encode the target text and enter the coded text into a computer. Thus, the codes should be both user-friendly and efficient.

As for the latter case of the language technology domain, it has become common to employ the noisy channel model to build statistical NL applications. This model also applies to predictive text entry: if the user makes a mistake in encoding the target text or entering the code, the resulting situation will fit the noisy channel model. In fact, some researchers have tried to correct the errors occurring during text entry: one group (Goodman *et al*., 2000) studied automatic correction of spelling errors in English, while my group (Iwasaki and Tanaka-Ishii, 2004) studied the same problem in Japanese. Most current predictive text entry systems, however, assume that the codes are entered correctly. In this paper, too, I assume that the text is encoded and entered without error.

Thus, the problem I discuss in this paper is very similar to the problems of text compression and speech recognition, and therefore, many of the solutions developed in those domains can also be applied to predictive text entry. These solutions include language models and tools from information theory.

The main stream of predictive text entry systems began in the 1960s in China and Japan, where thousands of characters are used to represent text. As far as I know, the earliest work on this topic was reported in the 1960s (Kurihara and Kurosaki, 1967) in Japan, but serious work on implementation started in the late 1970s. The codes used in this field are based on either phonetics or the shapes of characters. Phonetically, the codes consist of kana in Japanese (JustSystem, 2005) (MS-IME, 2005) and pinyin in Chinese (Chinese Input Methods, 2005). This type of text entry is now popular with native speakers of both languages. The other, less popular, type of code based on character shape is formed by decomposing characters into

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

components, as in ChDrv (2005) in Chinese or the *bushu* (i.e., the main part of each ideogram) input method in Japanese. This approach is only possible in languages whose characters are ideograms, so it is not applicable to languages using phonetic characters, such as English. Although this type of code was part of a significant body of research in the late 1980s, it is now used more often only to complement phonetic text entry. As almost all predictive text entry systems by phonetics are word or phrase-based, the discussion in this paper focuses on entry methodologies based on words.

Curiously, the same idea of predictive text entry was also devised by several researchers in regions whose languages contain only a small number of characters. For example, Darragh and Witten (1992) proposed predictive entry methods for English in the 1980s as a typing aid. The idea of predictive entry emerged internationally with the increasing popularity of various mobile devices, along with the development of improved technology for the disabled.

Although predictive text entry is thus becoming an important, world-wide human language technology, no complete discussion of code design and decoding techniques has yet appeared in the domain of human language technology. I thus discuss here the information-theoretic methodologies that have been applied to solve various problems related to predictive text entry.

2.2 Example using only four keys

To give a clear illustration of predictive text entry, I next show an example of a recent text entry system, TouchMeKey4, which was developed by my group and encodes English with only four digits. The system is designed for tiny mobile devices (as small as a wristwatch (IBM, 2001)) with only four buttons. It could also be applied to text entry systems for the disabled and the elderly.

Figure 1(1) shows the GUI of the TouchMeKey4 keypad. TouchMeKey4 now works in French, Thai, English, and Japanese; here, I show an example in English. Nine buttons are visible, with four on either side of the central boxes and a "quit" button at the bottom right. In this paper, I am only counting those buttons that are actually used to enter characters: that is, the four on the right-hand side. For TouchMeKey4, we also imposed the constraint that the buttons may only be pressed one at a time, because our interest lies fundamentally in the text entry depending on the number of keys. The reason is that the inclusion of key-chords combining multiple keys effectively increases the number of buttons available at the key-encoding level (for example, four key entry can be realized by two buttons when adopting pressing keys by chord).

Either six or seven letters of the alphabet are assigned to each button. Button Nos. 1–4 have the characters "abcdef", "ghijkl", "mnopqrs", and "tuvwxyz", respectively. The equivalent lowercase and capital letters are assigned to the same buttons. All other, non-alphanumeric ASCII characters are assigned to button No. 4. Suppose that we have just entered the string "human language". The text appears in the upper text box in the middle of the window (Figure 1(1)). We now want to enter the word "technology". Words are entered through a *single-tap-per-character*

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

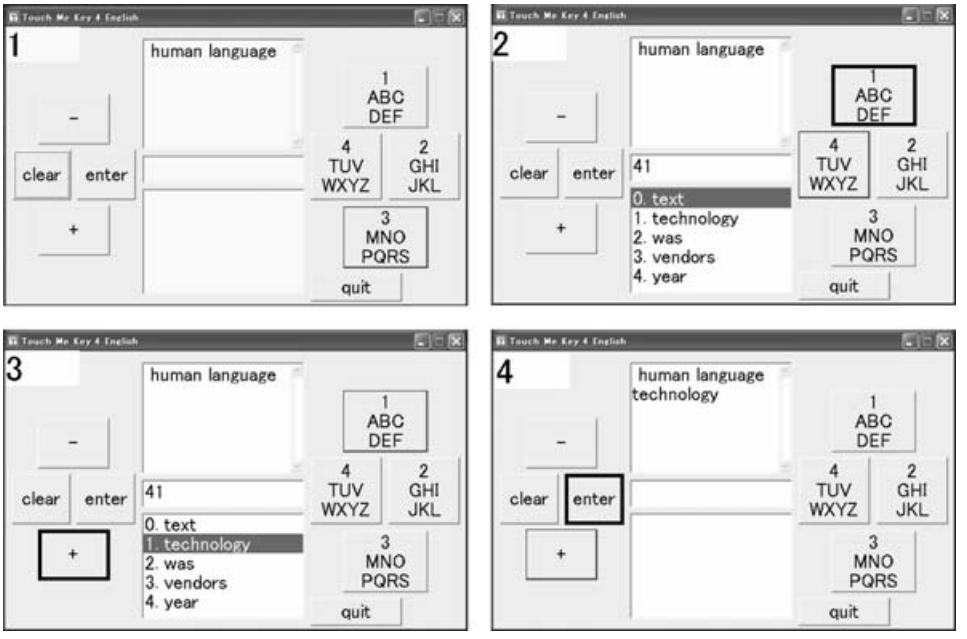


Fig. 1. Entering the word "technology" with the TouchMeKey4 keypad.

form of predictive entry; that is, a button is only *pressed once* to enter a character. For example, button No. 4 is pressed once to enter the "t" of "technology". To enter the subsequent "e", button No. 1 is pressed once.

After button No. 1 is pressed, the TouchMeKey4 window appears as shown in Figure 1(2).[1] Two changes have occurred, as seen by comparison with Figure 1-1. The first is that the text "41" appears in the text box in the middle of the window. This indicates the keys that the user has just pressed. The second change is that some words have appeared in the lower box in the middle of the window (a type of list box that we call the "candidate box"). These words are the candidate words corresponding to the user's input string, "41".

Each button press causes the TouchMeKey4 system to automatically search its dictionary for candidates. The candidates include both longer words and words of the same length as the entered sequence of digits, if such words exist. Thus, in this example, the candidates include all words that begin with one letter from the sequence "tuvwxyz" (button No. 4), followed by one letter from the sequence "abcdef" (button No. 1). In this case, "text", "technology", "was", "vendors", and "year" are all candidates.

The various candidates are sorted into a certain order before being placed in the candidate box and shown to the user. The order is based on the word probability as determined by an adaptive language model, PPM (prediction by partial match), that uses a base dictionary and a user corpus (A detailed description is given in Section 4.3). In this example, the base dictionary was constructed from one year of

[1] Note that the most recently pressed button is framed by a thick line in this illustration.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

the Wall Street Journal (WSJ) and contains around 93,000 different words, while the user corpus came from a computer magazine and contains 10,000 words. The nature of this particular user corpus is the reason why the relatively uncommon word "vendors" appears among the top five candidates (Figure 1(2)).

