Error Quantifying Metrics for Text Entry Systems Augmented with Word Prediction

Manoj Kumar Sharma
School of Information Technology
Indian Institute of Technology Kharagpur
manojsharma.net@gmail.com

Pradipta Kumar Saha
School of Information Technology
Indian Institute of Technology Kharagpur
itsmepradipta@gmail.com

Sayan Sarcar
School of Information Technology
Indian Institute of Technology Kharagpur
mailtosayan@gmail.com

Debasis Samanta
School of Information Technology
Indian Institute of Technology Kharagpur
dsamanta@iitkgp.ac.in

ABSTRACT
Of late, many text entry systems in users' languages with various text entry rate enhancement strategies are being proposed. To evaluate the effectiveness of such text entry systems, measuring error correction efficiency in addition to text entry rate have been advocated by researcher. Existing metrics for evaluating text entry errors are found inaccurate to evaluate text entry systems augmented with word prediction. This work attempts to bridge this gap. In this work, we redefine existing error classes as well as error quantifying metrics. In addition to this, we also introduce five different errors classes and six new metrics relevant to text entry error evaluation in the context of text entry systems augmented with word prediction. We substantiate the validity of error classes and efficacy of the metrics with a sufficient number of instances.

Author Keywords
Text entry error evaluation, word prediction, error quantification metric, text entry systems

ACM Classification Keywords
H.5.2 User Interfaces: Evaluation/methodology, theory and methods

General Terms
Human factors, measurement, performance

INTRODUCTION
The advent of new text entry mechanisms creates the necessity to establish the strong foundation of rapid and accurate text entry with digital devices. The performance of a text entry task can be measured by maintaining a log file of user activity, updated through background program. The log file is further analyzed to identify character level errors committed by users. The analysis is quantified by some metrics which work on a basic assumption that correction should be made through backspace key in the keyboard, without using mouse or other pointing devices.

APCHI '13, September 24 - 27 2013, Bangalore, India

http://dx.doi.org/10.1145/2525194.2525201

Earlier, Soukreff and Mackenzie [9] propose a technique based on the Levenshtein minimum string distance statistic for measuring error rate. Their work defines two metrics namely, Minimum String Distance (MSD) and Keystroke per Character ($KSPC$). Mackenzie further discusses the calculation of KSPC over a variety of text entry methods [5] (where $KSPC = or < or > 1$). However, for word prediction system, the calculated *KSPC* value is less than 1 whereas for *QWERTY* and multi-tap keyboard, the values are equal to and greater than 1, respectively. Furthermore, Soukreff and Mackenzie, in 2002 [6], modify the *Error Rate* calculation formula [9]. There, Per-character errors are categorized as *insertions*, *substitutions*, or *deletions*, by analyzing the alignments and applying a weighting factor. Soukoroff et. al. also identify the shortcomings of both *MSD* and *KSPC* metrics [10]. They propose a framework combining the analysis of the *presented text*, *input stream* and *transcribed text*. Their paper describes a unified *total error rate* by combining two constituents, the *corrected error rate* and the *not corrected error rate* [11]. The framework includes other measures like error correction efficiency, participant conscientiousness, utilized bandwidth, wasted bandwidth etc [7]. Wobbrock and Myers [13] extend the character-level error analysis to the input stream by dealing with *corrected character-level errors*, not just *uncorrected errors*. They propose algorithms for automatically detecting and classifying character-level input stream errors. Akiyo Kano et al. [4] describe a detailed categorization on typing errors made by children during a text copy exercise.

Word prediction [3] is one of the popular text entry rate enhancement strategies where it predicts word based on the character(s) or word(s) user has already typed. Thus, word prediction reduces the keystrokes required to compose text completely [12]. Advanced word prediction systems like POBox [8], VITIPI [1] etc. also support error detection and

correction during text entry. An example of text composition using word prediction is shown in Table 1.

Table 1. An example of text composition with word prediction

Presented text :	"overall_"
Input stream :	"oue***ra***<<<<ve(overall_)"
Transcribed text :	"ove***ra***ll_"

In the text entry system without prediction, the input stream (*IS*) usually contains more data than the transcribed string (*T*). Existing error evaluating metrics are suitable for evaluating text entry systems where users enter texts character by character. To compute the metrics, the keystrokes present in *IS* are categorized into four classes namely, *Correct* (C), *Incorrect and Not Fixed* (INF), *Fixed* (F) and *Incorrect but Fixed* (IF) [10]. On contrary, in text entry systems augmented with rate enhancement strategies like word prediction supported with or without error correction, more than one character are inserted on a single click. So, these classes become unsuitable for text entry systems augmented with word prediction. Moreover, in text entry systems augmented with word prediction, committed errors can be corrected by the user as well as the system. Thus, it is quintessentially important to redefine the existing metrics and propose new to resolve the issue. Consequently, the metrics also requires redefinition as well as inclusion of new ones. In this paper, we propose five different error classes and six new metrics suitable for measuring error in text composition.

The rest of the paper is organized as follows. Section 2 describes our approach to evaluate text entry system supported with word prediction. Section 3 presents the analysis of the proposed metrics. Finally, Section 4 concludes the paper.

PROPOSED APPROACH

In this section, first we define few terminologies used later in this paper. Next, we discuss the existing error classes followed by the proposed classes in the context of text entry system augmented with word prediction. Then, we propose new metrics which can capture newly identified classes keeping the existing one. Finally, we describe our approach to compute these classes and metrics.

Terminologies

The terminologies used in our proposed approach are listed below.

$T_1 \rightarrow$ Target text $T_2 \rightarrow$ Final composed text

$T_{log} \rightarrow$ Content of log file $iLog \rightarrow$ Intermediate log

$CN_1 \rightarrow$ Compact notation 1 $CN_2 \rightarrow$ Compact notation 2

$\alpha \rightarrow$ Deleted text $\beta \rightarrow$ Typed text after α

$\psi \rightarrow$ Final word after modification

$CDDTable \rightarrow$ Stores multiple modified data

$IndexTable \rightarrow$ Stores starting index and count of backspace

$DataTable \rightarrow$ Stores value of different error classes

In this paper, '_' represents the blank space between word, '$<$' denotes backspace and '$*$' signifies the word selected from the prediction list. The words within '$(\cdots)$' are represented as the predicted word.

Error Quantifying Classes

There are multiple error classes present to evaluate the text entry system [7]. The description of each class is given below.

1. C: The *C* class indicates the keystrokes of correct characters present in the typed text.
2. INF: This class contains wrongly entered keystrokes present in final text.
3. F: The keystrokes required for corrections are considered in the *F* class. The keystrokes belong to this class are backspaces.
4. IF: The keystrokes belongs to *IF* are those intermediate strokes which are in the IS while composing, but not present in the final text. However, backspaces are exempted from this class..

The definition of *corrected errors* [10] signifies any backspaced character as error even though the character is correct. Further, Mackenzie and Soukreff [11] point out this limitation and categorize backspaced character into two classes namely *corrected-but-right* and *corrected-and-wrong*.

