Input-based Language Modelling in the Design of High Performance Text Input Techniques

R. William Soukoreff [a]

I. Scott MacKenzie [a,b]

[a] Department of Computer Science
York University
Toronto, Ontario, Canada M3J 1P3
{will, smackenzie}@acm.org

[b] Unit for Computer-Human Interaction (TAUCHI)
Dept. of Computer & Information Sciences
FIN-33014 University of Tampere
Tampere, Finland

Abstract

We present a critique of language-based modelling for text input research, and propose an alternative input-based approach. Current language-based statistical models are derived from large samples of text (corpora). However, this text reflects only the output, or final result, of the text input task. We argue that this weakens the utility of the model, because, (1) users' language is typically quite different from that in any corpus; punctuation symbols, acronyms, slang, etc. are frequently used. (2) A corpus does not reflect the editing process used in its creation. (3) No existing corpus captures the input modalities of text input devices. Actions associated with keys such as Shift, Alt, and Ctrl are missing.

We present a study to validate our arguments. Keystroke data from four subjects were collected over a one-month period. Results are presented that support the need for input-based language modelling for text input.

Key words: Text input, Input-based language modelling

1 Introduction

This paper explores the difficulty of constructing statistical language models for use in text input research. This work is timely because of the tremendous interest, particularly recently [1-19], in developing mobile text input techniques that are optimised or otherwise enhanced by exploiting the statistical properties of a language. Our view is that such efforts are misguided or, at the very least, limited, by not accounting for the way text is input. We will argue that the language model for developing optimised text input techniques should be *input-based* rather than *output-based*, as is the current practice. The importance of input-based text modelling is demonstrated with a small study of keystrokes gathered during text entry on desktop systems. These terms and their implications are explained in more detail later.

Our intent is to demonstrate a methodology for observing the text input process. Further, we hope to stimulate further work toward building input-based models of text entry. We begin with the observation that natural languages are highly redundant.

1.1 Language Redundancy

`Youcanunderstandthissentenceandguesswhat___missingwordsare,becausethereisanenormousamountofredundancyin___Englishlanguage.`

There are 22 spaces, and two three-letter words missing from the above sentence, leaving one hundred and sixteen characters. Yet, the sentence remains understandable. It is true that the phrase is difficult to read and some uncertainty remains about the missing words. But, we understand the message, and so the information content remains unchanged. Arguably, about 19% of the original sentence was unnecessary (28 of 144 characters were removed). This phenomenon is due to the high redundancy in the English language.

The redundancy of a natural language such as English is an important quality that most speakers take for granted. It is like a built-in checksum that is automatically included in every English phrase. Without the checksum, we couldn't understand speech over a noisy telephone line, or proof-read someone else's writing, or use short-hand, or speed-read, or play the word game Hangman. All of these activities require us to fill in missing parts of words or sentences, or to recognise mistakes using only the surrounding context. Language redundancy allows speakers not only to identify mistakes or omissions, but usually to correct them; without the redundancy, omitting part of a sentence would be catastrophic.

The redundancy is a result of the spelling and grammatical properties of the language. Rules like "I follows E except after C", colloquial phrases like "on the other hand…", digraphs like "th" and "sh", and the non-homogeneous frequencies of letters (the letter "e" occurs more often than "q"), give rise to the high redundancy of a language. Each letter, syllable, word, phrase, and sentence is statistically dependant upon not only the context in close proximity, but also on the context extended over great numbers of characters. Consider the following text taken from a paper by Shannon:

> "…anyone speaking a language possesses, implicitly, an enormous knowledge of the statistics of the language. Familiarity with the words, idioms, clichés and grammar enables him to fill in

> missing or incorrect letters in proof-reading, or to complete an unfinished phrase in conversation." [20, page 54]

Shannon wrote "an enormous" instead of "a enormous" which has identical meaning but is grammatically incorrect. The additional "n" doesn't provide any more information; it is only there because the following word begins with a vowel. The "n" depends upon context, and the choice between "a" or "an" depends upon the first letter of the following word, two characters away.