Our target word "technology" appears as the second-ranked candidate. To select this word, the user highlights it by pressing the "down" button (labeled as "+") on the left-hand side of the window (Figure 1(3)), then pressing the "enter" button[2]. It can be seen that the selected candidate finally appears in the upper text box (Figure 1(4)).

Suppose that we do not apply the generalization of a single-tap-per-character system. Then we need to enable the exact entry of individual characters via the four buttons. One method of this type involves assigning an order to the characters on each key; a key is then pressed i times to obtain the ith character. This is typical of the default text entry methods for mobile phones (i.e., multiple-tap methods). There are two problems with this method, however. First, the user often must press a key numerous times to obtain a single target character (in our four-key example, button No. 3 must be pressed seven times in order to enter an "s"). Second, there is an ambiguity in the user action when two characters assigned to the same button must be entered one after another ("aa", requiring button No. 1 to be pressed twice, could also be interpreted as "b"). This situation requires the use of an escape. By contrast, TouchMeKey4 employs a far simpler method.

This instead raises the problem of dictionary deficiency, experimented in such studies as Gutowitz (2003) and MacKenzie *et al.* (2001). MacKenzie reported that when the fraction of non-dictionary words is greater than approximately 15%, predictive entry methods are slower than multi-tap methods. Gutowitz conducted interviews with 230 people who had experience with both predictive and conventional entry methods. He found that this problem of dictionary deficiency become the reason for not using predictive entry methods. Nevertheless, there are methods of overcoming this problem, as will be summarized in Section 6. Also, Japanese and Chinese people are forced to use predictive entry systems because of the characteristics of their languages, and these two cultures represent one of the largest computer-using communities in the world. Therefore, our group believes in the possibility of world-wide use of predictive text entry on a larger scale.

In the next two sections, I discuss codes and decoding models from a general point of view.

3 Coding

3.1 Code design

Two major types of codes are used in predictive text entry systems. The first type is based on the *substrings* of words and phrases. Shieber and Baker (2003) recently

[2] The user can also select the target word by simply clicking on it, rather than by pressing the "down" button. However, this feature was disabled in the user evaluation described later in this paper.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

studied this idea in general. For example, "intl" can be expanded to "international" or "intentional", while "an exmpl of 5 wrds" can be expanded into "an example of 5 words". There is still, however, the question of entry efficiency in this general method, and the *prefix* is used as the substring in most such proposals. Given a user input, the system searches for words with the corresponding *prefix* entered by the user. This method allows text entry to be performed with a small number of keystrokes. It is thus becoming more and more common with the growing popularity of small devices with small buttons that users want to press as little as possible. This tendency to minimize the user action is particularly important in the area of communication aids, as well. In this paper, prediction by prefix is considered because it is the most prominent method among those with codes based on substrings.

The second type of code is based on *degeneration*. For example, the alphabet mapping to digits on mobile phones (Tegic 9, 2000) employs a rule that transforms all occurrences of 'a', 'b' and 'c' into '2'. The major task of code design by degeneration is to define a set of rules for transforming a character set of large size into a smaller number of codes. Because the codes are for human use, they are first designed based on some linguistic feature, such as representing a language by consonants, with further degeneration to digits (Tanaka-Ishii *et al*., 2000).

The recent scaling down of mobile device form factors has made this problem more general, and various rules have been invented. In this paper, I consider this degeneration of characters in terms of mapping characters onto as few as two to four buttons.

Code designs for predictive text entry can obtain hints from human codes that have been in use since long before the development of computers. Regarding codes based on substrings, encoding by prefix frequently occurs in any language as abbreviation. The cognitive emphasis on prefixes is supported by the Bath-Tub model (Aitchison, 1994), which shows that human memory of spelling differs according to the position in a word: the prefix of a word is the most well memorized part, followed by the postfix, with the middle the worst memorized part.

Regarding degeneration, the contrast between consonants and vowels in language is seen throughout human history: the most notable example is the Semitic languages, which people once wrote using only consonants. Other examples include the common practice of eliminating vowels in creating abbreviations or program identifiers. Therefore, predictive text entry can be regarded as one application of long-existent human codes.

3.2 Efficiency of codes

The design of codes governs the efficiency of predictive text entry. For example, when using four keys, if all the characters from "a" to "w" were assigned to the first key, and "x", "y", and "z" were assigned to the second, third, and fourth keys, respectively, then the word prediction performance would clearly be poor. Even when such extreme possibilities are eliminated, the problem of defining the code rules remain, because many plausible rules are possible. We thus need to be able to measure the performance of a given code.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

The efficiency of the first type of code, based on encoding by prefix, can easily be measured. Trivially, the shorter the prefix, the more difficult the text is to predict.

On the other hand, the efficiency of the second type of code, based on generalization, cannot be determined trivially. One of the easiest measures is the number of candidates that exist for each code, but a better one is the average ranking of the target. An alternative measure is the conditional entropy. Given a code sequence, C, and a target word, W, the task of a predictive text entry system is to estimate W from C as accurately as possible. This can be measured by the *average conditional entropy*, $H(W|C)$ (Jelinek, 1997). The definition of this quantity is given by

$$\begin{aligned} H(W|C) &= \Sigma_{w,c} P(C=c) H(W=w|C=c) \\ &= -\Sigma_{w,c} P(C=c, W=w) log P(W=w|C=c), \end{aligned} \tag{1}$$

where $P(C=c)$ is the probability of input sequence c, and $P(W=w|C=c)$ is the conditional probability of a word for the given sequence c. When the estimation of W is less certain, $H(W|C)$ has a larger value. In other words, the greater the entropy, the more uncertainty there is in estimating the correct word. Therefore, the conditional entropy is suitable as a method for evaluating these codes.

Therefore, the conditional entropy can be utilized as an "evaluation function" for determining the optimal key assignment. Given a total number of code characters N_K and the number of alphabet characters A, however, the total number of possible assignment schemes is N_K^A. For example, with $N_K = 4$ for English (as in Figure 1), this amounts to 4^{26} possibilities. As such numbers quickly reach astronomical levels, we usually cannot find the optimal assignment. One way to tackle this problem is to utilize an approximation method, such as annealing. For example, Zhai *et al.* (2000) successfully applied annealing to find a better placement of English alphabet characters on a 2-dimensional plane, in order to create a better alphabetic keyboard.

One other factor we need to consider at this point is that codes must be chosen appropriately with respect to the characteristics of each language system. The English alphabet, for example, has a commonly known order. If this order was neglected and the letters "ajxgukh" were assigned to a code, it would be difficult for beginners to transform text into the correct codes, although this could potentially be more efficient for a well-trained user.

Keeping all this in mind, we examined coding sets for English by varying the number of code characters from 10 to 2. The resulting code efficiencies are listed in Table 1 in terms of the conditional entropy, average ranking, and number of candidates. The efficiencies were calculated for unigrams from the Wall Street Journal (WSJ, with around 93,000 different words obtained from around 30 MB of data).