5. IF_e $(Corrected - and - Wrong)$: This class presents the character(s) which are initially incorrect and later on corrected by user.
6. IF_c $(Corrected - but - Right)$: It contains character(s) which are initially correct and then deleted to correct other wrongly typed character(s).

Consider the following example for further clarification of above mentioned classes.

Target text : the_quick_brown

User input : $th\_qui\boldsymbol{x}<ck\_br\boldsymbol{p}own$

Transcribed text : $th\_quick\_br\boldsymbol{p}own$

This example contains three errors. An omitted 'e' which is the first vowel, an redundant 'x' which is later corrected by a backspace, a redundant 'p' which remains uncorrected. So, $C = 14$, $INF = 2$ (counting the extra 'p', and the missing 'e'), $IF = 1$ (deleted character 'x'), $F = 1$, $IF_e = 1$ ('x'), and $IF_c = 0$.

By definition, *INF* class represents the wrongly typed character(s) present in final text and *F* class indicates number of backspace(s); thus, these classes are also compatible for a system augmented with word prediction. However, other two classes i.e. *C* and *IF* are concerned with the performance of text entry system without word prediction. So, we modify these classes to judge the text entry performance in presence of a word prediction system. The classes are also compatible when prediction system automatically detects and corrects the mistakes made by the user.

We rename C class as C_{total} and divide it into two classes, C_{user} and C_{system}.

1. C_{user} : This includes total number of characters correctly entered by user during text composition. For example, T_1 is "$quick\_$", and T_{log} contains text "$\boldsymbol{q}a < \boldsymbol{u}\boldsymbol{x}\boldsymbol{c}(quick\_)$". Here, after entering "$\boldsymbol{q}a < \boldsymbol{u}\boldsymbol{x}\boldsymbol{c}$", user selects the word "$(quick\_)$" from prediction list. As a result, (T_2) becomes "$quick\_$". It may be noted that three characters ($\boldsymbol{q}$, $\boldsymbol{u}$ and $\boldsymbol{c}$) are correctly entered by the user, whereas, 'a' and 'x' are erroneous characters.

   Moreover, in any text entry system, the space character ('_') needs to be inserted by the user (one key press). Alternatively, in a word prediction system, space is automatically inserted after selecting a word from the prediction list. However, one key press is required to select the word from the prediction list. Hence, the value of $C_{user} = 3 + 1 = 4$.

2. C_{system} : This represents the total number of characters correctly entered by the word prediction system. In this case, character 'k' is correctly inserted and 'x' is corrected to 'i', hence $C_{system} = 2$.

3. C_{total} : This comprises of the total number of characters correctly entered by user and the word prediction system $[C_{total} = C_{user} + C_{system}]$.

We rename IF class as IF_{total} and divide it into two class as IF_{user} and IF_{system}. Further, IF_{user} class is split into IF_e and IF_c. Subsequently, IF_c class is subdivided into IF_r and IF_{nr}. The description of these classes are as follows.

1. $IF_e(Corrected - and - Wrong)$: This presents the character(s) which is initially incorrect and subsequently corrected by user. For example, T_1 is "$overall\_$" and T_{log} is "$ouera <<<< ve(overall\_)$", as a result T_2 is "$overall\_$". Here, user has entered the character 'u' which are initially wrong but later on corrected by user to 'v' and hence $IF_e = 1$.

2. $IF_r(Corrected - but - Right_{Required})$: This represents the character(s) which is initially correct but later on removed by user to correct some errors. However, these characters are required to populate the required word in prediction list. For example, T_1 is "$overall\_$", T_{log} is "$ouera <<<< ve(overall\_)$", and T_2 is "$overall\_$". After applying the effect of backspace in T_{log}, we get "$ove(overall\_)$". Here, the character 'e' is required which is initially deleted to correct the error. When it is entered again the required word is predicted and hence $IF_r = 1$.

3. $IF_{nr}(Corrected - but - Right_{NotRequired})$: This presents the character(s) which is initially correct and later on removed by user to correct the error. As the required word is predicted before these characters are typed, hence, it is not required to reenter them. Considering the previous example, the character "$uera$" is deleted to correct the error 'u'. Here, the character 'u' belongs to IF_e and character 'e' is an element of IF_r. However, the character sequence "ra" is deleted and not required to enter again because the correct word is already predicted by the system. Hence, those characters belong to IF_{nr} class. In this case $IF_{nr} = 2$.

4. $IF_c(Corrected - but - Right)$: This presents character(s) which is initially correct and then deleted to correct other wrongly typed character(s). These character(s) user may require to reenter in order to complete the target text $[IF_c = IF_r + IF_{nr}]$.

5. IF_{user} : This includes character(s) which is corrected by user during text composition. Therefore, it contains all character(s) belong to IF_e and IF_c.

6. IF_{system} : This presents character(s) which is corrected by system as user selects word from the prediction list. For example, in "$quxc(quick\_)$" 'x' is corrected to 'i' by the word prediction system, hence $IF_{system} = 1$.

7. $IF_{total}(Incorrect - and - fixed)$: This represents all character(s) which is corrected during text entry either by user or by the word prediction system $[IF_{total} = IF_{user} + IF_{system}]$.

Proposed Efficiency Measure Metrics

Various measures exist to quantify errors committed during text composition. In this section, we describe existing performance measure metrics as well as extend the scope of some previously defined metrics. These metrics would be beneficial to analyze input stream for character level error in text composition system without prediction as well as text composition system augmented with simple and advanced word prediction facility. The detailed description about these metrics is given below.

1. KSPC : This measure is a simple ratio of the number of entered characters (including backspaces) to the final number of characters in the transcribed string.

$$KSPC = \frac{|T'_{log}|}{|T_2|} \tag{1}$$

   where T'_{log} is computed from T_{log} by converting predicted word to '$*$'.

2. Correction efficiency : Correction efficiency is the ratio of a number of characters corrected during text entry to the number of backspaces required to correct them. It is defined as:

$$\text{Correction efficiency} = \frac{IF_{total} + \delta}{F + \delta} \tag{2}$$

   The δ, used in this equation, is a constant factor which takes care of the divide by zero problems. This situation occurs when user composes the text without any backspace and all corrections are done by the word prediction system. The value of δ has been considered as 0.01 in our computation.

