Automatic Whiteout: Discovery and Correction of Typographical Errors in Mobile Text Input

James Clawson, Alex Rudnick, Kent Lyons , Thad Starner
College of Computing and GVU Center
Georgia Institute of Technology
Atlanta, GA 30332
{jamer,alexr,kent,thad}@cc.gatech.ed

ABSTRACT

We detect and correct typing errors made on mini–QWERTY keyboards by analyzing features of the typing itself. Examining a database of mini–QWERTY typing data reveals that many errors made by typists are “off–by–one” errors. One likely cause of these errors is the relative size difference between the user’s thumb and the small, densely packed keys of the mini–QWERTY keyboard. Our goal with this work is to improve expert typing speeds and accuracy by automatically correcting the user’s typing errors before they are displayed on the screen. Using pattern recognition methods, we reduced the number of off–by–one errors by 39.78% and the total errors by 26.41%. This paper discusses the problem, the features used to detect errors, the techniques used to train the system, and future steps to generalize the algorithm for other keyboards and situations.

Keywords

Keyboard Input, Error Correction, Mobile Phones, Mini–QWERTY, Mobile Text Entry.

Categories and Subject Descriptors

H.5.2 [**:**]: User Interfaces, Input devices and strategies

1. INTRODUCTION

Miniature keyboards and miniature keypads are currently used extensively on mobile devices such as mobile phone handsets and personal digital assistants. The mini–QWERTY keyboard (Figure 1) is a common handheld mobile two–handed keyboard which contains at least one key for each letter plus a space bar and is configured in the same manner as a desktop QWERTY keyboard.

While the layout is analogous to desktop keyboards, mini-QWERTY keyboards contain keys that are densely packed to save space and are usually operated by a user’s two thumbs. Often the keys are smaller than the digit used to manipulate them (the thumb) resulting in difficulty of use. The user’s digit occludes visibility of the keys introducing ambiguity as to which key was actually pressed. Furthermore, Fitts’ Law, which describes the relationship between speed of movement, target size, and accuracy (adjusted target size) [8], implies that users will type less accurately as they type faster. Together these effects lead to typing errors where one digit may press multiple keys at once or a key adjacent to the intended key. These types of errors often occur, especially at rapid typing rates.

Figure 1: Commercial mobile phones with mini–QWERTY keyboards such as the Danger/T–Mobile Sidekick

In this paper we examine a set of mini–QWERTY keyboard text input data and identify a common type of error in the dataset that accounts for 61.28% of the total errors (off–by–one errors). We then use pattern recognition techniques to automatically recognize and correct these types of errors. We evaluate the effect of the correction on overall keystroke accuracy in an existing database and discuss how our algorithm can be employed to improve mobile text input on mini–QWERTY keyboards.

2. GENERATION OF THE DATA SET

We generated the data set used for our analysis from two longitudinal studies of mini–QWERTY keyboard use [1, 3]. In the first study we recruited 14 participants who had no prior experience with mini-QWERTY keyboard typing. They were randomly assigned to one of two subject groups, each using one of two different keyboard models. Subjects used the same keyboard throughout the experiment, which consisted of twenty 20–minute typing sessions.

The sessions involved subjects typing during several trial blocks; 10 phrases comprised each block. The phrases were taken from MacKenzie and Soukoreff’s set of 500 phrases designed for use in text entry studies [7]. The phrases use only lowercase letters and spaces with no punctuation. The canonical set was altered to use American English spellings. The test software prompts the user with the target phrase, displays the text produced by the user,

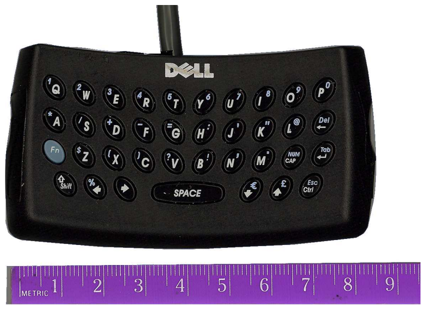

Figure 2: The Dell mini–QWERTY keyboard used in both studies.

records and displays the words per minute (wpm) and accuracy results for the previous sentence and the session as a whole.