But what if Shannon had instead wrote "*people* speaking a language possess…"? Replacing the singular "anyone" with the plural "people" requires the word "possesses" to become "possess". In this case, the terminal "es" of "possesses" is dependant upon a word 28 characters away. As well, the word "him" in the following sentence doesn't agree in number with "people" and must change, perhaps to "them"; "him" is affected by the word "people" which is 166 characters away and in a different sentence! Clearly the interdependence within English text is not limited to the local context, but can extend to distant parts of other sentences many characters away.

This introduction has demonstrated that natural languages, such as English, are highly redundant and that redundancy is a result of the statistical properties of the language. (Although we are using English as an example in this paper, language redundancy is a property of all natural languages, and we intend our arguments to apply to other languages as well.) We now consider how the redundancy within English is exploited by recent research in text input techniques.

2 Applications of Language Redundancy

There are many areas of research where language redundancy has been applied. Verdu [21] reports that as early as the 14^{th} century frequencies of letters were tabulated to aid in the decryption of secret messages. Redundancy still plays a role in cryptography, has an important role in data compression (where the goal is to remove redundancy via a reversible process), and is of concern in database retrieval.

However, our interest is text input. In general, designers of text input techniques construct statistical models of a language, and these are used in one of two ways. *Movement-minimizing techniques* exploit a language's statistical properties to construct input techniques wherein device or hand movement is as efficient as possible [2, 3, 5-7]. *Predictive input techniques* exploit the same properties, but use a predictive engine that increases text throughput by guessing or suggesting what the user will enter next [9-12]. Examples of both of these approaches and two hybrid approaches [16-18] are discussed in the following sections.

2.1 Movement-Minimising Input Techniques

Although Qwerty reins supreme on the desktop, recent consumer interest in small hand-held personal digital assistants (PDAs) and text messaging on pagers and mobile phones has driven the search for more efficient, portable, text input methods. This interest has produced new keyboard designs that require smaller and more efficient physical layouts, or that can be optimised for stylus (or single-finger) typing instead of two-handed typing. Soukoreff and MacKenzie presented a model [1] which predicts the maximum (expert) and minimum (novice) typing speeds for a given keyboard layout with stylus typing. They calculated character frequencies for the 26 alphabetic characters and the space character, and corresponding digram frequencies for all 27×27 character pairs, and coupled this statistical data with Fitts' law to construct a model of stylus typing. Zhang and MacKenzie used the model to propose a new keyboard layout with four space keys called OPTI [2, 3]. The inventors of the Fitaly keyboard (www.textwaresolutions.com) used an ad hoc optimisation approach to minimise the distance between common digrams. The resulting keyboard contains two space bars and the letters are arranged so that common pairs of letters are physically close to one another. An evaluation of these and other keyboards was presented by MacKenzie, Zhang, and Soukoreff [4]. Hunter, Zhai, and Smith [5] took a physics-based approach to find an optimised keyboard arrangement. They adopted the model of Soukoreff and MacKenzie and attempted to optimise the model using mechanical simulation and a Monte Carlo approach known as the Metropolis method. Their Metropolis Keyboard has hexagonal keys and is centred around the space key. In further work [6, 7], they compare their Metropolis keyboard to other common layouts, and present a user evaluation. For a more detailed review of these and other text input technologies see [8]. Clearly, statistical language data are useful to researchers trying to optimise movement during text entry.

2.2 Predictive Input Techniques

Language prediction has been of interest since well before Shannon's *Prediction and Entropy of Printed English* [20], the seminal work in language prediction research. In Jonathan Swift's *Gulliver's Travels,* published in 1726, in the *Letter from Capt. Gulliver to his Cousin Sympson* in the *Forward Matter*, Gulliver describes a machine controlling a printing-press that can automatically construct academic writing in any field using "proportion data" of the parts of speech. (It would save researchers a lot of work if such a machine were available today!)