The first column lists the total number of code characters(N_K). The second column gives labels for each code with a given number of code characters. In the third column, the text characters assigned to the respective code characters are grouped in parentheses. For example, 4-A indicates an assignment method using four code characters, with "abcdef" assigned to the first key, "ghijkl" to the second, "mnopqrs" to the third, and "tuvwxyz" and other ASCII symbols to the fourth.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

Table 1. *Possible codes and their conditional entropies*

N_K	Label	Groups of characters	Conditional entropy	Average ranking of target	Average number of candidates
10	–	(S_0)(S_1)(abc)(def)(ghi)(jkl) (mno)(pqrs)(tuv)(wxyz)	0.5	1.2	5.6
5	A	(abcdef) (ghijkl) (mnoS_0) (pqrsS_1) (tuvwxyz)	0.7	1.3	11.7
5	B	(S_0S_1) (abcdef) (ghijkl) (mnopqrs) (tuvwxyz)	1.1	1.6	20.8
5	C	(S_1mno) (abcpqrs) (deftuv) (ghiwxyz) (jklS_0)	0.8	1.4	15.1
4	A	(abcdef) (ghijkl) (mnopqrs) (tuvwxyzS_0S_1)	1.2	1.6	23.5
4	B	(S_0abc) (defghi) (jklmnopqrs) (tuvwxyzS_1)	1.2	1.7	26.2
4	C	(S_1jkl) (abcmno) (defpqrstuv) (ghiqxyzS_0)	1.5	2.2	42.4
3	A	(S_0abcdef) (ghijklmno) (pqrstuvwxyS_1)	1.5	2.1	45.4
3	B	(S_1ghipqrs) (abcjkltuv) (defmonqxyzS_0)	1.5	2.1	47.4
3	C	(S_1jklpqrstuv) (abcdefmno) (ghiwxyzS_0)	1.8	2.8	66.5
2		(S_0abcdefghijkl) (mnopqrstuvwxyzS_1)	2.9	6.5	180.6

Capital letters are assigned to the same keys as the corresponding lowercase letters. S_0 and S_1 indicate non-alphabetical ASCII symbols.[3] Note that 4-A corresponds to the TouchMeKey4 system shown in Figure 1(1). The groupings with the label C are alphabetically assigned more randomly than the other two methods for each number of code characters. The remaining columns list the three measures of code efficiency.

In general, the efficiency increases as the number of keys increases. This is a readily comprehensible result: a larger number of keys simplifies the task of estimation, making it less uncertain. By comparing the values for assignment methods using the same number of keys, we can see that the scores differ considerably. For example, the entropy of 5-B indicates greater uncertainty than with the other 5-x codes. We conducted similar tests in Thai and Japanese and used the resulting efficiency values to select appropriate key assignment sets (Inutsuka *et al.*, 2001; Tanaka-Ishii *et al.*, 2003b) for each language.

[3] These symbols are categorized into two groups, according to the scheme used on mobile phones.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

4 Decoding: Adaptive language models

4.1 Adaptiveness

Text entry systems are designed for personal use. As personal computers have become more widely distributed, they are now rarely shared among multiple users. This is even more true for mobile devices. Therefore, the language model used for predictive text entry should be adaptive. At the same time, this also provides one solution to the problem of high ambiguity.

Before we examine the actual adaptive models, we must first consider the characteristics of personal documents.

4.2 Characteristics of personal documents

There are two important characteristics in handling user documents:

- The total size of a uniform user document is small.
- The vocabulary size in a user document is small.

First, the total size is easily considered. For example, the total volume of e-mail that I sent over a year contained about 150,000 words, at most. The total number of words in a typical textbook (Bird and Wadler, 1990) is around 80,000. In contrast, newspapers easily exceed the level of a million of words per year: for example, Wall Street Journal contains over five million words each year. Therefore, the size of the existing corpus and that of a personal document are clearly not commensurate. This means that by employing only a user corpus, we cannot obtain a sufficient amount of data to construct a good language model.

Second, we examined the numbers of different unigrams and bigrams present in sets of 10,000 words contained in several types of personal documents, including chat text, manuals, patent text, and stories, and we compared these values with the numbers obtained from sets of 10,000 words of newspaper text. We found that the personal documents all contained less than 2000 different words, whereas the newspapers contained from 2300 to 2500 different words. One patent text even had less than 1000 different words. In addition, the number of different bigrams in the personal documents ranged from 4000 to 6000, whereas the newspapers had around 7000 different bigrams. Although the numbers of bigrams for the news corpora were not of a different order, the vocabulary sizes in the personal documents were clearly smaller.

This tendency toward a small vocabulary in a user corpus is good for adaptiveness, because the smaller the domain size and the vocabulary, the smaller the language model becomes, making estimation easier. This does not mean, however, that a user's overall vocabulary is small. Each person has a large general language model, and only part of it is used to communicate in a specific context. Therefore, we should not construct a language model only from user documents, which would be difficult anyway because of their small sizes. Instead, we should use two corpora at the same time:

- A large general corpus for developing the static initial language model (static).
- A user document for adapting the initial language model (dynamic).

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

My interest here is on the dynamic model rather than the initial static language model, because various language models proposed in conventional natural language processing (Manning, 1999) can be adopted as the initial model.

I next discuss four different adaptive language models that implement these ideas, starting with the easiest model and proceeding to the most complicated.

4.3 Four adaptive language models

As observed previously, adaptive language models have to respect the characteristics of users' texts. From this point of view, we require an approach that modifies the fundamental core language model based on a user's recent context. Adaptive models for this purpose should be dynamically updatable. For example, each time a user selects a candidate, the language model should be updated according to his action. This is because the size of the user corpus increases with use, so the model should be updated with each entry.

Models that fit our need have been proposed in two domains: language technology and text compression. In the former case, the major works were developed after corpus studies became more popular. One representative model is the cache model (Kuhn and De Mori, 1990), which reflects patterns of words kept in a cache component for a short term.

In the text compression domain, on the other hand, such models have a long history, starting from the naïve Move To Front (MTF) and continuing through PPM (Cleary and Witten, 1984; Bell *et al.*, 1990).

Here, among all these approaches, I compare four statistical adaptive models of increasing complexity, based on unigrams, MTF, co-occurrence, and n-grams, in order. Briefly, the first two models concern only a user's previous actions, whereas the latter two also consider the surrounding words as context at the same time as a user's action.

Unigram

The simplest of the four adaptive models is based on simply updating the importance score registered in the dictionary. For example, when the word *technology* is selected in Figure 1, its importance in the initial language model is incremented to a higher value.

This model includes some variance in terms of how to update the importance score. For the model as utilized in this paper, I update the unigram count every time the user chooses a word.

Move To Front

MTF was also originally utilized in the text compression domain (Bell *et al.*, 1990). It has been applied to many commercial text entry systems such as JustSystem (2005), where the most recent candidate selected by the user for a given input becomes the best candidate. After the user selects the nth best candidate ($n \neq 1$) in Stage 3, it

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

will be proposed as the best candidate when the same codes are entered the next time. For example, in Figure 1, the user entered "41", and the word *technology* was proposed as the second best candidate. Because the user selected *technology*, it will become the best candidate the next time he enters "41". The MTF model was first proposed in the field of text compression as a primitive adaptive method. Note that MTF is related to the unigram model, in that the maximum importance score is given to a candidate selected by the user.