Over the course of the study, the participants typed 33,945 phrases across all sessions, encompassing over 950,000 individual characters. Averaged over both keyboards, participants had a mean first session typing rate of 31.72 wpm. At the end of session twenty (400 minutes of typing) the participants had a mean typing rate of 60.03 wpm. The average accuracy rate for session one was 93.88% an gradually decreased to 91.68% by session twenty.

2.1 Sampling the Data Set

In Clarkson et al. [1] we state that the participants are considered expert typists by the time they begin the 16th typing session. As we are interested in improving the speed and accuracy of expert typing, we analyzed their data for the last five typing sessions for the Dell keyboard (Figure 2). We identified each character typed in a phrase as either correct or as an error. Upon encountering an error in the sentence, the remaining characters in the phrase (characters that occur after the error) were removed and are not included in the analysis. This truncation avoids difficulties in analyzing the user's editing behavior. More importantly, it avoids errors that may have cascaded due to an artifact of the data collection. Specifically, the test software employed to collect the data highlighted errors as users entered them, similar to the feedback provided by spell checking software. This highlighting potentially distracted the user, increasing her cognitive load and causing her to alter her natural behavior (for example, upon detecting the error, the user had to choose to correct the error or leave it uncorrected). Thus, all characters that occur in a phrase after the initial error are discarded. If the initial error occurs in the first two characters of the phrase the entire phrase is discarded. This subset of the data contains 4,480 phrases, 64,482 characters and 2988 errors.

3. ERROR ANALYSIS

Errors have long been considered an important source of insight into understanding a user's performance of a task. Grudin performed an analysis of error patterns for full–QWERTY desktop typing in an attempt to understand how complex motor task skills are organized and developed [5]. Like Grudin, we performed a similar analysis to discover the types of errors that occur [2]. Over the entire data set, we found that 40.2% of the users' errors were substitutions (replaced one letter with another), 33.2% were insertions (added an extra letter), 21.4% were deletions (left out a letter), and 5.2% were transpositions (exchanged the order of two letters). Upon further analysis, we encountered a different breakdown of errors that allowed us to classify the greatest number of errors in the set as "off–by–one" errors.

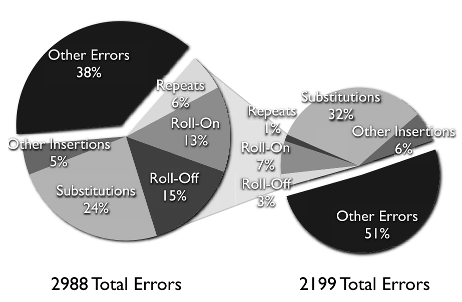

Figure 3: Breakdown of all expert typing errors before theoretically applying Automatic Whiteout to the data (left) and after (right).

3.1 Off–By–One Errors

Off–by–one errors consist of insertions (key repeats, roll–on, and roll-off errors) and substitutions of letters on the keyboard directly adjacent to the key the user intended to press. Accidental key repeats are insertions where the user unintentionally presses the same key twice (e.g. the user types"cat**t**" when she intended to type "cat"). Many of the remaining insertions result when the user presses an additional key either immediately to the left or to the right of the intended key, and 92% of these off–by–one insertions can be classified as either Roll–On and Roll-Off insertion errors. Roll–On insertions are where the inserted character comes before the intended character (e.g. the user types "ca**r**t" when she intended to type "cat"). Roll–Off insertions which occur when the inserted character comes after the intended character (e.g. the user types "cat**r**" when she intended to type "cat"). Finally off-by-one substitution errors occur when the intended character is replaced by the character immediately to the right or left of the intended character (e.g. the user types "ca**y**" or "ca**r**" when she intended to type "cat").

Our experimental data contains 2988 typing errors. Of those, 1825 (61.28%) are off–by–one. These errors occur more often than any other type of error in the mini-QWERTY keyboard typing data (see Figure 3 (left) for a breakdown of the off–by–one errors).