An interesting real technology utilising prediction to aid in text input is POBox. POBox [9, 10] allows users to enter part of a word and then search for similar words by spelling, pronunciation, or shape (for pictograph-based languages). It uses a static database for the searching functionality.

Another predictive input technology is the Reactive Keyboard [11, 12]. The Reactive Keyboard monitors what a user enters and presents text predictions for the user to choose from. The predictions are generated by finding longest matching sub-strings in the previously entered text. So, not only is the Reactive Keyboard predictive, but it adapts to the user's input and hence is not limited to a static set of words or phrases.

One final predictive text input technology deserves mention because the technology has been licensed widely and is currently available on PDAs and cellular telephones. The T9 input technology (www.tegic.com) uses a conventional telephone keypad, which is significant because the telephone keypad is ubiquitous and the letter arrangement is defined by an international standard [13]. As the user types, the text is not completely deterministic, because each keypress does not map to a single letter, but, instead, to one of three or four letters on each key. To overcome the ambiguity, word possibilities are presented as the user enters text. Although the disambiguating algorithm is proprietary, it likely uses character and word frequencies. Three evaluations of T9 exist [13-15].

Predictive text input technologies rely heavily upon statistical language models, and their ultimate performance will always be limited by the quality of the model.

2.3 Hybrid Input Techniques

Some text input techniques have been developed with both movement-minimizing and predictive features. Dasher [16, 17] is a predictive text input technique using a pointing device to select from predicted options. The options are presented to the user in boxes sized according to their relative probabilities, to optimise the movement time. The boxes expand as the pointing device hovers near them using video-game-like animation for fast text entry. Thus, the technique is both movement-minimizing and predictive. The technique is difficult to visualise without a demonstration. Conveniently, a demo is available online (http://www.inference.phy.cam.ac.uk/dasher/). Dasher uses a fixed set of character probabilities calculated with a prediction and partial match (PPM) algorithm.

Fluctuating optimal character layout (FOCL) [18] is a hybrid text input technique designed for text entry on small devices with a limited display (only three rows of text) and five buttons. Four cursor keys and an enter button are used to scroll around a 3 × 12 (approximate) grid containing all of the characters that can be entered. The characters are rearranged after each keystroke so as to minimise the number of cursor movements to select the next character. Rearrangement is guided by digram probabilities and knowledge of the previous character.

The text input techniques reviewed vary widely, and yet all (with the exception of Dasher [16, 17]) rely on fixed tables of language statistics culled from language models. Any inaccuracy in these models results in reduced performance of the interface. All of the techniques, including Dasher, focused on rapid inputting of text, and none provided the editing functionality that desktop users take for granted such as backspace, delete, shift, caps-lock, etc.

3 Caveats of Language Modelling

With respect to the input process, statistical language models are constructed in an attempt to capture the relative likelihood of letters, words, and phrases as they relate to all possible contexts as a user types. Text input researchers put a lot of effort into building their statistical models, often overlooking the importance of selecting the right "collection of symbols" from which to calculate the statistics of their model. While input technologies differ significantly in approach, they are similar in that they are all limited by the quality of the language model used. The quality of the language model, in turn, is limited by the body of raw text or symbols the model is based on.

Typically, researchers needing statistical language data construct or adopt a corpus of text from which to build their statistical models. A corpus is constructed by obtaining large quantities of text in the language of interest. Project Gutenberg (http://promo.net/pg/) is an on-line repository of electronic documents (mostly English), and is a common source of these texts. Alternatively, some researchers use published tables of statistical language data [22, 23]. These Project Gutenberg texts and statistical language tables are well suited to research in database retrieval, text compression, and other areas of applied linguistic research. All of these areas of research, however, use "finished" documents. By this we mean the text that remains after the text input task is over. However, text input research is different, because, to be effective it must focus on the dialog that occurs between the user and the input device *during* the text entry task. We will elaborate on this point in the following section.

