Measuring Errors in Text Entry Tasks: An Application of the Levenshtein String Distance Statistic

R. William Soukoreff
Dept. of Computer Science, York University
Toronto, Ontario
Canada M3J 1P3
will@acm.org

I. Scott MacKenzie
Dept. of Computer Science, York University
Toronto, Ontario
Canada M3J 1P3
+1 416-736-2100, smackenzie@acm.org

ABSTRACT

We propose a new technique based on the Levenshtein minimum string distance statistic for measuring error rates in text entry research. The technique obviates the need to artificially constrain subjects to maintain synchronization with the presented text, thus affording a more natural interaction style in the evaluation. Methodological implications are discussed, including the additional need to use keystrokes per character (KSPC) as a dependent measure to capture the overhead in correcting errors.

Keywords

Levenshtein minimum string distance, text entry, error rates

INTRODUCTION

Research in text entry is thriving, in part due to the demand for text messaging on mobile phones, and more generally to the on-going need for text entry on PDAs, pagers, and electronic organisers. With each new text entry technique considered, however, an empirical evaluation with users is paramount in establishing its viability.

Text entry evaluations usually focus on two statistics, speed and accuracy. Although the calculation of entry speed is straightforward, accuracy is another matter. Even, the intuitively simple measure "percent errors" is problematic unless entry is constrained by forcing the subject to maintain synchronization with the presented text. In this paper we present a robust method for measuring character-level accuracy that avoids the need for artificial constraints in the text entry task. The accuracy measure we present is based on a minimum string distance statistic.

TEXT INPUT RESEARCH

Text input research seeks to create and evaluate novel text input technologies. Empirical evaluations are conducted to measure speed and accuracy under controlled conditions. Repeated trials are necessary, yet generate great volumes of paired data, consisting of *presented text* (what subjects were asked to enter) and *transcribed text* (what they actually entered). From these data, speed and accuracy are calculated and analysed.

Speed

Because of the speed-accuracy tradeoff, evaluations must attend to both the speed of entry and the accompanying errors. Measuring speed is easy. Software records the number of characters entered, the elapsed time, and computes the entry rate in characters per second (cps) or words per minute (wpm). This contrasts sharply with the difficulty in measuring accuracy.

Accuracy

To measure accuracy, the transcribed and presented texts are compared and errors are identified. Using a computer for this task is preferred because of the volume of data and the need for consistency. However, automated error tabulation is tricky. Consider the following:

```
quick brown fox    (presented text)
quixck brwn fox    (transcribed text)
   ^^^^^^
```

How many errors are in the transcribed text? A simple character-wise comparison suggests six (shown). More likely, however, there are two: an insertion error "x", and an omission error "o". Although a human can estimate the number of errors, delegating the task to software is deceptively difficult. In view of this, experimenters usually adopt a methodology that (a) artificially constrains the task to keep subjects in sync with the presented text, (b) measures only word-level errors, (c) uses an ad hoc categorization of errors, (d) discards trials with errors, or (e) ignores errors. Space precludes an exhaustive review, but see [3].

In the next section, we present an algorithm applied in DNA analysis, phylogenetics, spelling correction, and linguistics, to measure the distance between two strings.

MINIMUM STRING DISTANCE

String research emerged from theoretical work on self-correcting binary codes. The algorithm presented is credited to Levenshtein [2], however it was discovered independently by at least nine researchers [1]. The algorithm yields the minimum distance between two strings defined in terms of editing primitives. The primitives are *insertion*, *deletion*, and *substitution*. Given two character strings, the idea is to find the smallest set of primitives that applied to one string produces the other. The number of primitives in the set is the minimum string distance (MSD).

As an example, consider the presented text `abcd` and the transcribed text `acbd`. There are three sets each containing the minimum number of primitives (two) to perform the transformation from transcribed to presented text:

Published in *Extended Abstracts of CHI 2001*, pp. 319-320

Set 1: delete the c, insert a c after the b
Set 2: insert a b after the a, delete the second b
Set 3: substitute b for the c, substitute c for the second b

There are three sets and MSD = 2. This simple example illustrates the challenge in designing a computer algorithm to compute the minimum set: there is often more than one minimum set, and the path through the possibilities is mired by the quantity, the type, and the location of errors.

The minimum string distance, denoted *MSD(A, B)*, where *A* and *B* are character strings, is a well-behaved statistic with several properties making it a suitable metric.