4. AUTOMATIC ERROR DETECTION AND CORRECTION

Having identified off–by–one errors as the most prevalent class of error in mini–QWERTY keyboard text input, we began to explore the possibility using pattern recognition techniques to automatically detect and correct these errors. For example, upon examining the data we observed that when a user creates a roll–on insertion error, the time between when the user pressed the incorrect key and when she pressed the correct key was significantly shorter than the time it normally takes for a user to correctly type the same two characters without error. In essence, the user accidently pressed two keys at the same time. The prevalence of off–by–one errors likely occurs because the target key and surrounding area is visually occluded by the user's thumb. This supposition is further supported by the implications of moving rapidly in a Fitts' Law task. Encouraged by our preliminary analysis, we decided to pursue the goal of leveraging characteristics of the users' input to detect and correct off–by–one errors.

4.1 Classifiers for Error Detection

Features	Description
dt	time difference from previous keystroke and the current
prevdt	time difference between previous two keystrokes
futdt	time difference until next keystroke
futnowdiff	difference between futdt and dt
nowprevdiff	difference between dt and prevdt
prob	Markov transition probability for previous three letters
logprob	log of prob
prob1	Markov transition probability for previous two letters
letterfreq	relative frequency of current letter, based on phrase set
neighborprob	Markov transition probability for previous two letters and current key's most likely neighbor
neighborprobdiff	neighborprob - prob
neighborfreq	relative frequency of most likely neighboring key, based on phrase set
neighborfreqdiff	neighborfreq - letterfreq
hdist	number of horizontal steps between previous key and current
vdist	number of vertical steps between previous key and current
hdistabs	absolute value of hdist
vdistabs	absolute value of vdist
ascii	ascii character code of current keystroke
allsame	Are the previous three keystrokes equal?
firstpair	Are the two keystrokes before the current one equal?
secondpair	Is the current keystroke equal to the previous?
firstandthird	Is the current keystroke equal to the keystroke two ago?
isletter	Is the current keystroke a letter (as opposed to whitespace)?
pisletter	Is the previous keystroke a letter?
ppisletter	Is the previous previous keystroke a letter?
dropprobdiff	Difference in Markov transition probability scores between the strings that would result in dropping the current keystroke and dropping the upcoming one.
dropprobdiffabs	Absolute value of dropprobdiff.
dropprobdiffsign	Sign of dropprobdiff.
dropprobdiff1	Like dropprobdiff, but with one-step Markov assumption.
probfuture	Markov transition probability of the previous, current, and upcoming keystrokes.
probfuturedropgain	The gain in probability score had by dropping the current keystroke, in light of the upcoming one
probfuture1	Like probfuture, but with one-step Markov assumption.
probfuturedropgain1	Like as probfuturedropgain, but with one-step Markov assumption.
futadjacent	Are the current and upcoming keystrokes adjacent?
next	ascii character code for upcoming keystroke
prevcuradjacent	Are the previous and current keystrokes adjacent?

Table 1: Features used in detecting off–by–one errors

To begin, we randomly assigned 10% of the phrases to independent test set and declared the remaining 90% to be the training set. We did not examine the independent test set until all features were selected and the tuning of the algorithm was complete.

From the training set we iteratively built a series of four training subsets, one for each classifier (roll–on, roll–off, repeats, and substitutions). The training subsets were built by sampling from the larger training set; each was designed to include positive examples of each class, a random sampling of negative examples, and a large number of negative examples that previously generated false positives (i.e., likely boundary cases). For a list of features we found useful for detecting errors, see Table 1. Using Weka [4], for roll–on, roll–off, and repeat errors, we used a decision-tree based approach (J48 decision tree) combined with a metacost algorithm. This approach automatically selected between the possible features from Table 1 and determined the decision boundaries for the classifiers. Due to our desire to avoid incorrectly classifying a correct keystroke as an error, we iteratively constructed these training sets and searched for proper weighting parameters for penalizing false positives (6X) until we were satisfied with the classification performance across the training set. Finally, we assembled our correction system (which we call "Automatic Whiteout") by combining the classifiers in series. First, each keystroke was tested as a key repeat error. If the keystroke was not such an error, the roll–on insertion test was applied. Finally, the roll–off insertion test was applied. Since these errors involve insertions, corrections simply involves removing the insertion from the text stream. The substitution test was not used in the combined system due to difficulty in determining the appropriate correction.