3.1 Input-based Language Modelling

When constructing language models for text input research statistical data are typically derived from the output (final static product) of text entry. We suggest that the data should be obtained from the input (the creation process), because the goal is to exploit redundancy in a language in optimising text *input*. Therefore, for text input research, a corpus should be constructed from input text and editing actions captured during the creation process with a technology as similar as possible to the text input method under investigation. We have three reasons for making this recommendation, (1) corpus text is not representative of the user language, (2) corpus text does not reflect the editing process, and, (3) corpus text does not capture input modalities. A detailed explanation of these points follows.

Corpus Text is Not Representative of User Language

The idea that a corpus is "representative of a language" is naive when the domain is users interacting with computing technology. Users typically use a much richer set of characters than appear in any corpus, and the statistical properties in that set are distinctly different from that in typical corpora. A simple example is the space key, which is the most common character in English text [1]. Yet, the space character is typically missing in tables of letter or digraph probabilities used to build language models (e.g., [22, 23][1]). Typically, only alphabetic characters are included.

As well, punctuation symbols are rarely included in letter or digraph tables. Both Isokoski [19] and Zhai et al. [6, 7] observe that some punctuation occurs more frequently than some of the less frequent letters. Inclusion of the space character and simple punctuation symbols is the first step, but we feel it is important to fully open the character set.

The characteristics of the text users enter are dependent on the application used to create the text. For example, we expect more formal prose to be entered using a word processor than an e-mail application. Additionally, the type of application depends upon the input device available – few people have the patience to enter volumes of text into a hand-held PDA device. The kind of text most likely entered in this context is short notes, phone numbers, URLs, acronyms, slang, etc., the statistical properties of which differ from formal English texts. Highly cryptic messages are common for text entry on cell phones and pagers.

Corpus Text Ignores the Editing Process

A corpus contains no information about the editing process, and we feel this is an unfortunate omission. Users are fallible and the creation of a text message – or interaction with a system on a larger scale – involves much more than the perfect linear input of alphanumeric symbols. The input process is really the editing process. Users do not produce text – they produce a series of editing commands that are interpreted by a text-box widget, wordprocessor, or command-line interface, resulting in text. [2]

We define *input actions* as the set of physical acts that the user can perform during the editing process such as hitting a key, using a mouse, writing with a stylus, depending upon the system used. Input actions result in *tokens*. A token is an instruction to the application. For example, when using a word processor, typing the letter "a" on the keyboard is an input action, the token is the instruction "insert the letter 'a' at the current cursor position". In this example, the input action produces a token that has an effect upon the final document. However, consider what happens as a user of Microsoft Word on the Windows operating system presses Alt-f, and then the letter "s". There are four input actions:

1. Press the Alt key
2. Press and release the "f" key
3. Release the Alt key
4. Press and release the "s" key

These keystrokes result in two tokens. Actions 1 through 3 cause the file menu to be displayed, and the word processor to enter a mode wherein it interprets the following keystroke as a selection from this menu. Action 4 creates the second token which is the instruction to the application to save the current open document, and then return to the normal editing mode. In this example, there were three keystrokes and no changes to the document itself; the text of the document does not contain a record of these keystrokes. When this happens, we say the result is *unrepresented keystrokes*.

An observation about this example is that it includes three unrepresented keystrokes for a very common task. Currently, the proportion of unrepresented keystrokes to total keystrokes that occurs during text input is unknown, but we believe that the proportion is probably high. Consider the unrepresented keys on the typical 101-key PC Qwerty keyboard: backspace, shift, control, caps-lock, escape, alt, cursor keys (including page-up, home, etc.), function keys, num-lock, insert, delete, and system keys like print-screen. Some unrepresented keys such as alt

[1] Although the linguistic data provided by Mayzner and Tresselt [23] does not include the space character, it does indirectly provide some data on spaces. Frequency data for alphabetic characters appearing at the beginning and end of words is presented. It is possible to obtain an estimate of the frequency of the space character from these data. Such an analysis [1] suggests that the frequency of space characters was 18%.

