Efficient Context-sensitive Word Completion for Mobile Devices

Antal van den Bosch
Tilburg Centre for Creative Computing
Tilburg University
P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands
Antal.vdnBosch@uvt.nl

Toine Bogers
Tilburg Centre for Creative Computing
Tilburg University
P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands
A.M.Bogers@uvt.nl

ABSTRACT

Word completion is a basic technology for reducing the effort involved in text entry on mobile devices and in augmentative communication devices, where efficiency and ease of use are needed, but where a low memory footprint is also required. Standard solutions compress a lexicon into a suffix tree with a small memory footprint and high retrieval speed. Keystroke savings, a measurable correlate of text entry effort gain, typically improve when the algorithm would also take into account the previous word; however, this comes at the cost of a large footprint. We develop two word completion algorithms that encode the previous word in the input. The first algorithm utilizes a character buffer that includes a fixed number of recent keystrokes, including those belonging to previous words. The second algorithm includes the complete previous word as an extra input feature. In simulation studies, the first algorithm yields marked improvements in keystroke savings, but has a large memory footprint. The second algorithm can be tuned by frequency thresholding to have a small footprint, and be less than one order of magnitude slower than the baseline system, while its keystroke savings improve over the baseline.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Algorithms, Human Factors, Performance

Keywords

Word completion, predictive text processing, mobile devices, context sensitivity, scaling, ergonomics

1. INTRODUCTION

Word completion is a basic technology for text entry on mobile devices, as well as an important component in augmentative language technology tools [3]. Its main aim is to reduce the number of keystrokes necessary during text entry by correctly suggesting the completion of the word currently being entered, as soon as possible, so that the word does not have to be completely keyed in. One straightforward strategy is to generate a suggestion at the word's unicity point, i.e. the point at which the word is the only word available in the algorithm's internal word model (e.g. a list of words) that fits the string of characters keyed in so far. The word completion algorithm may also venture to suggest a certain word even before any of the possible words' unicity points are reached. From the moment that the model suggests a correct word completion (visually displayed in a designated part of the device's screen), the user is typically able to click an "Accept" button to accord and enact the suggestion. Actual savings are made when the suggestion is accorded before the word's before-last character is keyed in. Although word completion algorithms are used in many devices with some success, current word completion systems remain imperfect, and are vulnerable to at least the following two factors.

The first factor that hampers essentially all word completion systems is that when a word currently being entered does not occur in the system's internal word or language model, the system will not suggest it, and the user will need to key in the full word. This problem can only be overcome by including more words in the construction of the algorithm, in the hope of having a better coverage of new, unseen text.

A second issue that applies to the simpler kind of word completion systems that are only based on a word list, is that after every space or punctuation mark, these systems start to guess words anew without taking into account the previous sequence of characters or words. Yet, a classic and fundamental insight, quantified in information theory [11], is that characters or words preceding the current word may contribute information that would enable suggesting the current word earlier than its unicity point, in some cases even immediately. Using information from the previously entered text may also help in picking one particular suggested word over alternative words with the same initial characters, but which are less likely given the previous word.

In this paper, we explore these two problems, provide solutions, and analyze their relative utility. It is structured as follows. After reviewing related work in the area of word completion (or predictive text processing) algorithms in Section 2, we introduce a context-insensitive baseline word completion algorithm and two context-sensitive variants in Section 3. After introducing our training and testing data and our simulation study in Section 4, we show the relation between the success of our three algorithms (measurable in the percentage of key presses saved) and the size and coverage of their word models, and we compare the three models, in Section 5. We conclude in Section 6 that by including context, keystroke savings can indeed be improved at the cost of mem-

MobileHCI 2008, September 2–5, 2008, Amsterdam, the Netherlands.
ACM 978-1-59593-952-4/08/09.

ory needed; and that a context-sensitive word completion algorithm that takes into account the full previous word (if frequency-thresholded) as a feature, yields substantial improvements in keystroke savings, against only a modest increase in memory usage. As a final note, we discuss valorization aspects of word prediction systems in general, and our study in particular, in Section 7.