This model works well when the degree of ambiguity is low. When it becomes large, however, the user frequently has to change preferences according to the context. For example, if the user is writing a text involving the words "technical" and "technology", these two words will both appear among the candidates when the user entry is "tech". If the user selects "technology", then the next time the user enters "tech", the system will propose "technology" as the best candidate, even though the user might want to enter "technique". As a result, the user quite often has to change his preference. To overcome this problem, the model must take the context, the surrounding words into account. This feature is better integrated in the following two models.

Adaptive co-occurrence

One way to consider the context is to utilize the co-occurrence. Suppose that the user has already entered the words

$$C = w_1, w_2, w_3, \ldots w_{n-1}, \tag{2}$$

and that the system is now estimating the next word, w_n. Given a window size, k, that defines the context, the best estimation will be

$$\hat{w_n} = argmax_{w_n} P(w_n | w_{n-k}, \ldots, w_{n-1}). \tag{3}$$

From Bayes' Theorem, $P(w_n|w_{n-k}, \ldots, w_{n-1})$ can be transformed into the following:

$$\hat{w_n} = argmax_{w_n} P(w_{n-k}, \ldots, w_{n-1}|w_n) P(w_n). \tag{4}$$

Estimating k words occurring at the same time requires a large corpus, which is a problem, since the user corpus is small. Therefore, we can approximate the right-hand side of the equation as follows:

$$\hat{w_n} \approx argmax_{w_n} P(w_n) \prod_{i=n-k}^{n-1} P(w_i|w_n). \tag{5}$$

If co-occurrence is not observed in the user corpus, w_i and w_n are assumed independent, and $P(w_i|w_n)$ is approximated by $P(w_i)$. If neither word is observed in the user corpus, the value in the initial dictionary is utilized.

Adaptive n-gram (PPM)

The last and most sophisticated model is based on the adaptive n-gram. This model incorporates both the unigram model and the co-occurrence model. As mentioned

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

at the beginning of this section, similar models, namely the cache model and PPM, were proposed in the domains of both language technology and text compression, with the major difference being only the weighting method. From a comparison of the cache model and PPM, I have chosen PPM (Bell *et al.*, 1990) for this work, as it is more general and the study emphasized the problem of defining the weighting parameters from user corpora.

PPM was originally proposed to make use of higher-order context prediction in arithmetic coding. It has been introduced successfully in user interfaces, as described in Ward *et al.* (2000), and it has also been applied to language analysis methods, such as Chinese segmentation (Teahan *et al.*, 2000).

Broadly, PPM interpolates the n-gram counts in a user corpus with initial statistics, denoted as $P_{init}(w_i)$. The following formula can be used to estimate the probability, $Pr(w_i)$, for the ith word, w_i:

$$Pr(w_i) = P_{init}(w_i) + P_{uc}(w_i) + \sum_{k=1}^{kmax} u_k P_{uc}(w_i|w_{i-k} \dots w_{i-1}). \tag{6}$$

In the original PPM model for text compression, P_{init} was set to a unique value (1/N, where N is the total number of elements to be coded). When applied to the situation here, where there are both a large corpus and a small user corpus, P_{init} can be regarded as the estimation obtained from the large initial corpus, while the P_{uc} are obtained from the user corpus. For P_{init}, we use simple word unigrams acquired from the large corpus, although this can be extended to higher order language models. The P_{uc} are calculated as follows, where N_{uc} is the total occurrence of words in the user corpus:

$$P_{uc}(w_i) = \frac{freq(w_i)}{N_{uc}} \tag{7}$$

$$P_{uc}(w_i|w_{i-k} \dots w_{i-1}) = \frac{freq(w_{i-k} \dots w_i)}{freq(w_{i-k} \dots w_{i-1})}. \tag{8}$$

Finally, u_k is a weighting probability that is multiplied by $P_k(w_i)$ to obtain the final $P(w_i)$. Among many methods of defining u_k (Bell *et al.*, 1990), we have chosen PPMA, PPMB with exclusion, PPMC with exclusion, PPMZ (Bloom, 1998; Bell *et al.*, 1990) and PPMII (Shkarin, 2002).

Using HMM with adaptive models

There is another family of language models: statistical language models concerning meta-level word chains, using the hidden Markov model (HMM). The basis of HMM is also an n-gram model, but it includes multiple levels by considering n-gram chains below the surface word level. Therefore, HMM can naturally be extended to make it adaptive both at the surface level and at deeper levels, along with any of the adaptive models described above.

According to our group's previous work (Tanaka-Ishii *et al.*, 2001), in which we integrated PPM with HMM, n-gram adaptation at the deeper part of speech (POS) level does not work very well, no matter which adaptive model is employed. This can be understood from the fact that a POS chain is less variant with respect to

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

Table 2. *Data used for evaluation*

	Japanese	English	Thai
Initial corpus	Mainichi Newspaper (30 MB)	Wall Street Journal (30 MB)	NECTEC (20 MB)
User corpus	E-mail (300 KB from one year of e-mail, or 1.5 MB)	Computer magazine (300 KB of text from 30 MB)	Stories (2 MB of the NECTEC corpus)

different users. Conversely, the alternate approach of integrating PPM into HMM by adapting a surface-level chain does work, although the major benefit of adaptation is obtained through PPM.

HMM is thus a model operating on a different axis when adaptive models are concerned, and it can coexist with any of those models. Since the major gain will come from adaptation at the surface word chain level, I have left HMM out of the evaluation portion of this paper.

5 Evaluation

5.1 Settings

I next present various evaluation results, from actual experiments conducted in English, Japanese, and Thai. We chose these languages for the following reasons:

- English is the international language and is representative of Indo-European languages.
- Japanese is one of the few languages for which predictive text entry systems are widely used.
- Thai has about 80 characters, so the number of people who learn the locations of all characters on a full keyboard by heart remains limited; as a result, predictive text entry methods are being actively studied.

The data used for the evaluations described in this paper are listed in Table 2. The second row indicates the large corpora used to construct the initial language model for each language. These were based on the Wall Street Journal (30 MB) for English, the Mainichi Newspaper (30 MB) for Japanese, and the NECTEC corpus (22 MB, with 2 MB selected for learning and testing) for Thai. Note that both Japanese corpora were segmented at the word level by using chasen (Chasen, 1997), while the Thai NECTEC corpus was originally segmented through distribution. The unigram counts were extracted in order to build the initial models.

Although my group experimented with other user corpora in both Japanese and English (Tanaka-Ishii *et al*., 2002; 2000), in this paper I only refer to those listed in Table 2, as they are sufficient to demonstrate the most important results. Unless noted otherwise, the corpora given in Table 2 were the default user corpora in the evaluations.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

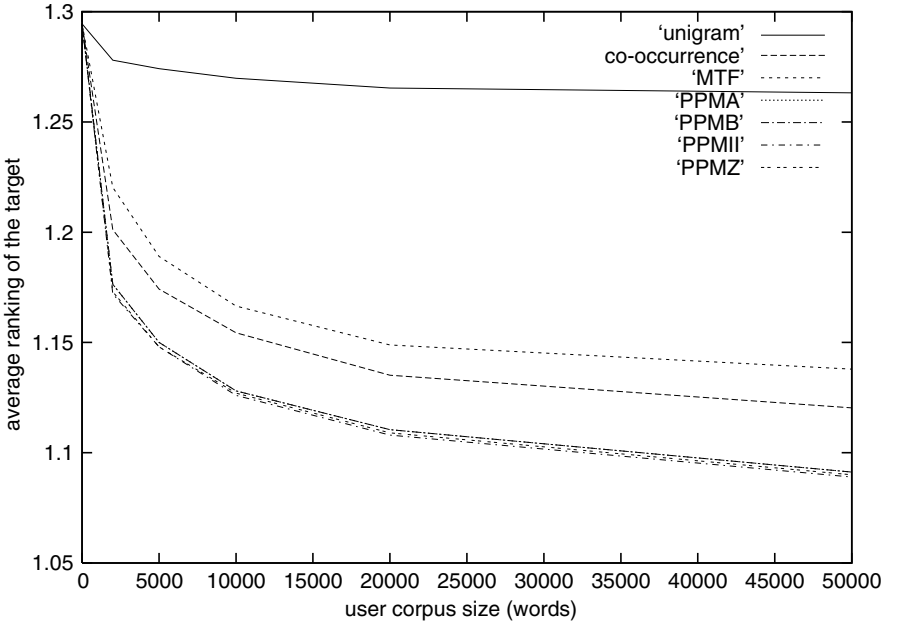


Fig. 2. Comparison of four adaptive language models.