[2] The word "edit" can be used to describe two different processes. It can mean to correct minor typing mistakes, or to describe the larger task that a newspaper editor performs, improving a text document by reorganising and rewriting sections of it. Here, we mean both. Users need to be able to correct minor mistakes, and to perform larger editing tasks.

and control usually work in conjunction with other keys which also then become unrepresented.

One final observation is that it is the application (and the operating system) that determines the tokens, and hence the input actions, that are unrepresented.

Corpus Text Does not Capture Input Modalities

Text documents do not account for how they were created. For example, a corpus includes both uppercase and lowercase characters. In simple language models this distinction is ignored (e.g., "A" and "a" are considered the same). A more expansive model can easily accommodate this distinction simply by treating uppercase and lowercase characters as distinct symbols. Yet, from the input perspective, both approaches are wrong. Uppercase and lowercase characters are never entered via separate keys on a keyboard; thus, the seemingly more accurate treatment of uppercase and lowercase characters as distinct symbols is just as wrong from the input perspective.

For the user's interaction with the shift and caps-lock keys to be accommodated in a model of text input, activity with these and related keys must be captured during the creation of text. In the terminology of the previous section, we say that the mode shift keys are *partially unrepresented keystrokes*; the shift key itself is not recorded in text, but the mode change to capital letters is evident in the final body of text. However, the capital letters in a text document only indicate that the shift mode was activated. The capital letters do not indicate whether the mode shift was activated with the caps-lock, left shift, or right shift keys on the keyboard, so the shift state is partially unrepresented.

The partial unrepresentation of the shift keys is a natural consequence of the input modalities of the technology – in other words, the multiple ways of enabling the shift mode with a Qwerty keyboard.

3.2 Currently, There Is No Text Input Corpus

We hope that our arguments have convinced the reader of the necessity of considering the input stream, as opposed to static text documents, when modelling text input. Just as it would be pointless to try to build a language model of the English language using a corpus containing only French text, building a model of text entry using only completed text documents is similarly misguided. However, no corpus of text input exists.

Next we present the results of a study we conducted. The purpose of the experiment was not to produce a new corpus of text input-stream data, but to justify the arguments made earlier in this paper, and to obtain some data. This study is preliminary; production of a text input stream corpus is a worthwhile endeavour, although not one that we have yet taken on.

4 A Study of a Typical Text Input Stream

We constructed software that records what users type as they go about their regular activities using computers. We used this software to collect keystroke data from four subjects for approximately one month.

4.1 Materials and Method

Software

To capture raw input text data, we wrote the KeyCapture software using Visual C++, for the Windows 98 operating system. This software takes advantage of several hooks in Windows to log user interaction at a low level. The hooks enable KeyCapture to receive keystroke data (both key presses and releases), mouse movement and mouse button clicks, before the active application receives this data. KeyCapture also monitors the window focus to record the active application during typing. The user's keyboard and mouse activity are recorded to a log file, and each entry is time-stamped with millisecond resolution. The KeyCapture software was designed so that when it is operating (recording the user's actions) it is essentially invisible, it has no open windows and no entry in the task bar, so the user is not aware of it or disturbed by it. We anticipate that the KeyCapture software will be of interest to others, and so the software with its source code are available at http://dynamicnetservices.com/~will/academic/textinput.

The reader may be surprised to find that we used desktop computers to gather our text input data, after identifying handheld computers as the focus of contemporary text input research. We found it very difficult to find subjects willing to have their keystrokes logged. As well as mundane text communications, our software captured passwords, personal e-mails, and confidential letters. The difficulty in finding enough willing subjects, with similar PDAs, that entered sufficient text using their PDAs, for an exploratory study, led us to this decision.