2. RELATED WORK

Word completion systems aim to reduce the effort spent on entering text in a digital device. Generally speaking, there are two different approaches to reducing the effort of text entry [12]. The first approach is to find a way of reducing the physical restraints of entering text by either using alternate modalities or by changing or augmenting the keyboard layout—such as re-ordering the key mappings. Examples of such re-orderings have been proposed by [6] and [13], among others.

The other approach—and the one we focus on in this paper—aims at reducing the amount of typing necessary to enter a text by using predictive typing aids such as word completion systems. Such aids try to predict the completion of the current word as it is being keyed in, and are available for input modalities ranging from full keyboards to the more restricted keypads of mobile phones. For instance, the popular T9 algorithm was designed specifically to offer predictive text entry on the standard 12-key keypad layout on the current generation of mobile phones [5]. However, a growing number of mobile phones, PDAs, and BlackBerrys sport full QWERTY keyboards. We therefore focus on predictive text entry using the latter type of keyboards in this paper.

Predictive text entry algorithms use context information to anticipate what block of characters (letters, n-grams, syllables, words, or entire phrases) a person is going to write next [3]. The block size the typing aid tries to predict, influences the potential savings in terms of time and keystrokes. Prediction of n-gram sized blocks has been applied by, for instance, Goodman et al. (2002), who used n-gram models to correct and predict user input using soft keyboards, i.e. on-screen keyboards operated using a touch screen or a stylus [4]. Using language models of at most the 6 previous characters, they were able to successfully correct user input, when it was not entirely clear what key the user meant to press. MacKenzie et al. (2001) also use letter sequence probabilities to disambiguate between ambiguous user input [9]. They emphasize the necessity of keeping the memory footprint small while at the same time maximizing predictive performance.

However, predicting n-grams or syllables can result in increased cognitive effort for the user who has to check and approve the predictions. Because words (whitespace-delimited sequences of letters) are more easily recognizable, thereby reducing the cognitive effort required, words are most widely used as predicted blocks [3].

One of the earliest applications of predictive text entry was the Reactive Keyboard by Darragh et al. (1990). They adopted a dynamic, implicit, and adaptive modeling strategy by using a tree-based memory structure to store recurring n-grams and quickly match substrings associated with predictions. Suggestions were sorted by length and frequency, with the longest, most frequent substrings being preferred [2].

How et al. (2005) took the previous word into account in the word completion task by using using a bigram word model to predict the most likely word in the sequence based on matching character prefixes of the current word being typed. They report time savings of 14.1% on average [6]. Tanaka-Ishii (2007) compared four language models for predictive text entry [13]: two simple models based on frequency and recency counts, a model based on co-occurrence between words, and an adaptive n-gram model that takes into account the probability distribution of words previously used by the user as well. The adaptive n-gram model performed best. Stocky et al. (2004) took a semantic approach to word prediction instead of a purely statistical one, by looking up common sense context from the Open Mind Common Sense (OMCS) project for each completed word [12]. All words from the retrieved contexts have their frequency scores updated in the text prediction dictionary; based on the first typed characters of the next word, words are then suggested by frequency. As their training material, they used a corpus of 5,500 e-mail messages sent over the course of one year by one user, consisting of 1.1M words. In addition, they used three smaller Web page corpora containing between 10,500 and 16,500 words.

Some general conclusions can be drawn from the body of related work. One is that experts tend to benefit more from predictive text entry for reduced-size keyboards than novices do, but for standard-sized keyboards this situation is reversed [2]. This is because predictive typing aids are more likely to be useful in situations with non-standard keyboards, where people cannot enter characters as fast as normal. Predictive text entry also tends to yield less benefit when predicting SMS text than normal text, because the vocabulary is more personalized and because average word length tends to be smaller, hence the potential savings are also reduced. Finally, a general problem is that a lack of standardized test corpora makes comparing the different approaches difficult. The majority of predictive text approaches tend to be evaluated on different corpora, usually composed of newspaper text. An additional problem is that every research effort seems to be evaluated using different metrics as well: where one approach is evaluated using hit ratio (i.e. the level of accuracy in predicting the correct word), others are evaluated in terms of keystroke savings or time savings.