5. RESULTS

Error Type	corrections (possible)	detected	wrong corrections	OBO error reduction
Roll–On	24(39)	61.54%	0	12.90%
Roll–Off	32(37)	86.47%	2	17.20%
Repeats	18(22)	81.81%	0	9.67%
Subs	19(78)	24.36%	0	10.22%
Automatic Whiteout	74(98)	75.51%	2	39.78%

Table 2: Automatic Whiteout successfully corrected 74 of 98 possible corrections of our test set. Automatic Whiteout consists of roll–on insertions, roll–off insertions, and key repeats. Off–by–one substitutions and other off–by–one errors were not used in Automatic Whiteout due to difficulties in correction.

Error Type	Avg. corrections (possible)	Avg. detected	Avg. wrong corrections	Avg. OBO error reduction
Roll–On	27.86(58.57)	47.56%	1.0	10.30%
Roll–Off	52.00(64.71)	80.35%	1.71	19.29%
Repeats	13.86(25.29)	54.79%	.71	5.04%
Automatic Whiteout	93.71(146.71)	63.87%	3.43	34.63%

Table 3: Automatic Whiteout performance averaged across seven user–independent tests. On average, users made 260.71 off–by–one errors.

We validated each individual classifier on the independent test set and then validated Automatic Whiteout as a whole. The results are shown in Table 2. The test set contained 6345 keystrokes, 307 of which were errors. Of the 307 errors, 186 are off–by–one errors. Note that the final Automatic Whiteout classifier detected and corrected the same numbers and types of errors as the independent classifiers.

Figure 3 illustrates the impact of applying Automatic Whiteout to our entire data set. Theoretically 789 of the off–by–one errors would have been detected and automatically corrected reducing the total number of errors typed from 2988 to 2199 (see Figure 3 (right) for a breakdown of errors that remain after successfully applying Automatic Whiteout to the dataset). This result is a 26.41% reduction in total errors. Examining the data from only our independent test set shows an improvement from 307 to 233 errors, or a 24.10% reduction in total errors.

In a product, Automatic Whiteout would not have the benefit of training on its user's typing. Instead, a mini-QWERTY keyboard manufacturer would train the Automatic Whiteout system on the typing of a large set of expert users before embedding it. While we only have typing from seven expert users, we can still simulate such a situation by training on six of these users and testing on the data from the seventh, unseen user. This "leave–one–out" procedure leads to seven combinations of training and test users. Table 3 shows the results from these tests averaged over the seven users.

6. DISCUSSION

Automatic Whiteout successfully detected and corrected over 39% of the off–by–one errors in our expert mini–QWERTY typing test set. We were surprisingly successful in detecting roll–off insertions, while substitution correction proved particularly difficult. Figure 3 compares the distribution of errors from our original typing data to the distribution that would occur while using our Automatic Whiteout algorithm. In theory, approximately 27% of the users' overall errors could have been avoided with almost no negative effect from false corrections introduced by the system. By comparing the results in Table 2 and Table 3, we see that the Automatic Whiteout procedure generalizes well across expert users. Having examples of the user's typing in the training database improves the results but only slightly. These results are encouraging and point to a possible commercial use of the process.

While our goal has been to correct existing errors without introducing new errors into users' typing, any correction algorithm has the potential to make mistakes. First, the system could correct an error incorrectly. For example, the user may have typed "ca**m**," intended "ca**b**," and the systems corrects the input to "ca**n**." These mistakes are unlikely to cause a negative user experience since both the originally entered text and the processed text contain the same type of error in the same location in the word. Cases where the system changes a correct keystroke to an incorrect keystroke are more noticeable and should be avoided. Our test data shows that Automatic Whiteout is very successful in this regard (Table 2). If such mistakes are infrequent, the user experience will still likely be positive since we are correcting many more errors than we introduce. Furthermore, since the user is typing at such a rapid pace and with inherent ambiguity, the miss–correction may go unnoticed. Finally, given the features used to detect errors, the system will not attempt to change the keystroke again if the user backspaces to correct an introduced error or if the user types slowly. Thus, repeated mis–corrections are avoided.