Participants

Five volunteer participants were solicited, although one coincidentally stopped using his computer just as we began our study and so we considered his data insufficient. Of the remaining four subjects, 3 were male, 1 was female; ages ranged from 25 to 45 years. All participants are computer literate and use their computers on a regular basis. Three subjects were frequent users of standard desktop applications, such as email, word processing, and web browsing. One subject spent almost all of his time using Visual Basic; the three others split their time between word processing and e-mail.

Although this is a small number of users, we feel the data collected is sufficient for this preliminary study of text-input-based language modelling.

4.2 Results and Discussion

In total approximately 590 megabytes of log files were collected over the course of one month, corresponding to approximately 360,000 individual keystrokes, and almost seven million mouse events (movements and button clicks).

Applications Used

In analysing the data from our study, we calculated the percentage of keystrokes users typed into each of the applications they used. Table 1 presents the most frequently used applications, and the proportion of keystrokes received by these applications. The "Notes" column indicates applications other than word processing and e-mail, that received a large percentage of subjects' keystrokes. The data in Table 1 suggest that word processing and e-mail consumed a large number of the total keystrokes entered by our subjects.

Keystroke Frequency Data

We used the keystroke data files generated by our study to calculate keystroke frequency data. Table 2 presents the top 15 keystrokes entered by each subject, and in total. Notice that the second most probable character, accounting for 6.74% of the characters typed by our subjects, was the backspace key. This is an important result. Earlier we reported several recent keyboard designs using statistical language modelling, none of which optimised for the second most probable key – the backspace key.

It is interesting that every subject other than s2 had backspace as the second most frequent keystroke. We investigated this and found that while the other subjects were heavy users of the backspace key, s2 preferred to use the cursor keys, control key, and shift key to select errant text for correction. This accounts for s2's higher than typical prevalence of cursor keys. In general, this suggests that the second most frequent key is an editing key, either backspace or a cursor key, depending upon the editing style of the user. This also demonstrates one strength of input-based analysis – we have identified two quite different editing strategies. This observation would have gone unnoticed if output based text analysis was performed.

The importance of distinguishing upper and lower case alphabetic characters is illustrated by the high frequency of the shift key. It was the sixth most frequent key, accounting for slightly over 4% of the keystrokes.

Our data show that several punctuation marks are more likely than some of the less frequent letters, which agrees with observations by Isokoski [19] and Zhai [6, 7]. The most probable punctuation character is the period with a frequency of 1.4% making it more likely than P, W, F, Y, G, V, B, comma, apostrophe, K, X, tab, hyphen, Z, semicolon, J and Q (listed here in order by decreasing frequency).

Unrepresented Keystrokes

Table 3 supports our contention that a typical corpus does not account for the editing process, as it misses a great deal of input activity. Of the subjects' keystrokes, 31.4% of them are keys classified as unrepresented. This implies that corpora not based on data captured during the input process are missing 31.4% of the keystroke input. Additionally, several unrepresented keys (such as control, alt, delete and backspace) cause other regular alphabetic keystrokes to become unrepresented, so the 31.4% statistic presented in Table 3 is a lower-bound for the proportion of unrepresented keystrokes.

Our measure of the relative occurrence of the space keystroke was 9.69% – lower than the 18% in common English cited earlier. This could be a result of corpora [22, 23] not accounting for unrepresented characters, or possibly our subjects used words with a longer average length (and hence fewer spaces per character).

Table 1 - Application Usage by Percent Received Keystrokes

Subject	Total Keystrokes	Word Processor	E-mail	Other
s1	31,328	55%	25%	
s2	82,875	30%	16%	42% C coding
s3	232,372	26%	66%	
s4	10,148	0% †	0% †	75% Visual Basic 13% Netscape
Average		28%	27%	

† Subject s4 did not use his computer for e-mail or word processing, once the KeyCapture software was installed.