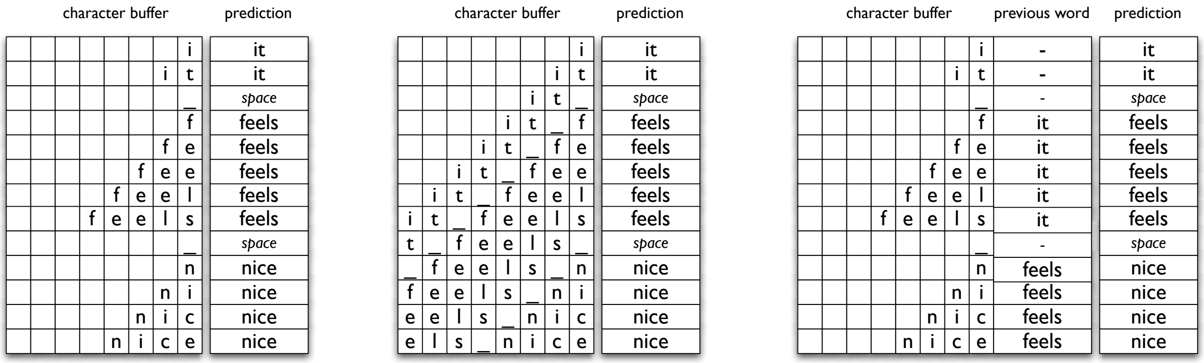
3. THREE WORD COMPLETION ALGORITHMS

The two disjoint goals of a fast word completion algorithm are (i) that it needs to hold a wide-coverage word or language model, and (ii) it should be able to say at the earliest possible point of a character sequence being entered, with a success rate that should be as high as possible, to which word the sequence can be completed.

This task can be phrased as a classification problem in which an input sequence of entered keys is mapped to the word that is actually being intended by the person keying the text. Suppose that a person is keying in the text “it feels nice” on a normal QWERTY keyboard. First, the word prediction algorithm needs to suggest “it” during the first two keystrokes. Then, after being reset by the space bar, it needs to suggest “feels” throughout the sequence of that word being keyed in, etcetera. Its classification task thus can be made explicit as thirteen classification instances depicted at the left-hand side of Figure 1.

To train a classifier, either a lexicon or a text could be encoded with the scheme as illustrated in Figure 1, resulting in as many labeled instances as characters in the lexicon or text. The classifier, then, has to fulfill the need of preserving all the words in the lexicon or training text, and being able to generate, given new input of key sequences, predictions of the correct words as soon as possible as the word is being keyed in.

To this purpose, the general structure of *tries* are well suited [8]. In our study, we employ the IGTree algorithm [1], which implements trie compression and classification. In the first compression phase, a list of words (either from a lexicon or a text) is compressed by IGTree into a trie structure. To do this, first a one-time ordering of features is computed, where the features are the positions



character buffer								prediction
							i	it
						i	t	it
							_	*space*
							f	feels
						f	e	feels
					f	e	e	feels
				f	e	e	l	feels
			f	e	e	l	s	feels
							_	*space*
							n	nice
						n	i	nice
					n	i	c	nice
				n	i	c	e	nice

character buffer								prediction
							i	it
						i	t	it
					i	t	_	*space*
				i	t	_	f	feels
			i	t	_	f	e	feels
		i	t	_	f	e	e	feels
	i	t	_	f	e	e	l	feels
i	t	_	f	e	e	l	s	feels
t	_	f	e	e	l	s	_	*space*
_	f	e	e	l	s	_	n	nice
f	e	e	l	s	_	n	i	nice
e	e	l	s	_	n	i	c	nice
e	l	s	_	n	i	c	e	nice

character buffer								previous word	prediction
							i	-	it
						i	t	-	it
							_	-	*space*
							f	it	feels
						f	e	it	feels
					f	e	e	it	feels
				f	e	e	l	it	feels
			f	e	e	l	s	it	feels
							_	-	*space*
							n	feels	nice
						n	i	feels	nice
					n	i	c	feels	nice
				n	i	c	e	feels	nice