Usually, a small part of a text focuses on one topic and has a local bias. If we employ a user corpus as it is for evaluation purposes, the results will change according to the part that we use. Thus, we separated the user corpora into sentences, then re-ordered the sentences randomly. Next, learning corpora of various sizes were cut out incrementally, so that the larger user corpora contained smaller ones. At the same time, test texts were cut from different parts of the user corpora and used to measure the efficiency of adaptation.

In the next two sections, I compare the four adaptive language models and their entry efficiencies in terms of adaptiveness with a user document. Although the efficiency can be measured in terms of the entropy, I instead measure it here by using more realistic values: the ranking of the preferred target and the number of keystrokes.

The experiments thus involved many parameters: the language (English, Japanese, or Thai), the model for adaptation (MTF, unigram, co-occurrence, or PPM), the type of code (10, 5, 4, 3, or 2 keys; and in Japanese, kana), and the size of the user document (ranging from 5k to 50k words). I focus here on the key trends by only showing representative and significant results.

5.2 Comparison of adaptive language models

I first examine the differences among the adaptive language models.

The results are shown in Figure 2. The horizontal axis indicates the quantity of the user corpus that was learned, while the vertical axis shows the ranking of the target candidate. The language here was Japanese, and the other parameters were

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

set to kana for the code and prediction by word-based entry, without prediction by prefix.

The several lines gathered together with the lowest rankings (i.e., closer to a ranking of 1) indicate that the PPM models outperformed all the other models. From best to worst, the models performed in the order of PPM, co-occurrence, MTF, and unigram. This result was somewhat expected, given that PPM remains one of the best adaptive language modeling approaches. Among the various PPM models, only very slight differences were seen. These models' overall performance correlated with the experiments reported in the text compression domain for testing with texts (Bell *et al*., 1990; Shkarin, 2002).

Experiments using other parameters consistently produced similar results: PPM always performed the best, and the performance differences were small when the ambiguity was low. As a result, assuming PPM as the language model, I next examine these performance differences.

5.3 Comparison of codes

Next, I discuss the results when the codes were varied, with PPM set as the adaptive model. Given that many predictive text entry systems adopt prediction by prefix (Masui, 1999; ZI-Corporation, 2003), and that the number of keystrokes per word is the least with prefix prediction (Tanaka-Ishii *et al*., 2000), I assume the use of that feature here as well.

Given prediction by prefix, the ranking of the target differs according to the user entry method. Therefore, my group evaluated the entry efficiency in terms of the number of keystrokes per word, as follows. The number of keystrokes is the sum of the keystrokes required for the input and selection operations. The keystrokes for selection were counted as n when choosing the nth candidate. In predicting words, there are multiple points at which the target word can be chosen. For example, in Figure 1, the word "technology" appears as the second best choice after the user has typed in "41". The user can thus either select the target at this point or type in another "1" to indicate the "c" and increase the target's rank. The automatic routine only chooses the target after it has appeared as the best candidate; otherwise, the routine continues to accept new inputs for the word. When the full length of the word has been entered, the target word with the highest current ranking is chosen.

Figure 3 shows the relation between the quantity of learning data (horizontal axis) and the number of keystrokes per word (vertical axis), with English at the top and Japanese at the bottom. The lines indicate various values of N_K (for 10, 5, 4, 3 and 2 code characters and the codes labeled "A" in Table 1). The larger the value of N_K, the lower the line. The horizontal solid line (around 5.5 keystrokes per word for English, and around 5.0 for Japanese) indicates the baseline, which is the average numbers of keystrokes needed to process a single word with a full keyboard in each language (without prediction by prefix).

When there were no learning data, entry with three or four buttons required more keystrokes than the baseline value. After learning 10,000 words, however, the number of keystrokes fell below the baseline for $N_K > 4$. For $N_K = 3$, on the other