Table 2 - Frequencies of the Fifteen Most Common Keystrokes

s1		s2		s3		s4		All	
9.18	Space	10.42	Down	12.87	Space	8.75	Space	9.69	Space
7.14	Back	7.95	Space	8.69	Back	8.72	Back	6.74	Back
5.29	Down	5.57	Up	7.36	E	4.85	E	5.28	E
4.93	E	5.35	Shift	6.07	T	4.39	Down	5.13	Down
4.19	A	5.33	Right	5.05	O	4.29	Return	4.15	T
3.85	Shift	4.49	Control	4.64	I	4.01	T	4.14	Shift
3.84	I	4.00	E	4.45	A	3.89	Shift	3.72	O
3.42	O	3.96	Left	4.18	S	3.88	O	3.68	I
3.28	T	3.73	Delete	4.16	N	3.83	I	3.62	A
3.27	R	3.25	T	3.79	R	3.57	R	3.31	S
3.22	N	3.12	S	3.46	Shift	3.31	A	3.17	N
2.98	Up	2.54	O	2.68	H	3.21	S	3.16	R
2.92	Right	2.54	A	2.32	L	3.18	N	2.66	Up
2.72	S	2.42	Back	2.24	C	2.84	D	2.62	Return
2.48	Delete	2.38	I	2.14	D	2.26	H	2.30	Right
							Remaining Represented Keystrokes:	28.86 %	
							Remaining Unrepresented Keystrokes:	7.76 %	

Table 2: The numbers represent the proportion of each keystroke indicated, as a percentage. For example, the number 9.18 for "space" under s1 means that 9.18% of the keystrokes entered by s1 were spaces. The keystrokes are ordered by their frequency of occurrences. The percentages for the subjects do not sum to 100% because this table only shows the most frequent 15 characters.

Table 3 - Frequency of Unrepresented Characters

Frequency (%)	Unrepresented Key
6.7	Back
5.1	Down
4.1	Shift
2.7	Up
2.6	Return
2.3	Right
2.1	Delete
1.7	Left
1.6	Control
0.8	Next
1.6	*The Rest*
31.4	*Total*

A revealing calculation to perform is the normalisation of the space keystroke frequency. From Table 3 it is apparent that no more than[3] (100 - 31.4 =) 68.6% of keystrokes are represented. Of these, 9.69% were spaces. Thus, if the final text was used to measure the frequency of the space character it would find a frequency of at least (9.69 / 68.6 =) 14.1% for the space character. If we assume that each backspace, delete, and control character cause one represented keystroke to become unrepresented, then (68.6 - 6.7 - 2.1 - 1.6 =) 58.2% of keystrokes remain represented, and 16.6% of these are spaces. Thus, our 9.69% figure is likely the result of both factors, corpora omitting unrepresented keystrokes, and subjects entering longer words.

5 Conclusions

While we do not present a novel statistical model of English here, nor a new comprehensive corpus text of input data; we suggest that output-based models are lacking in their ability to represent the text input task. The output-based approach makes the models weak or unsuitable for research in text input problems for the following reasons: (a) typical text files used in language modelling are not representative of the user language, (b) these texts contain no data from the editing process, and (c) the text corpus does not account for the input modalities of the technology used to create the text.

The utility of input text language models can be improved if the corpus is collected using an input task

[3] We say "no more than" because delete, backspace, alt, and control keystrokes cause other keystrokes to become unrepresented. So the percentage of represented keystrokes is actually lower than this figure.

similar to that which the users will ultimately face in terms of both the physical device used, and the type of text entered.

We would very much like a corpus of text input data to be created and made available to the research community. However, this would be difficult; it is hard to find subjects willing to have their personal communications made public. Yet such a corpus would be invaluable to the text input community. Until such a corpus becomes available, capturing text input data with tools like the KeyCapture software is possible.