Figure 1: Example classification instances derived from the sentence "it feels nice" for the context-insensitive algorithm (left), the character context algorithm (middle), and the previous-word context algorithm (right).

in the character buffer. The ordering of the features is determined by their information gain ratio with respect to predicting the word [10]. Given any reasonable amount of examples, the rightmost feature, i.e. the most recently pressed key, has the overall highest predictive power, hence the largest gain ratio. The overall ordering is from right to left. Subsequently, a root node is created, which represents the most likely word when no key is pressed yet. This root node fans out to a first layer of nodes through arcs, where the arcs represent all possible keystrokes. Each first-layer node represents the most likely word given a single keystroke, and branches out to second-layer nodes, connected by arcs that denote all possible keystrokes given the first keystroke. A node becomes an end node if the arc leading to that node uniquely identifies a single word, even if the arc is not the last character of the word. This algorithm is recursively applied until a compressed trie is produced, ready to process new instances.

The trie classifies new incoming instances (where the keystrokes are known, but the word needs to be predicted) by traversing the tree with the current character buffer (cf. Figure 1) as input features. Starting with the root node, it deterministically takes the arc representing the rightmost character, and continues following matching arcs in the ordering of the features until either (i) it encounters an end node, at which point the predicted word is emitted as output, or (ii) it encounters a non-ending node from which it finds no more matching arcs connecting to nodes further down the tree, at which point it emits the most likely word so far.

In the larger context of the text application, this trie is embedded in a real-time wrapper that reads each incoming keystroke, updates the character buffer (shifting it leftward with each keystroke, erasing it after each space), sends the buffer to the trie, catches the prediction emitted by the trie, and presents this to the user, who can then press a special "Accept" key to accept the trie's suggestion.

To include context of previously entered text, two basic options are available and tested in this paper. The first, illustrated in the middle of Figure 1, is to not reset the character buffer after the spacebar or a punctuation key is hit, but rather to keep the character buffer filled with a fixed number of recently pressed keys – thus including the previous word, or at least a part of it, or sometimes even a part of the before-last word.

The second variant is to include the full previous word (or words) keyed in before the last space. The right-hand part of Figure 1 displays the setup explored in this paper, where a simple character buffer that is reset after each space, is accompanied by a single feature carrying the word previous to the word currently being predicted. In the trie, this "previous word" feature is mixed with the other letter features, making the trie somewhat more complicated in shape. An unrestricted word feature can have hundreds of thousands of unique values (i.e. all unique words occurring in the million-word corpora used for training), hence a trie that would split such a node would be shattered over a Zipfian distribution of nodes, including a long tail of hapax values (words occurring once, typically about half of all the unique words in the training corpus) offering no generalization power, and likely producing incorrect next-word suggestions [14]. As suggested by Van den Bosch (2005), a word-valued feature can be made more efficient when integrated in a trie structure when only the most frequent values of the feature, e.g. the top n most frequent words, are kept as discrete values, and all other values are lumped together under a dummy value. In empirical tests we set $n = 200$, producing the best performance on a held-out test set. Our word-valued feature thus causes a 201-fold branching in the trie at the level it is invoked.

4. EXPERIMENTAL SETUP

We run simulation studies on a large amount of controlled data, split into training and testing material. These simulation studies emulate human typing behavior in a deterministic way, i.e., they emulate a fixed strategy in typing in which correct word completion suggestions are accepted at the earliest possible point. Running such an artificial simulation study abstracts away from human error and deviating strategies (such as ignoring correct suggestions occasionally), allowing experiments with very large amounts of data, but disallowing the measurement of real processing times, measurements of energy spent, or brain activity measurements. Essentially, we assume that keystroke savings are a sensible correlate of human effort savings.

Our experiments are based on Dutch data, with the purpose of posing a slightly larger challenge than English. Like English and German, Dutch is a germanic language. Like German, Dutch has a productive compounding morphology, allowing for long admissible words. This phenomenon puts a pressure on the unknown words problem, as long compounds tend to be rare, and compared to English, take away frequency counts from the words they are composed of.