3. User conscientiousness : This metric compares the number of characters corrected by the user during text entry to the number of errors made, indicating how meticulous user is while correcting errors.

$$\text{User conscientiousness} = \frac{IF_{user}}{IF_{total} + INF} \tag{3}$$

4. Correct contribution : This metric is used to measure the percentage of correct contribution by the user over the total correctly entered text.

$$\text{Correct contribution} = \left(\frac{C_{user}}{C_{total}}\right) \times 100\% \quad (4)$$

5. Corrected saving : This is used to measure the percentage of correct contribution by the system in the total correctly entered text.

$$\text{Corrected saving} = \left(1 - \frac{C_{user}}{C_{total}}\right) \times 100\% \quad (5)$$

6. Total error rate : This metric represents percentage of total error occurred during text composition and calculated as:

$$\text{Total error rate} = \frac{IF_{total} + INF}{C_{total} + IF_{total} + INF} \times 100\% \quad (6)$$

7. Corrected error rate : This metric represents the percentage of errors corrected during composition of the desired text. This is calculated as:

$$\text{Corrected error rate} = \frac{IF_{total}}{C_{total} + IF_{total} + INF} \times 100\% \quad (7)$$

8. Uncorrected error rate : This represents the percentage of uncorrected errors left out in final transcribed text and can be calculated as:

$$\text{Uncorrected error rate} = \frac{INF}{C_{total} + IF_{total} + INF} \times 100\% \quad (8)$$

9. Corrected by user error rate : This metric represent the percentage of error corrected by user.

$$\text{Corrected by user error rate} = \frac{IF_{user}}{C_{total} + IF_{total} + INF} \times 100\% \quad (9)$$

10. Corrected by system error rate : This metric represents the percentage of error corrected by the system.

$$\text{Corrected by system error rate} = \frac{IF_{system}}{C_{total} + IF_{total} + INF} \times 100\% \quad (10)$$

11. Corrected and wrong error rate : This indicates the fixed keystrokes that are initially erroneous.

$$\text{Corrected and wrong error rate} = \frac{IF_e}{C_{total} + IF_{total} + INF} \times 100\% \quad (11)$$

12. Corrected but right error rate : This metric contains the fixed keystrokes that are initially correct.

$$\text{Corrected but right error rate} = \frac{IF_c}{C_{total} + IF_{total} + INF} \times 100\% \quad (12)$$

13. Corrected but right and required error rate : This metric represents the percentage of character belong to the IF_r during composition of the desired text.

$$\text{Corrected but right and required error rate} = \frac{IF_r}{C_{total} + IF_{total} + INF} \times 100\% \quad (13)$$

14. Corrected but right and not required error rate : This metric represents the percentage of characters belong to the IF_{nr} class during composition of the desired text. This contains the characters which are correctly typed but not required and deleted by user.

$$\text{Corrected but right and not required error rate} = \frac{IF_{nr}}{C_{total} + IF_{total} + INF} \times 100\% \quad (14)$$

Computing Error Metrics

The basic steps to compute different error classes are shown in Fig. 1 and described as follows. We take the input as T_1 and T_{log}. For each set of backspace in T_{log}, we identify and store overwritten data (if exists) in $CDDTable$. We also remove this overwritten data from T_{log} and convert all the set of backspace into *box notation*. Next, we utilize $CDDTable$ and predict word for the generation of string CN_1. We create $IndexTable$ using this CN_1 and $iLog$. For each entry in $IndexTable$, we compute α, β from CN_1 and ψ from $iLog$. These data are then stored into $DataTable$ for further analysis. Next, we utilize the created $DataTable$ for computation of IF_e, IF_c, IF_r and IF_{nr}. Subsequently, we also compute CN_2 and T_2 from T_{log}. Finally, all other error classes like F, IF_{total}, INF, C_{user}, C_{system}, C_{total}, IF_{user}, IF_{system} are computed using CN_1, CN_2, T_2 and $iLog$, respectively.

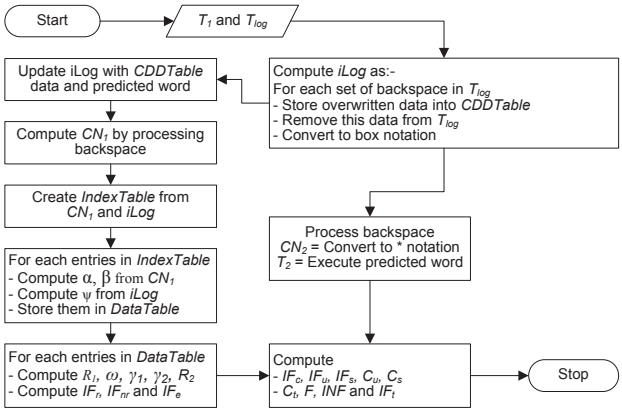


Figure 1. Flowchart of the proposed error evaluating methodology

Steps Involved in Processing of Log File

A log file is maintained when user composes the text using word prediction system. The log file can further be analyzed to compute various performance measuring metrics. Here, T_1, T_{log} and T_2 represent the target text to be composed, content of log file and final transcribed text, respectively.

In an advanced word prediction system, committed errors can be corrected in two ways, using backspace or selecting the desired word from prediction list. Suppose, the target text T_1 is "$quick\_$" which consists of 6 characters including space (see Table 2). To compose T_1, user commits spelling error by inserting character *'a'* at the place of *'u'* and after that, uses backspace '<' to correct it. Further, in the same string, user enters "***uxc***" rather than "***uic***". Still there exist a spelling error (*'x'* instead of *'i'*) in composed word which is not corrected by user. The log file generated by word prediction system contains the information about all the actions taken by the user and represented as T_{log} which is "$qa < uxc(quick\_)$", shown in Table 2. Note that, when the desired word is selected from the prediction list, a space is automatically added to the content. Based on different sequences of typed text, backspace and prediction, some extra processing can be performed on T_{log} (explained in the next section). The sequence of backspace(s) is converted into *box notation* for further computation. The processed log data is represented as $iLog$.

Table 2. Example of text composition with advanced word prediction

Terms	Description
T_1	$quick\_$
T_{log}	$qa<uxc(quick\_)$
$iLog$	$qa[01]uxc(quick\_)$
CN_1	$qa[01]uxc*$
CN_2	$quxc*$
T_2	$quick\_$

The $iLog$ is further modified by converting the predicted word, the string within bracket "(...)", to '*'. The modified data is stored in compact notation (represented as CN_1 in Table 2). CN_1 may contain misspelled characters and corresponding backspace by user. Another compact notation (represented as CN_2 in Table 2) stores the data after applying the effect of backspace on CN_1. Finally, by applying the effect of backspace on $iLog$ and replacing user typed sequences with the predicted one, we get the composed text stored as T_2.

Computation of $iLog$

T_{log} contains user log data, that is, all of the typed characters, spacebar, backspaces etc. as well as all selected words from the prediction. Thus, some of these data get modified or overwritten multiple times and not appeared in the final transcribed text. These overwritten data needs to be removed from T_{log} in order to identify final text. $iLog$ contains data from T_{log} after removal of overwritten data.