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

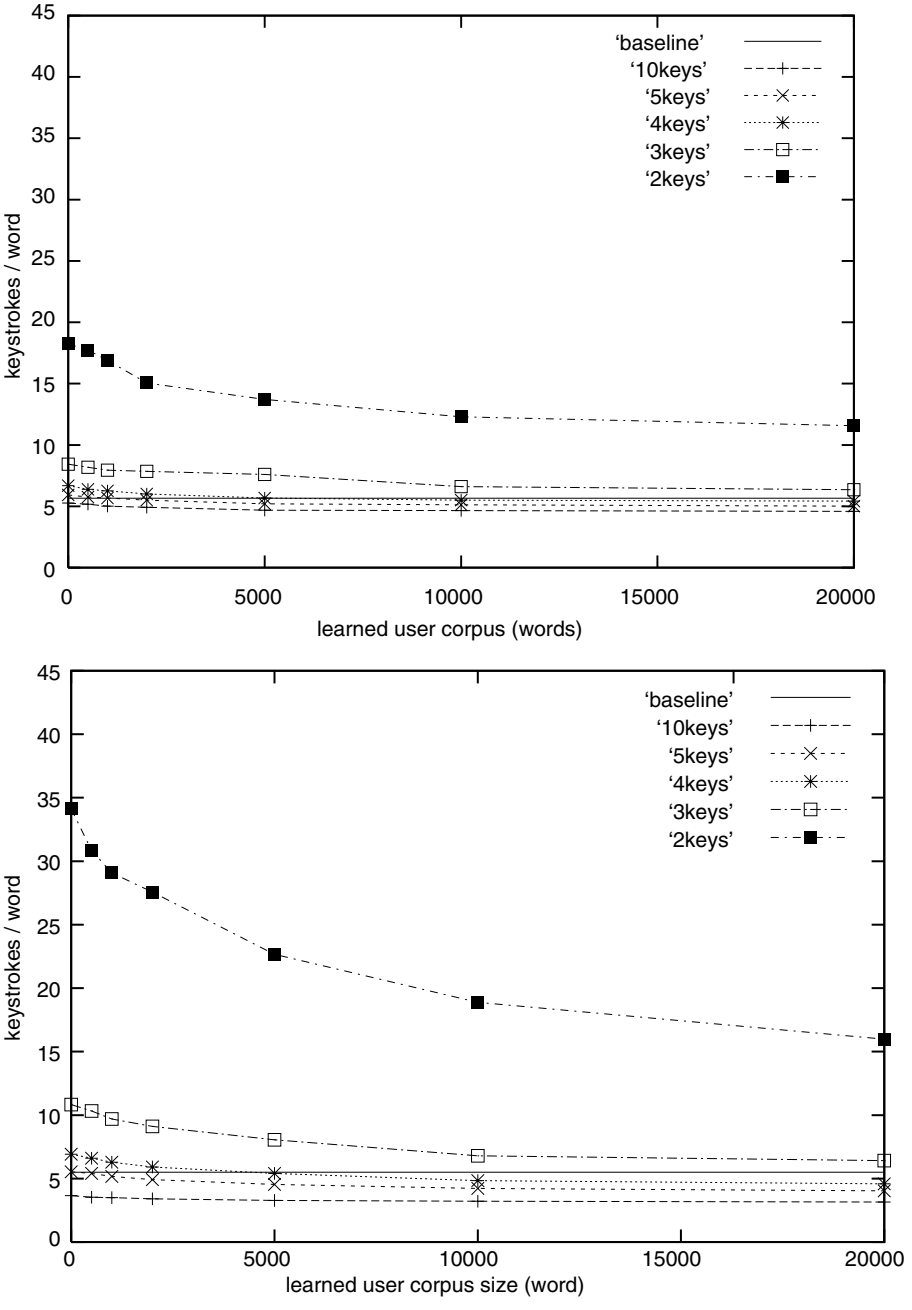


Fig. 3. Comparison of codes: English (top), Japanese (bottom).

hand, the number of keystrokes remained at around 6.0 per word, even with the increased size of the user corpus.

The trends described here were also observed when we tested different user corpora. Basically, the more specific the user corpus domain is, the more effective

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

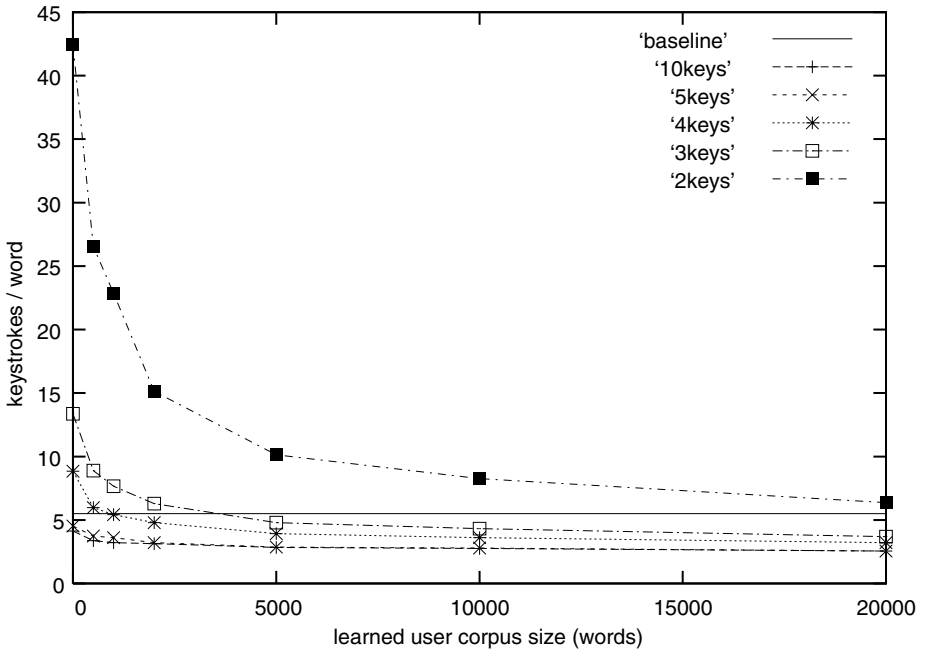


Fig. 4. Learning curve for a corpus of limited vocabulary.

the adaptation. A typical example is shown in Figure 4, which is the learning curve applied to another user corpus in Japanese. The content in this case was a functional programming language textbook (Bird and Wadler, 1990), which had a more restricted domain than that of e-mail. Comparing this figure with the bottom graph in Figure 3, it can be observed that the learning curve is steeper, and in this case, even TouchMeKey3 (i.e., TouchMeKey4 with only three keys) seems usable.

5.4 User evaluation

Finally, we conducted a user evaluation with the TouchMeKey system and investigated its usability. We performed two experiments, with different settings for the numbers of keys, subjects, and languages, different levels of expertise for the subjects, and different entry environments, as summarized in Table 3. Note that in Experiment 2, all the subjects were using TouchMeKey10 for the first time, but they were not complete novices, unlike in Experiment 1, because the key assignments were exactly the same as those used for Japanese cell phones.

The subjects participated in 10 testing sessions, each of which was 30 minutes long. For each session, the subjects were told to continue entering a given text as quickly as possible, without pausing. The vocabularies of the given texts were taken solely from the learned user corpora (though these texts were not used for training the models): this was done by selecting sentences consisting entirely of words from the vocabularies existing in the learned corpora. TouchMeKey4 learned a 10,000-word user corpus before it was handed to each subject.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

Table 3. *Settings for the user evaluation*

Experiment	1	2
Number of keys	4	10
Number of subjects and language	3 English 2 Japanese 2 Thai	5 Japanese
Level of expertise	Novice	All know key assignments but are testing TouchMeKey10 for first time
Device	Game controller	Digit keypad

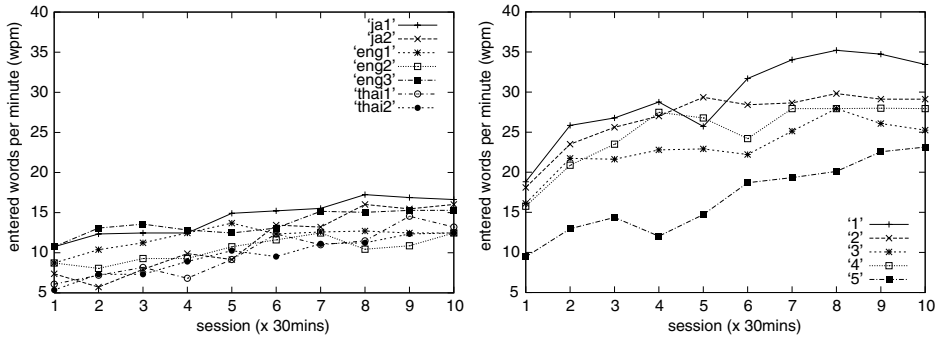


Fig. 5. Typing speed with TouchMeKey: 4 keys (left), 10 keys (right).

Figure 5 illustrates the results for speed. The left-hand graph shows the results for TouchMeKey4, while the right-hand graph corresponds to TouchMeKey10. The horizontal axis represents the sessions, and the vertical axis shows the average number of words per minute (wpm) in each session. Each line indicates the speed of a subject over time.

With regard to human learning, the more highly a subject was trained, the faster he or she became. The speeds of some subjects who had had difficulties at the beginning of the tests doubled by the end. After 5 hours of training, each subject could enter text at a faster rate: the TouchMeKey4 record was above 12 wpm, and the TouchMeKey10 record was above 19 wpm. The typical speed of entry by the multi-tap method on a mobile phone is in the range of 5–10 wpm (James and Reischel, 2001), so even TouchMeKey4 obviously enabled a comparable rate of text entry.

With TouchMeKey4, we continued further testing with one subject, and he reached speeds as fast as 23 wpm. This is comparable to the speeds obtained by the single-tap method on a cell phone (7–25 wpm (James and Reischel, 2001)).