As a text corpus for training and testing, we used the first four months of a Dutch local newspaper's full article archive, split into a training corpus of up to ten million words, and a disjoint test set of 100,000 words. The corpus thus consists of journalistic text, covering local, national, and international topics of all sorts. In addition, we include one alternative test set representing more colloquial and social language, viz. a collection of transcribed face-to-face dialogues from the Spoken Dutch Corpus[1]. This subcorpus is categorized as the most spontaneous register in the corpus, and contains a total of 2,444,755 words after basic cleanup (we removed transcribed disfluencies, and converted filled pauses to commas, to better resemble orthographic text).

Note that with both sets we are using running text as training data, and not a lexicon. Obviously we need the text to be able to generate the left-hand contextual features in the two context-sensitive word completion system variants. At the same time, this approach does mean for the context-insensitive variant that it is trained on all unique words occurring in the corpus, and nothing else, e.g. no external balanced or encyclopedic list of Dutch words, which could be a welcome boost in principle. In order to allow for a proper comparison between the three systems, we train and test all three on the exact same material.

4.1 Experimental design and evaluation

Keeping the test sets constant, we vary the amount of training material in a pseudo-exponential series, starting at 10,000 words, ending at 10,000,000. At each step in the curve, we perform a full IGTree experiment in which we generate a trie, and process the test sets with the trie. At each simulation experiment, we measure the following for evaluation purposes:

1. The number of nodes in the trie;
2. The number of characters processed per second by the IGTree classification algorithm;
3. The percentage of keystrokes saved.

The latter percentage of keystrokes saved is the proportion of the total number of keystrokes needed to generate the entire test text, following all correct suggestions of the system as soon as possible, against the total number of keystrokes needed when the full test text would be typed in character by character. In principle, keystrokes can be saved if the word is correctly suggested before the before-last character is pressed. The amount of keystrokes saved will therefore be 0.0% or higher; at 0.0%, the word completion system is completely ineffective.

5. RESULTS

Figure 2 displays the learning curve results of our three variants, measured on the 100,000-words test set drawn from the same source as the training material. The x-axis, representing the number of words in the training corpus, follows a logarithmic scale, while the y-axis represents the percentage of keystrokes saved against the default situation of typing in all words character by character. First, we observe that all three lines exhibit an upward trend; trained on more data, they lead to better keystroke savings. A second observation is that the context-sensitive variants, both the one based on a fixed-length buffer with previous character context (the dashed line) and the one based on the previous word context (the dotted line), appear to be exhibiting a log-linear growth; with every doubling of the training data, they yield a constant improvement in keystroke savings. In contrast, the curve representing the context-insensitive baseline system appears to taper off, falling increasingly behind the the context-sensitive variants.

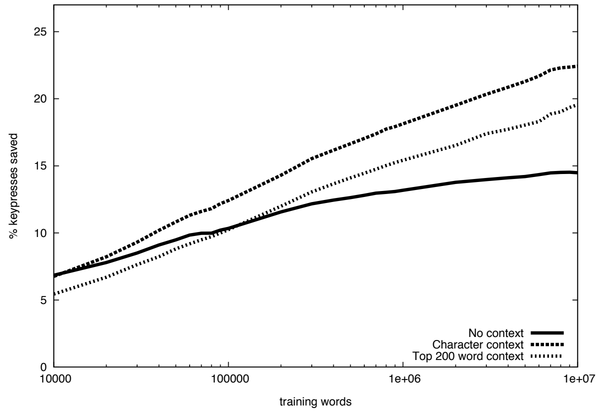


Figure 2: Learning curves of the three word completion variants, in terms of the percentage of characters saved, measured on processing the 100,000-word test text.

At the maximal training set size of 10 million words, the baseline system saves 14.5% keystrokes. In contrast, the system using previous character context saves 22.4% (a relative increase of 54%), and the system using the previous word as context saves 19.6% (a relative increase of 35%).