To calculate $iLog$, we first initialize $iLog$ with the value of T_{log}. Then for each set of backspace B in $iLog$, we count the occurrence of backspace, (i.e. '$<$' in B) and convert the continuous backspaces in *box notation*. Let, T_1 is "$the\_quick\_brown\_$" and T_{log} is "$t(the\_)qw < ui(quiet\_) <<< (quick\_)br(brown\_)$". Thus initial value of $iLog$ is "$t(the\_)qw<ui(quiet\_)<<<(quick\_)br(brown\_)$" and after converting set of backspaces into *box notation*, $iLog$ becomes "$t(the\_)qw[01]ui(quiet\_)[03](quick\_)br(brown\_)$". Now, if the backspace set B does not modify any word which is entered using the prediction system, then computation can be continued for next backspace set. Otherwise, we identify the predicted word and also user's input sequence which results that predicted word. In this example, the second backspace set modifies a word entered using prediction. Here, the predicted word is "$quick\_$" and user input sequence which results "$quick\_$" is "$qw[01]ui$". Presence of backspace in user input indicates that it contains multiple corrected data. So, we store the starting position of user input in $iLog$, user input sequence and predicted word in $CDDTable$. Now we replace user input sequence by predicted word in $iLog$. For this example, $CDDTable$ contains 7, $qw[01]ui$, $quiet\_$ and after replacing "$qw[01]ui$" by "$quick\_$", value of $iLog$ becomes "$t(the\_)quiet\_[03](quick\_)br(brown\_)$". The stored data in $CDDTable$ also contributes in computation of error classes related to user, that is, IF_{user}. Thus, after calculating CN_2 and T_2 form $iLog$, we update $iLog$ by adding $CDDTable$ data with it. The value of CN_1 is calculated from this updated $iLog$. After adding $CDDTable$ data, final value of $iLog$ is "$t(the\_)qw[01]ui(quiet)quiet\_[03](quick\_)br(brown\_)$".

Creation of $IndexTable$

While typing, users can commit errors which may or may not be corrected in the final transcribed string. The corrections are made with the help of backspace ($<$). The occurrence of backspaces along with their starting index are calculated and stored into $IndexTable$. We illustrate the computation of $IndexTable$ with a suitable example below.

Suppose, T_1 is "$the\_quick\_brown\_$" and T_{log} is "$thw\_qui <<<<< e\_qu(quick\_)bxow <<< r(brown\_)$" (see Table 3). As a result, $iLog$ contains data as "$thw\_qui[05]e\_qu(quick\_)bxow[03]r(brown\_)$" and CN_1 contains data as "$thw\_qui[05]e\_qu*bxow[03]r*$". Now the index of each '[' in CN_1 are recorded as k and the corresponding value between '[' and ']' are stored as v. We store each k, v pair present in CN_1 into $IndexTable$. In addition to this, we also store the index of '[' in $iLog$ which is represented as k_{log} (see Table 4).

Table 3. Example 1

Terms	Description
T_1	"$the\_quick\_brown\_$"
T_{log}	"$thw\_qui<<<<<e\_qu(quick\_)bxow<<<r(brown\_)$"
$iLog$	"$thw\_qui[05]e\_qu(quick\_)bxow[03]r(brown\_)$"
CN_1	"$thw\_qui[05]e\_qu*bxow[03]r*$"
CN_2	"$the\_qui*br*$"
T_2	"$the\_quick\_brown\_$"

Table 4. $IndexTable$

k	k_{log}	v
7	7	5
20	27	3

Computation of α

α represents a sequence of characters which are removed by user in order to correct error(s) from text under composition. It is computed from CN_1 and $IndexTable$. The steps involved in computation of α are described in Algorithm 1. We extract first k^{th} characters from CN_1 and store it in δ. Then, we process backspace, if any, in δ. Now, α is the last v character(s) of δ, where v represents the number of consecutive backspaces for given k.

Input: CN_1, (k, v) pair from $IndexTable$
Output: α for given (k, v)
1 $\delta \leftarrow$ copy first k^{th} characters from CN_1
2 $\delta \leftarrow$ process backspace in δ
3 $\alpha \leftarrow$ copy last v characters of δ
4 **return** α

Algorithm 1: Computation of α

Let T_1 is "$the\_quick\_$" and T_{log} is "$thw\_qux < i <<<<<< e\_qui(quick\_)$". As a result, CN_1 contains "$thw\_qux[01]i[05]e\_qui*$" and (k, v) pair in $IndexTable$ are $(7, 1)$ and $(12, 5)$. Now if we calculate α for 2^{nd} entry in $IndexTable$,

then k is 12 and v is 5. So, δ is "thw_qui" and after applying the effect of backspace α results "w_qui".

Computation of β

β represents the sequence of characters which are entered by user after removing the error(s) from the composed text. It is computed from CN_1. Algorithm 2 represents the steps involved to compute β. In this algorithm, β_{start} and β_{end} represent starting and ending positions of β in CN_1, respectively. Let C represents the number of words in α which is required to identify the proper position of β_{end}. The array $content$ represents the characters which identify the end of individual β. First, we initialize β_{end} with β_{start}. Then, we search for the position of any character of $content$ array in CN_1 starting from β_{end} and store the new position in β_{end}. This search is executed for C times. Finally, β is computed by copying characters form β_{start} to β_{end} from CN_1.

Input: CN_1, k and C as word count of α
Output: β for given k

$\beta_{start} \leftarrow \beta_{end} \leftarrow k+4$
$content[\,] \leftarrow \{$ '*', '_', '[' $\}$
for $i \leftarrow 1$ **to** C **do**
    $\beta_{end} \leftarrow$ `IndexOfAny(`CN_1,*content*,β_{end}`)`
    if $\beta_{end} = -1$ **then** $\beta_{end} \leftarrow$ `Length(`CN_1`)`

    else if $CN_1[\beta_{end}] \neq$ *'['* **then** β_{end}++

$\beta \leftarrow$ copy characters between β_{start} to β_{end} of CN_1
return β

Algorithm 2: Computation of β

Considering the example shown in Table 5, when k is 12, α is "w_qui", hence C becomes 2. So, β_{start} is 16 and initially β_{end} is also 16. Finally, β_{end} is 21, resulting β as "e_qui*".

Table 5. Example 2

Terms	Description
T_1	"$the\_quick\_$"
T_{log}	"$thw\_qux<i<<<<<e\_qui\ (quick\_)$"
$iLog$	"$thw\_qux[01]i[05]e\_qu(quick\_)$"
CN_1	"$thw\_qux[01]i[05]e\_qui*$"
CN_2	"the_qui*"
T_2	"$the\_quick\_$"

Computation of ψ

ψ represents the word(s) which is visible after user input or modification. It is computed from $iLog$ and $indexTable$. The procedures to compute ψ are stated in Algorithm 3. In this algorithm, ψ_{start} and ψ_{end} represent starting and ending positions of ψ in $iLog$, respectively. There may be prediction or multiple backspaces while composing the current word before k_{log} position. So ψ_{start} need to be calculated accordingly. Let C represents the number of words in α, which is required to identify the proper position of ψ_{end}. Now, we compute Δ by applying the effect of backspace on characters between ψ_{start} and ψ_{end} from $iLog$. The typed characters present in Δ are then replaced with the predicted word and stored in ψ.