Figure 6 shows the average error rate for the same subjects in each session using four keys. The horizontal axis shows the number of sessions, while the vertical axis indicates the error rate, which varied from 5% to 22%. We could not see any

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

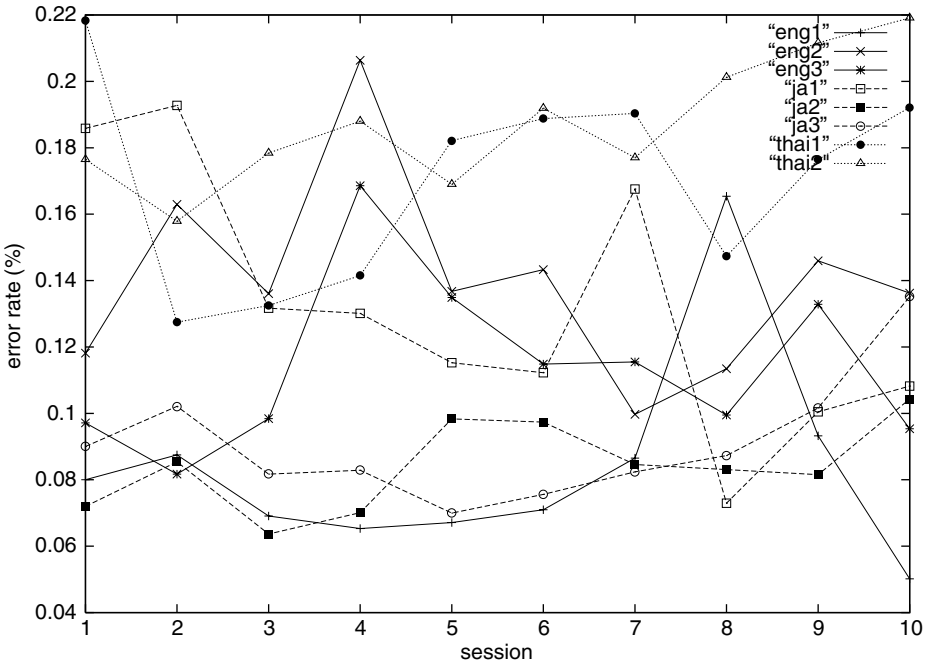


Fig. 6. Typing error rate with TouchMeKey: 4 keys.

decrease in the error rate after several sessions of training. The degree of error thus seemed to depend more on the individual.

This result shows that TouchMeKey4, in particular, placed a heavier cognitive load on the users. This is also shown by comparing the speeds shown in the two graphs of Figure 5 with the entry speeds for a full keyboard (40-60 wpm) and for a stylus with a virtual keyboard (32.5 wpm), reported by Zhai *et al.* (2000). The heavier cognitive load resulted from various factors:

1. The users were unfamiliar with the text entry method.
2. They were unfamiliar with entry using only a few buttons.
3. They had not memorized the key assignments.
4. They had to scroll down through many candidates.
5. They had to tackle the experiment with non-deterministic behavior caused by the adaptive function.

The subjects in Experiment 1 (4 keys) had all of these factors, whereas the subjects in Experiment 2 (10 keys) had only the last two. As a result, the first three factors induced a heavier cognitive load in Experiment 1 as compared to Experiment 2. This is one major reason for the speed difference. As for scrolling down through candidates, the average ranking difference between TouchMeKey10 and TouchMeKey4 was not so different. As for the non-determinism due to the adaptive function, we did not compare the cognitive loads with and without adaptiveness, but none of the users were opposed to using the adaptive function, least of all the subjects in Experiment 1.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

6 Vocabulary not registered in the dictionary

Thus far, we have assumed that all entered vocabulary exists in the dictionary. In a realistic case, however, a required target may not exist in the dictionary even if it was constructed from a large corpus. In this case, the user cannot enter the word. Thus, predictive text entry systems cannot easily handle texts written with special vocabulary words that have not been registered in the dictionary. For example, text written in a particular dialect, or containing old words or words with specific technical meanings, is difficult to handle. This problem is the major weak point of predictive text entry, as reported by Gutowitz (2003).

The problem is generally handled through the creation of a user dictionary. The user can enter unregistered words in some way (e.g., character by character) in order to register them in the user dictionary. After that, the system ranks these words highly when the user enters the corresponding sequence. It becomes his responsibility, however, to register such vocabulary, and this often becomes a cumbersome task. The situation gets worse when the entry system has only a few keys, because the process of entering the characters for a new vocabulary word will also become predictive. To alleviate this problem, some companies offer dictionaries of vocabularies for specific purposes, such as a chat dictionary or a dialect dictionary (JustSystem, 2005). Nevertheless, the style by which anyone user enters text into a device is likely to be unique, or more precisely, *user-specific*.

My group previously proposed a solution (Tanaka-Ishii *et al*., 2003a) based on dynamically acquiring missing vocabulary from the user corpus. That paper reports on an automatic method that dynamically obtains user-specific vocabulary from a user's unanalyzed documents. When the user makes an entry, the system dynamically extracts the corresponding chunks from the user text and suggests them, along with words suggested by the dictionary. With this method, texts in a particular style or concerning a specific domain can be entered using a predictive text entry system.

This approach is based on the interesting observation that a user typically *reuses* vocabulary at a 70% rate after entering only a small amount of text, which is again similar to the cache model. Thus, missing words are also reused, after appearing once in the text. Applying this tendency, we could develop a method of extracting repeatedly used strings from a small user corpus.

We hence evaluated the relationship between the corpus size and successfully acquired words, for user corpora in both Japanese and English based on chat applications, stories (obtained from the Gutenberg Project site for English and the Aozora Bunko site for Japanese), technical papers (obtained from the ACL anthology), patents (from the Japanese patent ministry), and manuals (for the UNIX system). We made the following observations:

- A user corpus with 5000 words was already effective in complementing unregistered words. (This corresponds to Figures 2 and 3).
- This trend held in both English and Japanese.
- The successfully acquired vocabulary ranged from 30% to 70%, depending on the number of unregistered words that originally existed in the user corpus.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

When the text contained more unregistered words, our method proved more effective.

As this method is string based, it also has proven effective for extracting certain frequently used phrases. With further study along this line, we believe that predictive text entry can be extended toward phrase-based prediction by prefix, which remains a problem for our future work.

The cognitive aspect of how to deal with unregistered words has not been verified in the work reported in this paper. It is difficult to measure, because there are several ways to enter unregistered words, depending on the language, and above all, the speed of text entry strongly depends on the test subject. James and Reischel (2001) examined this aspect with a few subjects in English. Such cognitive experiments also remain for future work on predictive text entry.

7 Conclusion

Predictive text entry is now becoming an important human language technology, widening the communication possibilities for small mobile devices and for the disabled. The goal of this paper was to discuss the models and technologies used in predictive text entry systems based on the ambiguity of language. Text entry is analogous to encoding and decoding for text compression, except that the codes used for the entry process must be human-readable. I first discussed code design and the evaluation of codes by means of the conditional entropy. Turning to the decoding process, I examined various adaptive language models, assuming that the entry action is intended for personal use and that the edited user documents contain limited vocabulary.

Then, I compared various models and codes, and finally, I examined the results of user evaluations conducted in English, Japanese, and Thai. The average speed of our test subjects was 14 words per minute (wpm), and the fastest subject reached 23 wpm, suggesting the possibility that predictive text entry systems can be a key technology for the disabled. The system and related corpus is available from Tanaka-Ishii (2005b).