However, context sensitivity comes at the cost of a larger memory footprint. Using the same logarithmic x-axis as Figure 2, Figure 3 displays the number of tree nodes needed at the various training set sizes by the three systems. The y-axis is also logarithmic. In this log-log space, the numbers of tree nodes needed by both two character-buffer-based systems, the context-insensitive baseline, and the system using the previous character context have a linear relation with the number of training instances. At the largest training set size the context-insensitive baseline system uses about 15 times less memory than the system using the character context: 1.7 million nodes vs. 26.2 million nodes. Slightly in contrast, the amount of nodes needed by the system using the previous word as context tapers off with more training data, ending up at about double the amount of nodes, 3.4 million, compared to the baseline system at the same 10 million training words. The memory footprint curve of this system even displays a mild decrease, caused by the fact that the word-valued feature in the trie ends up in a lower place in the gain ratio feature ranking when more training data is available—i.e. with more training data the feature is tested deeper in the tree, causing relatively less fragmentation higher up in the tree.

Besides memory footprint, we would like to ensure that our variants are not exceptionally slower than the baseline, as the algorithm has to be fast enough to follow real time keystrokes, the average speed of which has been estimated at 0.12 seconds per keystroke for a good typist, and 0.28 seconds for a bad typist [7]. Figure 4 displays the speed of the three systems using the same logarithmic learning curve axis, measured over the full 100,000 word test set in uninterrupted classification mode, on a state-of-the-art multicore computing server. The figure shows that the speed of the three systems, despite their widely differing memory footprints, is quite similar, and sufficiently high. The speed of all three systems declines from about 65 thousand characters (keystrokes) per second

[1] http://lands.let.kun.nl/cgn/ehome.htm

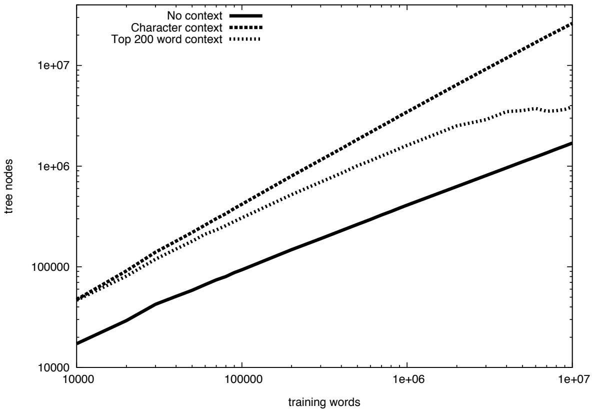


Figure 3: Memory usage of the three word completion variants, in terms of the number of trie nodes.

to around 40 thousand characters per second for the two context-sensitive systems, and around 50 thousand characters per second for the context-insensitive baseline system at 10 million words of training data. Even if the processing unit of the mobile device is two orders of magnitude slower than the AMD Opteron chipset of our computing server, it would be fast enough to generate new suggestions immediately after each new keystroke.

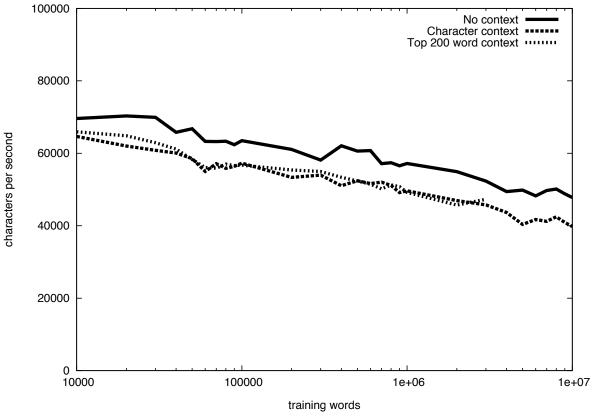


Figure 4: Speed of the three word completion variants, in terms of the number of characters processed per second.

The results displayed in Figures 2, 3, and 4 are obtained with the 100,000-word test set disjoint from, but from the same origin and register as the training set of newspaper texts. As argued earlier, an external test set of a different register would pose a potentially difficult challenge to the word completion algorithm, just as genre, domain and register differences do in language modeling [14]. Table 1 lists the keystroke savings attained on the external test set of transcribed spontaneous Dutch face-to-face dialogues, in contrast with the savings on the in-register test set, by the three systems. It is quite apparent from the two columns in Table 1 that the keystroke savings on the test set from the different register are dramatically lower, under 10%. At the same time, the relative gains by the context-sensitive methods follow the same pattern as observed before, and even slightly better in terms of relative improvement.