Input: $iLog$, k_{log}, v and C as word count of α
Output: ψ for given k_{log}

$\mu \leftarrow$ copy up to $(k_{log}-v)^{th}$ characters from $iLog$
$\psi_{start} \leftarrow$ `LastIndexOf(`μ,'_'`)` $+1$
while $iLog[\psi_{start}] =$ *'['* **do**
    $\mu \leftarrow$ copy up to $(\psi_{start}-2)^{th}$ characters from $iLog$
    $\psi_{start} \leftarrow$ `LastIndexOf(`μ,'_'`)` $+1$
if $iLog[\psi_{start}]=$*')'* **then** ψ_{start}++

$\psi_{end} \leftarrow k_{log}+4$
for $i \leftarrow 1$ **to** C **do**
    $\psi_{end} \leftarrow$ `IndexOf(`$iLog$,'_',ψ_{end}`)` $+1$
    if $\psi_{end} = 0$ **then** $\psi_{end} \leftarrow$ `Length(`*iLog*`)`

$\mu \leftarrow$ copy characters between ψ_{start} and ψ_{end} of $iLog$
$\Delta \leftarrow$ process backspace on μ
$\psi \leftarrow$ replace typed character(s) with predicted word, if any, on Δ
return ψ

Algorithm 3: Computation of ψ

Considering the example shown in Table 5, for (k_{log}, v) pair as $(7, 5)$, α becomes "w_qui" and as a result C is 2. So, ψ_{start} is calculated as 0 and ψ_{end} is initialized with 11. Then ψ_{end} is changed to 13 and finally it contains 22. Thus, the character sequence between ψ_{start} and ψ_{stop} is "$thw\_qui[05]\ e\_qu(quick\_)$". After applying the effect of backspace on that sequence, Δ is "$the\_qu(quick\_)$". Next typed characters "qu" will be replaced by the predicted word "$quick\_$" resulting ψ as "$the\_quick\_$".

Edit distance

Given two strings A and B, the score *edit distance* is defined as the minimum number of edit operations needed to transform B into A or vice-versa, with the allowed edit operations being insertion, deletion, or substitution of a single character [2]. For example, if A is "$quickly$" and B is "$qxciklly$", then edit distance between A and B is 4.

Maximum match

We have used the following technique to find the maximum match between two strings.

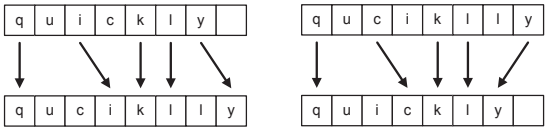
LCS: Given two strings A and B, the term *LCS* is defined as the longest subsequence common to both A and B [2]. For example, A is "$quickly$" and B is "$qxciklly$". The computation of LCS between A and B is shown in Table 6. The actual subsequences are deduced in a "$traceback$" procedure that follows the arrows backwards, starting from the last cell in the table. When the length decreases, the sequences must have a common element. Several paths are possible when two arrows are shown in a cell (see Table 6). Figure 2(a) indicates the subsequence for A and B which is "$qikly$". In other words, $LCS(A, B)$ is "$qikly$". As the length of longest common subsequence is five so it has 5 similar characters (the last cell shown in Table 6). Similarly, the longest common subsequence between "$qxciklly$" and "$quickly$" is "$qckly$" (i.e. $LCS(B, A)$ is "$qckly$", shown in Fig 2(b)).

Identification of individual α, β and ψ

Suppose, user enters a sequence of texts and realizes that some errors are prevalent in the previously entered text sequence. This error can occur in the current word or in any

Table 6. Computation of the longest common subsequence (traceback are shown with arrow)

ϕ	ϕ	q	x	c	l	k	l	l	y
ϕ	0	0	0	0	0	0	0	0	0
q	0	↖1	←1	1	1	1	1	1	1
u	0	↑1	←↑1	←1	1	1	1	1	1
i	0	↑1	←↑1	1	↖2	2	2	2	2
c	0	1	1	↖2	←↑2	2	2	2	2
k	0	1	1	2	2	↖3	3	3	3
l	0	1	1	2	2	3	↖4	←4	4
y	0	1	1	2	2	3	4	4	↖5



(a) LCS(A, B) (b) LCS(B, A)

Figure 2. The longest common subsequence

previous word(s). In order to correct an error, user selects the backspace to remove character(s) up to the error position and enters some new character(s). A single word can contain multiple errors in different positions. Thus, user may choose both backspace and prediction multiple times to compose a word. Sometimes, user also needs to modify the predicted word after selecting a wrong word from the prediction list.

In order to handle these situations, we need special attention for computing α, β and ψ. As α may contain more than one word, proper position is need to be identified in CN_1 to get β (steps $3-5$ in Algorithm 4). Next step is to process the text and identify whether some word(s) is missing between α and β and finalize the value accordingly by deciding whether α, β and ψ will be split or not.

In Table 7, α is "$is\_word\_$" which contains two spaces. Moreover, the last character in α is also ('_') hence, the value of $count$ is 2 and β becomes "$wo**$" (see CN_1 and steps $3-5$ in Algorithm 4). Similarly, value of ψ is "$word\_is\_$". Next we process the text to decide whether α, β and ψ will be split or not. We concatenate $LCS($"$is\_$", "$wo*$"$)$ and $LCS($"$word\_$", "$*$"$)$ which will be null (represented as Str, steps $10-23$ in Algorithm 4). Whereas, $LCS(\alpha, \beta)$ becomes "wo"; as $LCS(\alpha, \beta)$ is not equal with Str hence, the value of α, β and ψ will remain same (i.e. do not split, steps $38-44$ in Algorithm 4). This process stores individual α, β and ψ in $DataTable$.

Table 7. Example 3

Terms	Description
T_1	"$this\_word\_is\_peculiar\_$"
T_{log}	"$th(this\_)is\_word\_<<<<<<<<<wo(word\_)(is\_)p(peculiar\_)$"
$iLog$	"$th(this\_)is\_word\_[08]wo(word\_)(is\_)p(peculiar\_)$"
CN_1	"$th*is\_word\_[08]wo**p*$"
CN_2	"$th*wo**p*$"
T_2	"$this\_word\_is\_peculiar\_$"

Computation of error class: IF_{user}

Once the data table is initialized, for each entry in $DataTable$, we retrieve value for α, β and ψ (steps $4-6$ in Algorithm 5) and compute the character belong to IF_r class, also represented as R_1. This can be computed by taking $LCS(\alpha, \beta)$ (step 7 in Algorithm 5). There can be two options: R_1 does not contain any character or it has some characters. If $R_1 \neq null$, we extract the last character present in R_1 and represent it as ω. We then count the occurrences of ω in R_1, let it be represented by $count_1$, and find the position of $count_1{}^{th}$ occurrences of ω in α, let it be at position j. Next, we extract the $(j+1)^{th}$ character onward from α and store them in γ_1. Similarly, we compute γ_2 from ψ (steps