References

1. Soukoreff, R. W. and MacKenzie, I. S. (1995). Theoretical upper and lower bounds on typing speeds using a stylus and soft keyboard. *Behaviour & Information Technology*, 14(6), 370-379.
2. Zhang, S. X. (1998). *A high performance soft keyboard for mobile systems* (M.Sc. Thesis). University of Guelph.
3. MacKenzie, I. S. and Zhang, S. X. (1999). The design and evaluation of a high-performance soft keyboard. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI '99*, 25-31. New York: ACM.
4. MacKenzie, I. S., Zhang, S. X., and Soukoreff, R. W. (1999). Text entry using soft keyboards. *Behaviour & Information Technology*, 18(4), 235-244.
5. Hunter, M., Zhai, S., and Smith, B. A. (2000). Physics-based graphical keyboard design. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems - CHI 2000*, 157-158. New York: ACM.
6. Zhai, S., Hunter, M., and Smith, B. A. (2000). The Metropolis keyboard: An exploration of quantitative techniques for virtual keyboard design. *Proceedings of the ACM Conference on User Interface Software and Technology - UIST 2000*, 119-128. New York: ACM.
7. Zhai, S., Hunter, M., and Smith, B. A. (2002). Performance Optimization of Virtual Keyboards. *Journal of Human Computer Interaction*, 17(2 & 3), 229-270.
8. MacKenzie, I. S. and Soukoreff, R. W. (2002). Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17(2 & 3), 147-198.
9. Masui, T. (1998). An efficient text input method for pen-based computers. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI '98*, 328-335. New York: ACM.
10. Masui, T. (1999). POBox: An efficient text input method for handheld and ubiquitous computers. *Proceedings of the International Symposium on Handheld and Ubiquitous Computing - HUC '99*, 289-300.
11. Darragh, J. J., Witten, I. H., and James, M. L. (1990). The reactive keyboard: A predictive typing aid. *Computer*, 23(11), 41-49.
12. Darragh, J. J. and Witten, I. H. (1991). Adaptive predictive text generation and the reactive keyboard. *Interacting with Computers*, 3(1), 27-50.
13. Silfverberg, M., MacKenzie, I. S., and Korhonen, P. (2000). Predicting text entry speed on mobile phones. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 2000*, 9-16. New York: ACM.
14. Bohan, M., Phipps, C. A., Chaparro, A., and Halcomb, C. G. (1999). A psychophysical comparison of two stylus-driven soft keyboards. *Proceedings of Graphics Interface '99*, 92-97. Toronto, Ontario: Canadian Information Processing Society.
15. James, C. L. and Reischel, K. M. (2001). Text input for mobile devices: Comparing model prediction to actual performance. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 2001*, 365-371. New York: ACM.
16. Ward, D. J., Blackwell, A. F., and MacKay, D. J. C. (2000). Dasher - A data entry interface using continuous gestures and language models. *Proceedings of the ACM Conference on User Interface and Software Technology - UIST 2000*, 129-137. New York: ACM.
17. Ward, D. J., Blackwell, A. F., and MacKay, D. J. C. (2002). Dasher: A Gesture-Driven Data Entry Interface for Mobile Computing. *Journal of Human Computer Interaction*, 17(2 & 3), 199-228.
18. Bellman, T. and MacKenzie, I. S. (1998). A probabilistic character layout strategy for mobile text entry. *Proceedings of Graphics Interface '98*, 168-176. Toronto, Ontario: Canadian Information Processing Society.
19. Isokoski, P. (1999). *A minimal device-independent text input method* (M.Sc. Thesis). University of Tampere, Tampere Finland.
20. Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, 30, 51-64.
21. Verdu, S. (1998). Fifty years of Shannon theory. *IEEE Transactions on Information Theory*, 44(6), 2057-2078.
22. Underwood, B. J. and Schulz, R. W. (1960). *Meaningfulness and verbal learning*: Philadelphia: Lippincott.
23. Mayzner, M. S. and Tresselt, M. E. (1965). Table of single-letter and digram frequency counts for various word-length and letter-position combinations. *Psychonomic Monograph Supplements*, 1(2), 13-32.