Table 1: Difference in keystroke savings on the in-domain 100,000-word test set, vs. the out-of-domain spontaneous dialogue test set at the maximal training set size by the three systems.

	% Keystrokes saved	
System	In-register test	Out-of-register test
Baseline	14.5	5.3
Character context	22.4	9.8
Word context	19.5	8.4

6. DISCUSSION

In this paper we explored two variants of a baseline word completion system, that make use of previous-word context in two different ways. We ran simulation experiments, measuring keystroke savings, under the assumption that these savings are a correlate of human effort saved when the word completion algorithm would be incorporated in text entry software. One variant includes the previous character context up to a fixed number of characters, while the other variant included a frequency-thresholded representation of the previous word that was keyed in before the current one. Arguably, the variant encoding the full previous word captures the linguistic intuition that the identity of the previous word holds the relevant information, in contrast to the previous characters. On the other hand, the previous characters do encode the identity of the previous word or words implicitly. Also, they hold partial letter-by-letter information on the end of the previous word, capturing morphological information on suffixes and inflections, which may carry predictive information as well.

With more data, all systems perform better, but while the baseline system's performance gain tapers off, the gains of the context-sensitive systems improve over the baseline, and appear to increase at a log-linear pace. At 10 million words of training data, the best keystroke savings percentage of 22.4% is attained by the system using the previous character context, a relative improvement of 54% over the baseline savings percentage of 14.5%.

Unfortunately, this optimal savings percentage comes with the largest memory footprint. Given that a straightforward (uneconomical) implementation of the trie costs 20 bytes per node, the best-performing system would require 524 Mb of working memory, which is an infeasible amount of memory on most current mobile devices. The variant that uses the frequency-thresholded previous word as contextual information uses substantially less memory and only about double the amount of memory that the baseline system uses. The frequency thresholding, tuned by selecting an optimal top-n words on a held-out test set, resulted in n being set to 200, which is similar to the the typical number of function words in germanic languages (the top 200 most frequent Dutch words include most of the closed-class function word categories such as determiners, conjunctives, pronouns, and to a lesser degree prepositions, and contain relatively few open-class words).

As an additional analysis, we compared the efficiency of our three systems on a test set that contains widely different type of text than the training set. While the training set contains newspaper text, the external out-of-register test set contained a large set of transcribed spontaneous face-to-face conversations about mundane topics and containing a lot of social discourse and chit-chat. Importantly, we reported a dramatic drop in the keystroke savings on this type of text, to under 10%. At the same time, we noted considerably better savings by the context-sensitive methods as compared to the baseline system.

In future work, one obvious extension to the current matrix of experiments is to mix the roles of training and test sets systematically (so that the transcribed dialogue corpus is also training set in one set of experiments), and also to generate a combined training set of the two registers. Other more extreme test sets may be considered, in which abbreviations are used typical of mobile phone and chat text entry, such as “w8” for “wait”, or where the use of language is sloppy and full of spelling and grammar errors. Both short words and errors pose a challenge to word completion algorithms, as few savings can be made with short words, and errors constitute unknown tokens.

Other avenues of future research include robustness to multilingual data, which can be seen as an extreme form of different registers; in reality, it is quite likely that many users will actually generate different texts in different registers and languages on the same device. Also, personalization and incremental training and adaptation to individual users should be integrated in the current approach, as well as adaptability to keyboards with less keys than the language has characters (such as mobile phone keypads, or QWERTY-keyboards for certain Asian languages).

7. VALORIZATION

Word completion subsystems are frequently included in communication aids in order to increase the user’s rate of communication. Aids for predictive text entry have several important applications, with improved access for people with disabilities being among the most important ones. People with severe motor and oral disabilities such as cerebral palsy or hemiplegia can greatly benefit from tools that enable them to increase their communication speed [3]. For many years, predictive text entry techniques have also found application in languages with large-volume character sets such as Chinese or Japanese, where the standard keyboard does not have enough keys to accommodate all the characters present in the language. Here, typing aids can enable the projection of a wide range of characters with a limited number of keys; in fact, this is currently the major method of solving this problem [13].