Input: $IndexTable$, CN_1 and $iLog$
Output: α, β and ψ in $DataTable$
1 **foreach** *entry in* $IndexTable$ **do**
2 $\alpha \leftarrow$ compute α with CN_1, k and v
3 $count \leftarrow$ count the occurrences of '_' in α
4 **if** *last character in* $\alpha \neq$ '_' **then** $count$++
5
6 $\beta \leftarrow$ compute β with CN_1, k and $count$
7 $\psi \leftarrow$ compute ψ with $iLog, k_{log}, v$ and $count$
8 ▷ initialize: $\alpha_1 \leftarrow \alpha$, $\beta_1 \leftarrow \beta$, $Str \leftarrow$ NIL, $ind \leftarrow temp \leftarrow 0$
9 $Content[\,] \leftarrow$ {'*', '_'}
10 ▷ $Store[\,,\,]$ is a $count \times 2$ dimensional array
11 **for** $i \leftarrow 0$ **to** $count - 1$ **do**
12 ▷ identify input sequence for individual word from α_1
13 $temp \leftarrow \alpha_1$.IndexOf('_')
14 **if** $temp \neq -1$ **then** $ind \leftarrow temp + 1$
15
16 **else** $ind \leftarrow$ length[α]
17
18 $Store[i, 0] \leftarrow$ copy characters from 0 to ind of α_1
19 $\alpha_1 \leftarrow$ remove character from 0 to ind of α_1
20 ▷ identify input sequence for individual word from β_1
21 $temp \leftarrow \beta_1$.IndexOfAny($content$)
22 **if** $temp \neq -1$ **then** $ind \leftarrow temp + 1$
23
24 **else** $ind \leftarrow$ length[β]
25
26 $Store[i, 1] \leftarrow$ copy characters from 0 to ind of β_1
27 $\beta_1 \leftarrow$ remove character from 0 to ind of β_1
28 $Str \leftarrow Str+$ LCS($Store[i, 0], Store[i, 1]$)
29 **if** $LCS(\alpha, \beta) = Str$ **then**
30 ▷ Split α, β and ψ when condition is true
31 **for** $i \leftarrow 0$ **to** $count - 1$ **do**
32 dT$[nI + i, 0] \leftarrow k$
33 dT$[nI + i, 1] \leftarrow v$
34 dT$[nI + i, 2] \leftarrow Store[i, 0]$ ▷ value for α
35 dT$[nI + i, 3] \leftarrow Store[i, 1]$ ▷ value for β
36 $temp \leftarrow \psi$.IndexOf('_')
37 **if** $temp \neq -1$ **then** $ind \leftarrow temp + 1$
38
39 **else** $ind \leftarrow$ length[ψ]
40
41 dT$[nI + i, 4] \leftarrow$ copy characters from 0 to ind of ψ
42 $\psi \leftarrow$ remove character from 0 to ind of ψ
43 $nI \leftarrow nI + count$
44 **else**
45 ▷ α, β and ψ will not be splitted
46 dT$[nI, 0] \leftarrow k$
47 dT$[nI, 1] \leftarrow v$
48 dT$[nI, 2] \leftarrow \alpha$
49 dT$[nI, 3] \leftarrow \beta$
50 dT$[nI, 4] \leftarrow \psi$
51 nI++

Algorithm 4: Identification of individual α, β and ψ

$9 - 19$ in Algorithm 5). On the other hand, when R_1 is $null$, we assign γ_1, γ_2 as α and ψ, respectively (steps $21 - 22$ in Algorithm 5).

Input: $DataTable$
Output: Value of error class
1 **for** $i \leftarrow 0$ **to** *nI* **do**
2 ▷ initialize: $R \leftarrow err \leftarrow R_1 \leftarrow R_2 \leftarrow \omega \leftarrow \gamma_1 \leftarrow \gamma_2 \leftarrow$ NIL
3 ▷ initialize: $count_1 \leftarrow count_2 \leftarrow count_3 \leftarrow nI \leftarrow 0$
4 $\alpha \leftarrow$ dT$[i, 2]$ ▷ Retrieve α
5 $\beta \leftarrow$ dT$[i, 3]$ ▷ Retrieve β
6 $\psi \leftarrow$ dT$[i, 4]$ ▷ Retrieve ψ
7 $R_1 \leftarrow$ LCS(α , β) ▷ characters $\in IF_r$
8 **if** *length* $[R_1] > 0$ **then**
9 dT$[i, 5] \leftarrow R_1$ ▷ Store IF_r characters
10 dT$[i, 6] \leftarrow \omega \leftarrow$ last character in R_1
11 **for** $j \leftarrow 0$ **to** *length*$[R_1]$ **do**
12 **if** $R_1[j] = \omega$ **then** $count_1$++ ▷ count the occurrence of $\omega \in R_1$
13
14 dT$[i, 7] \leftarrow count_1$
15 **for** $j \leftarrow 0$ **to** *length*$[\alpha]$ **do**
16 **if** $\alpha[j] = \omega$ **then**
17 $count_2$++ ▷ count the occurrence of $\omega \in \alpha$
18 **if** $count_1 = count_2$ **then**
19 dT$[i, 8] \leftarrow \gamma_1 \leftarrow$ copy $(j + 1)^{th}$ characters onward from α ▷ Compute γ_1
20 **for** $j \leftarrow 0$ **to** *length*$[\psi]$ **do**
21 **if** $\psi[j] = \omega$ **then**
22 $count_3$++ ▷ count the occurrence of $\omega \in \psi$
23 **if** $count_1 = count_3$ **then**
24 dT$[i, 9] \leftarrow \gamma_2 \leftarrow$ copy $(j + 1)^{th}$ characters onward from ψ ▷ Compute γ_2
25 **else**
26 dT$[i, 8] \leftarrow \gamma_1 \leftarrow \alpha$ ▷ Compute γ_1
27 dT$[i, 9] \leftarrow \gamma_2 \leftarrow \psi$ ▷ Compute γ_2
28 dT$[i, 10] \leftarrow R_2 \leftarrow$ LCS(γ_1, γ_2) ▷ characters $\in IF_{nr}$
29 dT$[i, 11] \leftarrow$ length$[\alpha]$
30 dT$[i, 12] \leftarrow IF_r \leftarrow$ length$[R_1]$
31 dT$[i, 13] \leftarrow IF_{nr} \leftarrow$ length$[R_2]$
32 dT$[i, 14] \leftarrow IF_e \leftarrow$ dT$[i, 11]-$ dT$[i, 12]-$ dT$[i, 13]$
33 $R \leftarrow R_1 + R_2$
34 $err \leftarrow \alpha$
35 **for** $j \leftarrow$ *length*$[R] - 1$ **to** 0 **do**
36 $\phi \leftarrow err$.LastIndexOf($R[j]$)
37 $err \leftarrow$ Remove the characters from err at position ϕ
38 dT$[i, 15] \leftarrow err$ ▷ characters $\in IF_e$
39 **for** $j \leftarrow 12$ **to** 15 **do**
40 **for** $i \leftarrow 0$ **to** nI **do**
41 dT$[nI, j] \leftarrow$ dT$[nI, j]+$ dT$[i, j]$