- Well-defined zero: $MSD(A, B) = 0$, if and only if $A \equiv B$
- It is bounded: $0 \le MSD(A, B) \le max(|A|, |B|)$, where $|A|$ denotes the length of *A*
- It is commutative: $MSD(A, B) = MSD(B, A)$

One final feature of the MSD statistic is the ease with which it may be calculated.

Calculating the Minimum String Distance

The algorithm we present is from [1].

function **r(x, y)**
  if x = y return 0
  otherwise return 1

function **MSD(A, B)**
  for i = 0 to |A|
    D[i, 0] = i
  for j = 0 to |B|
    D[0, j] = j
  for i = 1 to |A|
    for j = 1 to |B|

$$D[i, j] = \min\begin{pmatrix} D[i-1, j]+1 \\ D[i, j-1]+1 \\ D[i-1, j-1]+r(A[i],B[j]) \end{pmatrix}$$

  return D[|A|, |B|]

D

B \ A=	a	b	c	d
	0	1	2	3	4
a	1	0	1	2	3
c	2	1	1	1	2
b	3	2	1	2	2
d	4	3	2	2	**2**

***Lev(A,B)* = 2**

Computing the entries in the matrix D starts in the top-left cell and proceeds to the bottom-right. The value in the bottom-right cell is the minimum string distance.

TEXT ENTRY ERROR RATE

Using the MSD statistic we propose the following definition of text entry error rate, given a presented text string (*A*) and a transcribed text string (*B*):

$$ErrorRate = \frac{MSD(A,B)}{max(|A|,|B|)} \times 100\% \quad (1)$$

Calculated using Eq. 1, the error rate represents the smallest proportion of characters considered errors given the two strings. It is important to use the larger of the two string sizes in the denominator to ensure (a) the measure does not exceed 100%, (b) undue credit is not given if the user enters less text than presented, and (c) an appropriate penalty is attributed if the user enters more text than presented. Some examples follow:

Presented/Transcribed Text	Length	MSD Error	Rate
`the quick brown fox` [a]	19	-	-
`thiquick brown fox` [b]	18	2	10.5%
`the quicj beown fix` [b]	19	3	15.8%
`the quick brown foxxx` [b]	21	2	9.5%

[a] presented text, [b] transcribed text

METHODOLOGICAL IMPLICATIONS

Using the MSD error rate removes the need to artificially constrain the task during text entry evaluations. Subjects may be directed to proceed "quickly and accurately"; however, there is no need for strict use of procedures such as "ignore errors", "correct all errors", or "maintain synchronization with the presented text". As demonstrated, Eq. 1 produces a meaningful error metric even in the presence of various degradations in the transcribed text.

If some, but not necessarily all, errors are corrected, the task is indeed more natural. However, a new twist is added that must be accounted for, lest we replace one problem with another. Now, there are two kinds of errors: those corrected and those remaining. The MSD error rate (Eq. 1) is appropriate for the latter, but what about the former? For this, we propose "keystrokes per character" (KSPC). Although KSPC has been used as a *characteristic* of interaction techniques, we suggest using KSPC as a *dependent measure*. Here is a simple example:

```
the quick brx<own fox → the quick brown fox
             ^^
```

The user committed one error, entering 'x' instead of 'o', but correcting took two keystrokes, BACKSPACE (<) followed by 'o'. Twenty-one keystrokes produced 19 characters, so KSPC = 21 / 19 = 1.11. KSPC will rise for less skilled users or if more effort is invested in correcting errors.

CONCLUSION

We proposed a new metric — the MSD error rate — for text entry error rates, based on the minimum distance between the presented and transcribed texts. This measure obviates the need for artificial constraints in the task, thus improving the methodology in empirical evaluations of text entry techniques. Additionally, keystrokes per character (KSPC) should be used as a dependent measure to capture the overhead in correcting errors as text is entered.

REFERENCES

[1] Kruskal, J.B. An overview of sequence comparison, *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*, ed. D. Sankoff, and J.B. Kruskal. (Reading, MA: Addison-Wesley, 1983) 382.

[2] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions and reversals, *Soviet Physics-Doklady 10* (1966), 707-710.

[3] MacKenzie, I. S., and Zhang, S. X. The design and evaluation of a high-performance soft keyboard, *Proceedings of CHI '99*, 25-31.

Published in *Extended Abstracts of CHI 2001*, pp. 319-320