Other popular applications of predictive text entry are input situations with a reduced number of keys, such as mobile phones, and situations without direct tactile feedback, such as stylus-operated touch screens. Most current mobile phones are equipped with some version of a word completion algorithm; the T9 algorithm [5], for instance, is claimed to be shipped with 2.5 billion devices[2], on the basis of which the owners could claim to have the most widely distributed piece of language technology in the world.

Our particular contribution to the current state of the art in word completion technology is that we formulate recommendations on how the efficiency of this technology can be improved, at fairly limited costs (a mildly higher memory footprint). Our learning curve experiments offer a tool to chart the trade-off between having more training material and more context sensitivity (causing higher text entry efficiency) and needing more memory. As memory limitations of mobile devices are gradually and steadily being reduced, our recommendations could be used by mobile device developers to support the decision to move from a relatively limited word completion system to a less constrained context-sensitive system, offering customers a noticeably better service.

Acknowledgments

The research reported in this paper has been funded by the Dutch Ministry of Economic Affairs, under its Innovation Programme on Man-Machine Interaction, IOP MMI. The authors wish to thank Menno van Zaanen, Herman Stehouwer, Peter Berck, other members of the ILK (Tilburg) and ILPS (Amsterdam) research groups, and Frank Hofstede for comments and suggestions.

8. REFERENCES

[1] W. Daelemans, A. Van den Bosch, and A. Weijters. IGTree: Using Trees for Compression and Classification in Lazy Learning Algorithms. *Artificial Intelligence Review*, 11:407–423, 1997.

[2] J. J. Darragh, I. H. Witten, and M. L. James. The Reactive Keyboard: A Predictive Typing Aid. *Computer*, 23(11):41–49, 1990.

[3] N. Garay-Vitoria and J. Abascal. Text Prediction Systems: A Survey. *Universal Access in the Information Society*, 4(3):188–203, 2006.

[4] J. Goodman, G. Venolia, K. Steury, and C. Parker. Language Modeling for Soft Keyboards. In *Proceedings of IUI ’02*, pages 194–195, New York, NY, USA, 2002. ACM.

[5] D. L. Grover, M. T. King, and C. A. Kushler. Reduced Keyboard Disambiguating Computer. Patent No. US5818437, Tegic Communications, Inc., Seattle, WA, October 1998.

[6] Y. How and M.-Y. Kan. Optimizing Predictive Text Entry for Short Message Service on Mobile Phones. In M. J. Smith and G. Salvendy, editors, *Proceedings of HCII ’05*, Las Vegas, NV, July 2005. Lawrence Erlbaum Associates.

[7] D. Kieras and B. John. The GOMS Family of Analysis Techniques: Tools for Design and Evaluation. Technical Report CMU-HCII-94-106, Carnegie Mellon University, 1994.

[8] D. E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Reading, MA, 1973.

[9] I. S. MacKenzie, H. Kober, D. Smith, T. Jones, and E. Skepner. LetterWise: Prefix-Based Disambiguation For Mobile Text Input. In *Proceedings of UIST ’01*, pages 111–120, New York, NY, USA, 2001. ACM.

[10] J. Quinlan. C4.5*: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[11] C. Shannon. A Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27:379–423 and 623–656, 1948.

[12] T. Stocky, A. Faaborg, and H. Lieberman. A Commonsense Approach to Predictive Text Entry. In *CHI ’04: CHI ’04 Extended Abstracts on Human Factors in Computing Systems*, pages 1163–1166, New York, NY, USA, 2004. ACM.

[13] K. Tanaka-Ishii. Word-based Predictive Text Entry using Adaptive Language Models. *Natural Language Engineering*, 13(1):51–74, 2007.

[14] A. Van den Bosch. Scalable Classification-based Word Prediction and Confusible Correction. *Traitement Automatique des Langues*, 46(2):39–63, 2006.

[2] http://www.nuance.com/news/pressreleases/2007/20070824_tegic.asp