Algorithm 5: Calculation of IF_{user} class

After computation of γ_1 and γ_2, we compute characters belong to IF_{nr} and IF_e (steps $23 - 32$ in Algorithm 5). Character(s) belong to IF_{nr} is computed by taking $LCS(\gamma_1, \gamma_2)$. Whereas, IF_e contains those characters which are present in α but neither part of IF_r nor IF_{nr}. In order to compute the value of IF_e, we merge the content of IF_r and IF_{nr}, and store the result as R. For each character present in R from right to left, we scan err from right to left and remove that character from err (steps $28 - 32$ in Algorithm 5). Finally, the characters left out in err are the characters of IF_e. The last step is to compute the total IF_r, IF_{nr} and IF_e. It can be computed by taking the sum of all IF_r, IF_{nr} and IF_e present in $DataTable$ (steps $33 - 36$ in Algorithm 5).

Referring to the previous example, α is "$is\_word\_$", β is "$word**$" and ψ is "$word\_is\_$". The character belong to IF_r class (R_1) can be computed as $LCS($"$is\_word\_''$, "$word * *''$) which is "wo". Hence, the last character present in R_1 (ω) is "o". Next, we find the proper position of ω in α, β and ψ and compute γ_1, γ_2. In this case, γ_1 = "$rd\_$" and γ_2 is "$rd\_is\_$". Character belong to IF_{nr} (R_2) is computed as $LCS(\gamma_1, \gamma_2)$ which is "$rd\_$". R, computed by concatenating R_1 and R_2, contains "$word\_$". Hence, IF_e contains "$is\_$". Finally, IF_r, IF_{nr} and IF_e hold the values as 2, 3 and 3, respectively, shown in Fig. 3

Detail description of incorrect but fixed by user [IF(user)]

Alpha	Beta	Word	R1	W	wCount	Gamma1	Gamma2	R2	lAlpha	IFr	IFnr	IFe	R3
is_word_	wo**	word_is_	wo	o	1	rd_	rd_is_	rd_	8	2	3	3	is_
										2	3	3	

Figure 3. Computation of IF_{user}

Computation of other error classes
The computation of F, INF, IF_{system}, IF_{total}, C_{user}, C_{system} and C_{total} are described in this section. Let Cnt represents the count of '$*$' in CN_1. Initially, we count the total occurrence of backspaces presented in T_{log} and store the value in F which is also interpreted as the essence of number of errors corrected by user by means of backspace. The number of characters correctly entered by user (i.e. C_{user}) can be computed by taking the maximum match between T_1 and CN_2 then adding it with Cnt.

$$C_{user} = LCS(T_1, CN_2) + Cnt \tag{15}$$

To compute C_{system}, we convert the occurrence of '$*$' with '_' in the string CN_2 and take maximum match between T_2 and the converted data. Let the result of maximum match be δ. We identify the number of character(s) present in T_2 but absent in δ. This represents the total number of characters present in C_{system}. As a result, C_{total} can be computed by summing up C_{user} and C_{system}. The value of INF can be computed as *Edit Distance* between T_1 and T_2.

$$INF = Edit\ Distance(T_1, T_2)$$

The steps involved in computation of IF_{system} are as follows. For each predicted word in T_{log}, we identify corresponding user input sequence and apply the effect of backspace, if any. Then,maximum match between user input and predicted word is calculated. We subtract the result of maximum match from user input which gives the character corrected by the system.

The value of incorrect and fixed by user (IF_{user}) can be obtained by summing the IF_e and IF_c. Here IF_c is a summation of IF_r and IF_{nr}. Finally, the IF_{total} can be measured by summing up IF_{user} and IF_{system}. Figure 4 represents the computation of F, INF, C_{user}, C_{system}, C_{total}, IF_{user}, IF_{system}, IF_{total}, IF_e, IF_c, IF_r and IF_{nr} for the example shown in Table 7.

ANALYSIS OF THE PROPOSED METRICS

The proposed error evaluating metrics are applicable in a wide variety of text entry systems which include simple

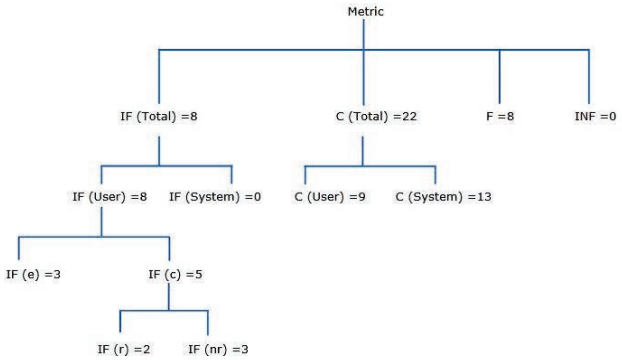


Figure 4. Classification of Errors

character by character text transcription system, word prediction system with and without word correction facility etc. The quantitative evaluation for those systems is described in this section. Suppose, the target text T_1 is "$the\_quick\_brown\_fox\_jump\_over\_the\_wall\_$". To compose this sequence of characters, different system performances are being calculated.

Text entry system without prediction: To compose the target text (T_1), suppose T_{log} is "$the\_quick\_b\boldsymbol{e} < rown\_fo\boldsymbol{y}\_j\boldsymbol{e} < ump\_o\boldsymbol{uer} <<< ver\_the\_wall\_$". Therefore, T_2 is "$the\_quick\_brown\_fo\boldsymbol{y}\_jump\_over\_the\_wall\_$". This text contains only one spelling variation i.e. 'y' in "$fo\boldsymbol{y}$" (instead 'x' for "fox"). The detail description is shown in Table 8.

Simple word prediction system: For the word prediction system which does not correct the spelling error, the calculation of different error metrics is described below. Suppose, T_1 is again "$the\_quick\_brown\_fox\_jump\_over\_the\_wall\_$" and user composes "$th(the\_)qu(quick\_)b\boldsymbol{e} < ro(brown\_)fo\boldsymbol{y}\_j\boldsymbol{e} < u(jump\_)o\boldsymbol{uer} <<< ve(over\_)t(the\_)w(wall\_)$". Here, spacebar is automatically inserted after the selection of word from prediction list. Note that for the word "$over$", user follows the sequence "$o\boldsymbol{uer} <<< ve(over\_)$". In other word, user misspells the character 'u' for 'v' and types "$ouer$". Then user erases the 3 characters ("uer") using the backspace ('$<$') and types the characters "ve". Finally, the word "$over$" appears in the prediction list and it is selected by the user. Therefore, we can easily observe that the task requires 10 keypress (o***uer***<<<ve*) by the user including the selection of word from the prediction window. The detail description is shown in Table 8.