As for future works, my group is now focusing on two main problems. The first is phrase-based prediction by prefix. This is the most challenging task in predictive text entry systems and language modeling. It is very important for application to communication aids, as the disabled prefer systems that minimize the number of actions, such as key-presses. The second problem is to implement one-button entry. Although four-button entry seems to be the minimum for Japanese, there are special cases, particularly in applications for the disabled, in which the user can only output one signal. We are now trying to figure out how predictive text entry methods can be utilized even in this extreme case. In addition to these two problems, the cognitive aspect of using predictive text entry, especially in terms of the adaptive function and error recovery, remains as an important consideration for future work.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

References

Aitchison, J. (1994) *Words in the Mind*. Blackwell.

Bell, T. C., Cleary, T. G. and Witten, T. H. (1990) *Text Compression*. Prentice Hall.

Bird, R. and Wadler, P. (1990) *Introduction to Functional Programming*. Kindai Kagaku sha. (Translated by M. Takeichi.)

Bloom. C. (1998) Solving the problems of context modeling. Available from http://www.cbloom.com/papers/index.html, visited August 2005.

Chasen (1997) Manual of Japanese mophological analyzer chasen. JAIST Technical Report, http://chasen.aist-nara.ac.jp/index.html.en.

ChDrv (2005) Linux Chinese HOWTO English version: (Chapter Chdrv InputMethod). http://www.tldp.org/HOWTO/Chinese-HOWTO-4.html .

Chinese Input Methods (2005) Encyclopedia: Chinese input methods for computer. http://www.nationmaster.com/encyclopedia/Chinese-input-methods-for-computer .

Cleary, J. G. and Witten, I. H. (1984) Data compression using adaptive coding and partial string matching. *IEEE Trans. Communications*, **32**(4): 392–402.

Darragh, J. and Witten, I. H. (1992) *The Reactive Keyboard*. Cambridge University Press.

Goodman, J., Venolia, G., Steury, K. and Parker, C. (2000) Language modeling for soft keyboards. *7th International Conference on Intelligent User Interface*.

Gutowitz, H. (2003) Barriers to adoption of dictionary-based text-entry methods:a field study. *EACL 2003 Workshop on Language Modeling for Text Entry Systems*, pp. 33–41.

IBM (2001) Watchpad 1. 5. http://www.trl.ibm.com/projects/ngm/index_e.htm , visited Aug., 2005.

Inutsuka, Y., Tanaka-Ishii, K. and Takeichi, M. (2001) Thai input system with digits. *NLPRS*, pp. 579–584.

Isokoski, P. (2001) Model for unistroke writing time. *Proceedings of the Annual SIGCHI Conference*.

Iwasaki, H. and Tanaka-Ishii, K. (2004) An interactive proofreading system for inappropriately selected words on using predictive text entry. *The 1st International Joint Conference on Natural Language Processing*, pp. 755–764.

James, C. and Reischel, K. (2001) Text input for mobile devices: Comparing model prediction to actual performance. *Proceedings of the Annual SIGCHI Conference*.

Jelinek, F. (1997) *Statistical Methods for Speech Recognition*. MIT Press.

Jones, P. (1998) Virtual keyboard with scanning and augmented by prediction. *Proceedings of European Conference series on Disability, Virtual Reality and Associated Technologies*. The university of Reading. available from http://www.icdvrat.reading.ac.uk/1998/papers/1998_06.pdf, visited Aug., 2005.

JustSystem (2005) Atok: Japanese kana-kanji conversion system. http://www.atok.com/, in Japanese.

Kuhn, R. and De Mori, R. 1990. A cache-based natural language model for speech reproduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(6): 570–583.

Kurihara, A. and Kurosaki, H. (1967) About a transformation method of kana into kanji text. *Technical Report of the University of Kyusyu*, **39**: 659–664. In Japanese.

MacKenzie, I. S., Smith, K. H., Jones, D. and Skepner, T. (2001) Prefix-based disambiguation for mobile text input. *UIST*, pp. 111–120.

Manning, H. and Schutze, C. D. (1999) *Foundations of Statistical Natural Language Processing*. MIT Press.

Masui, T. (1999) PO_BOX an efficient text input method for handheld and ubiquitous computers. *The ACM Symposium on User Interface Software and Technology*, pp. 113–119.

Masui, T. (2004) Survey of text entry systems. Available from `http://pitecan.com/InputSystems/`, visited in 2005.

MisawaHome (2002) Cutkey. `http://www.misawa.co.jp/CUTKEY/index.html`.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press

MS-IME (2005) Microsoft input method editor (IME), Japanese version. Available from Microsoft Home Page.

Origin Instruments (2005) Headmouse and computer access software. http://orin.com/index.htm.

Pete (2005) Pete: Japanese predictive text entry system for the disabled. http://at.a-brain.com/PeteHP/, visited Aug., 2005.

Rau, H. and Skiena, D. (1994) Dialing for documents: An experiment in information theory. *UIST*.

Shkarin, D. (2002) PPM: One step to practicality. *Data Compresion Conference*. IEEE Computer Society.

Sugimoto, M. and Takahashi, T. (1996) SHK: Single hand key card for mobile devices. *Conference Companion, CHI'96*, pp. 7–8.

Tanaka-Ishii, K., Inutsuka, Y. and Takeichi, M. (2000) Japanese input system with digits – Can Japanese be input only with consonants?. *Human Language Technology Conference 2001*.

Tanaka-Ishii, K., Inutsuka, Y. and Takeichi, M. (2001) Personalization of text input systems for mobile phones. *NLPRS*, pp. 177–184.

Tanaka-Ishii, K., Inutsuka, Y. and Takeichi, M. (2002) Entering text using a four button device. *The 19th International Conference on Computational Linguistics*, pp. 988–994.

Tanaka-Ishii, K., Inutsuka, Y. and Takeichi, M. (2003a) Acquiring vocabulary for predictive text entry through dynamic reuse of a small user corpus. *Annual Meeting of the Association for Computational Linguistics*, pp. 407–414.

Tanaka-Ishii, K., Inutsuka, Y. and Takeichi, M. (2003b) Entering Japanese with a few button device. *Information Processing Science Journal*, **44**(2): 433–442. In Japanese.

Tanaka-Ishii, K. (2005a) A survey on text entry by a few buttons – towards universal communication. *Journal of Informtion Processing Society of Japan*, **46**(6): 691–696. In Japanese.

Tanaka-Ishii, K. (2005b) TouchMeKey site. Available from `http://kumish.net/tmk.html`.

Teahan, W. J., Wen, Y., MacNab, R. and Witten, I. H. (2000) A compression-based algorithm for chinese word segmentation. *Computational Linguistics*, **26**: 375–393.

Tegic 9 (2000) Tegic 9 home page. http://www. t9. com.

Ward, D., Blackwell, A. F. and MacKay, D. J. C. (2000) Dasher: a data entry interface using continuous gestures and language models. *The ACM Symposium on User Interfface Software and Technology*, pp. 129–137.

Zhai, S., Hunter, M. and Smith, B. A. (2000) The metropolis keyboard – an exploration of quantitative techniques for virtual keyboard design. *UIST*, pp. 119–128.

ZI-Corporation (2003) ZI home page. Available from http://zicorp. com/.

https://doi.org/10.1017/S1351324905004080 Published online by Cambridge University Press