By using features of the users' typing, we can perform error correction without the need for a full dictionary. As noted by MacKenzie et al. [6], using a dictionary for disambiguation in mobile text input has many issues. The greatest limitation comes from managing out of dictionary words (proper nouns, abbreviations, slang, etc.). In informal mobile texting and e-mail, many words may be out of dictionary and encourage inappropriate corrections if a dictionary is employed. Furthermore, many spelling agents (e.g. the one in Microsoft Word) test a word after the user has typed a space, indicating the end of the word. While effective, these spelling agents can distract the user by drawing attention to an error (highlighting or underlining the word). The interruption of the user's attention could potentially slow their typing speed. Dictionaries can also consume large amounts of memory on mobile devices that are often resource constrained. By not using a dictionary, we believe our detection/correction solution could be embedded into the firmware of the keyboard for a mobile phone or PDA. Finally, dictionaries are language specific and by using pattern recognition techniques there is the potential to be more language independent. One key feature of our algorithm is that it uses the immediate context of the keystroke and the frequencies of letter digrams (pairs) and trigrams (triples) of the language. In future work we would like to explore building a letter frequency model that accounts for multiple languages.

Leveraging features of the user's typing and using Automatic Whiteout enables us to detect and correct errors as the user types, often mid–word. As a result, the correction can happen transparently to the user, and errors can be fixed before the character appears on the screen. We believe that such automatic keystroke level correction might allow the user to sustain rapid typing speeds as the user is able to input text without being distracted by errors, a hypothesis we will test in future work.

7. FUTURE WORK

While we are encouraged by our results, many questions remain. First, we are interested in how the algorithm generalizes. For example, does the system need to be trained for each new type of mini–QWERTY keyboard or can a version of the system generalize to many different models? Does Automatic Whiteout work as well when the typist is in a different context, such as typing while not looking at his keyboard? How successful is Automatic Whiteout when employed by non–expert typists? Are more corrections desirable at the expense of more false corrections? We intend to examine these questions using the second keyboard and the "blind" typing conditions from the expert typist database [1, 2].

We are also interested in conducting user evaluations to compare mini–QWERTY typing speeds and accuracies with and without the use of the Automatic Whiteout correction system. A study that gathers live data will allow us to determine the effect, and noticeability, of the system on users. Furthermore it would be very interesting to explore how much the users come to depend on the Automatic Whiteout features as they learn and become expert typists.

8. CONCLUSIONS

We have demonstrated a technique for automatically detecting and correcting up to 26.41% of the total errors made by mini–QWERTY experts while typing. Using pattern recognition techniques we are able to perform corrections without relying on the use of a built–in dictionary. Hopefully, automatic error detection and correction will enable our solution to rapidly detect and correct errors before the incorrectly typed character appears on the screen. By reducing the number of errors displayed to a user, we potentially reduce distractions and enable the user to continue to input text uninterrupted. We have demonstrated a solution that improves the user experience by reducing errors and potentially relieving cognitive load associated with correcting errors. Our hope is that this solution enables users to improve typing rates through reduced distraction and improve the quality of written communication authored on mobile devices.

9. REFERENCES

[1] E. Clarkson, J. Clawson, K. Lyons, and T. Starner. An empirical study of typing rates on mini–qwerty keyboards. In *CHI '05 extended abstracts*, pages 1288–1291, New York, NY, USA, 2005. ACM Press.

[2] J. Clawson, K. Lyons, E. Clarkson, and T. Starner. Mobile text entry: An empirical study and analysis of mini–qwerty keyboards. *Submitted to the Transaction on Computer Human Interaction Journal*, 2006.

[3] J. Clawson, K. Lyons, T. Starner, and E. Clarkson. The impacts of limited visual feedback on mobile text entry for the twiddler and mini-qwerty keyboards. In *Proceedings of ISWC 2005*, pages 170–177, 2005.

[4] S. R. Garner. Weka: The waikato environment for knowledge analysis. pages 57–64, 1995.

[5] J. Grudin. *Cognitive aspects of skilled typewriting*, chapter Error Patterns in novice and skilled transcription typing, pages 121–143. Springer–Verlag, New York, 1983.

[6] I. S. MacKenzie, H. Kober, D. Smith, T. Jones, and E. Skepner. Letterwise: prefix-based disambiguation for mobile text input. In *UIST'01*, pages 111–120. ACM Press, 2001.

[7] I. S. MacKenzie and R. W. Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03 extended abstracts*, pages 754–755. ACM Press, 2003.

[8] R. W. Soukoreff and I. S. MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci. *Int. J. Hum.-Comput. Stud.*, 61(6):751–789, 2004.