Advanced word prediction system: To compute the different error metric, we consider the target text as "$the\_quick\_brown\_fox\_jump\_over\_the\_wall\_$" and user composed text as "$th(the\_)qu(quick\_)b\boldsymbol{e}ro(brown\_)fo\boldsymbol{y}\_j\boldsymbol{e} < u(jump\_)o\boldsymbol{u}e(over\_)t(the\_)w(wall\_)$". Note that for the word "$over$", user composes the sequence as "$o\boldsymbol{u}e(over\_)$". We can easily observe that it requires 4 keypress by the user including the selection of the word from the prediction window. As the proposed system detects and corrects the spelling error, the number of key press required gets reduced.

The calculation of different error classes and performance evaluating metrics for above mentioned three examples on system without prediction, with simple and advanced prediction are represented in Table 8(b) and 8(c), respectively. In future, in case of quantifying error classes and error metric for different examples, we apply formulae from existing literature as well as proposed in this paper.

Table 8. Efficiency measure for without prediction, simple prediction and advanced prediction

(a) Target text, user log data and final text

Target text (T_1)		$the\_quick\_brown\_fox\_jump\_over\_the\_wall\_$
Log data (T_{log})	Without prediction	$the\_quick\_be < rown\_foy\_je < ump\_ ouer <<< ver\_the\_wall\_$
	Simple prediction	$th(the\_)qu(quick\_)b\boldsymbol{e} < ro(brown\_)fo\boldsymbol{y}\_j\boldsymbol{e} < u(jump\_)o\boldsymbol{uer} <<< ve(over\_)t(the\_)w(wall\_)$
	Advanced prediction	$th(the\_)qu(quick\_)b\boldsymbol{e}ro(brown\_)fo\boldsymbol{y}\_j\boldsymbol{e} < u(jump\_)o\boldsymbol{u}e(over\_)t(the\_)w(wall\_)$
Final text (T_2)		$the\_quick\_brown\_fo\boldsymbol{y}\_jump\_over\_the\_wall\_$

(b) Classification of errors

Efficiency measure	Without prediction	Simple prediction	Advanced prediction
C_{user}	38	24	23
C_{system}	0	14	15
C_{total}	38	38	38
F	5	5	1
INF	1	1	1
IF_{total}	5	5	3
IF_{user}	5	5	1
IF_{system}	0	0	2
IF_c	2	2	0
IF_e	3	3	1
IF_r	2	1	0
IF_{nr}	0	1	0

(c) Error metrics calculation

Efficiency measure	Without prediction	Simple prediction	Advanced prediction
KSPC	1.26	0.90	0.72
Correction efficiency	1.00	1.00	3.00
User conscientiousness	83.33	83.33	25.00
Correct contribution	100	63.16	60.53
Correct savings	0.00	36.84	39.47
Total error rate	13.64	13.64	9.52
Corrected error rate	11.36	11.36	7.14
Uncorrected error rate	2.27	2.27	2.38
Corrected by user error rate	11.36	11.36	2.38
Corrected by system error rate	0.00	0.00	4.76
Corrected and wrong error rate	6.82	6.82	2.38
Corrected but right error rate	4.55	4.55	0.00
Corrected but right and required error rate	4.55	2.27	0.00
Corrected but right and not required error rate	0.00	2.27	0.00

Software tool

To make this methodology easier to use, we make a tool available to the research community. It can be accessible from **`http://www.nid.iitkgp.ernet.in/Metric/`**. This software is written in C#.NET 3.5 with Microsoft Silverlight. This needs user to specify the target text and the content of log file for analysis. It shows the values of the existing and proposed metrics. We verify that the tool works for all three conditions, a) without prediction, b) prediction without error correction support and c) prediction with error correction support. As the service is developed to support unicode input, the usefulness of the service is not limited to any particular language.

CONCLUSION AND FUTURE WORK

Existing error classes and metrics are unable to quantify errors of text entry systems augmented with word prediction. This limitation is overcome in this work. The redefined and proposed error classes and error quantifying metrics, according to this work, is applicable to various text entry systems with and without word prediction. Based on the work, a software tool has been developed with which error correction efficiency can be evaluated automatically. This tool thus helps user interface designer to evaluate a text entry system and user performance. The proposed metrics further can be extended for text entry system where multiple keys are used to compose a single character (e.g. "Shift", "Ctrl" key).

REFERENCES

1. Boissiere, P., and Dours, D. VITIPI: Versatile Interpretation of Text Input by Persons with Impairments. In *Proceedings of the 5th International conference on Computers helping people with special needs. Part I*, R. Oldenbourg Verlag GmbH (1996), 165–172.
2. Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.
3. Fazly, A. The Use of Syntax in Word Completion Utilities. Master's thesis, Department of Computer Science, University of Toronto, 2002.
4. Kano, A., and Read, J. C. Text Input Error Categorisation: Solving Character Level Insertion Ambiguities using Zero Time Analysis. In *Proceedings of BCS-HCI*, British Computer Society (Swinton, UK, UK, 2009), 293–302.
5. MacKenzie, I. S. KSPC (Keystrokes per Character) as a Characteristic of Text Entry Techniques. In *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction (MobileHCI)*, Springer-Verlag (London, UK, 2002), 195–210.
6. MacKenzie, I. S., and Soukoreff, R. W. A Character-level Error Analysis Technique for Evaluating Text Entry Methods. In *Proceedings of the second Nordic conference on Human-computer interaction (NordiCHI)*, ACM (New York, NY, USA, 2002), 243–246.
7. MacKenzie, I. S., and Tanaka-Ishii, K. *Text Entry Systems: Mobility, Accessibility, Universality*. Morgan Kaufmann Inc., 2007.
8. Masui, T. POBox: An Efficient Text Input Method for Handheld and Ubiquitous Computers. In *Handheld and Ubiquitous Computing*, Springer (1999), 289–300.
9. Soukoreff, R. W., and MacKenzie, I. S. Measuring Errors in Text Entry Tasks: An Application of the Levenshtein String Distance Statistic. In *CHI Extended abstracts*, ACM (New York, NY, USA, 2001), 319–320.
10. Soukoreff, R. W., and MacKenzie, I. S. Metrics for Text Entry Research: an Evaluation of MSD and KSPC, and a New Unified Error Metric. In *Proceedings of CHI*, ACM (New York, NY, USA, 2003), 113–120.
11. Soukoreff, R. W., and MacKenzie, I. S. Recent Developments in Text-entry Error Rate Measurement. In *CHI '04 extended abstracts*, ACM (New York, NY, USA, 2004), 1425–1428.
12. Trnka, K., McCaw, J., Yarrington, D., McCoy, K. F., and Pennington, C. User Interaction with Word Prediction: The Effects of Prediction Quality. *ACM Transaction on Accessible Computing 1*, 3 (2009), 1–34.
13. Wobbrock, J., and Myers, B. Analyzing the Input Stream for Character-level Errors in Unconstrained Text Entry Evaluations. *ACM Transactions on Computer-Human Interaction 13*, 4 (2006), 458–489.