ONLINE ADAPTATION FOR MOBILE DEVICE TEXT INPUT PERSONALIZATION

By

Tyler Baldwin

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science

2012

ABSTRACT

ONLINE ADAPTATION FOR MOBILE DEVICE TEXT INPUT PERSONALIZATION

By

Tyler Baldwin

As mobile devices have become more common, the need for efficient methods of mobile device text entry has grown. With this growth comes new challenges, as the constraints imposed by the size, processing power, and design of mobile devices impairs traditional text entry mechanisms in ways not seen in previous text entry tasks. To combat this, researchers have developed a variety of text entry aids, such as automatic word completion and correction, that help the user input the desired text more quickly and accurately than unaided input.

Text entry aids are able to produce meaningful gains by attempting to model user behavior. These aids rely on models of the language the user speaks and types in and of user typing behavior to understand the intent of a user's input. Because these models require a large body of supervised training data to build, they are often built offline using aggregate data from many users. When they wish to predict the behavior of a new user, they do so by comparing their input to the behavior of the "average" user used to build the models.

Alternatively, a model that is built on the current user's data rather than that of an average user may be better able to adapt to their individual quirks and provide better overall performance. However, to enable this personalized experience for a previously unseen user the system must be able to collect the data to build the models online, from the natural input provided by the user. This not only allows the system to better model the user's behavior, but it also allows it to continuously adapt to behavioral changes. This work examines this personalization and adaptation problem, with a particular focus on solving the online data collection problem.

This work looks at the online data collection, personalization, and adaptation problems at two levels. In the first, it examines lower level text entry aids that attempt to help users

input each individual character. Online data collection and personalization are examined in the context of one commonly deployed character-level text entry aid, key-target resizing. Several simple and computationally inexpensive data collection and assessment methods are proposed and evaluated. The results of these experiments suggest that by using these data assessment techniques we are able to dynamically build personalized models that outperform general models by observing less than one week's worth of text input from the average user. Additional analyses suggest that further improvements can be obtained by hybrid approaches that consider both aggregate and personalized data.

We then step back and examine the data assessment and collection process for higher-level text entry aids. To do so we examine two text entry aids that work at the word level, automatic word correction and automatic word completion. Although their stated goal differs, these aids work similarly and, critically, fail similarly. To improve performance, data assessment methods that can detect cases of system failure are proposed. By automatically and dynamically detecting when a system fails for a given user, we are better able to understand user behavior and help the system overcome its shortfalls. The results of these experiments suggest that a careful examination of user dialogue behavior will allow the system to assess its own performance. Several methods for utilizing the self-assessment data for personalization are proposed and are shown to be plausibly able to improve performance.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Joyce Chai, for her years of help and support. It is not an exaggeration to say that without her insight and guidance this work would not have been possible. I owe her a debt of gratitude for guiding me through all stages of the process. Her advice has been beneficial to me not just within the central research context, but in handling the minutia of the process as well. I cannot thank her enough.

Likewise, I would like to extend my gratitude to the members of my committee. Their help has continually been valuable to me over the years, and they have always made themselves available to humor me in whatever capacity. I would like to thank them both for being available in times when I needed their guidance and for giving me the space to work in times when I did not.

Finally, I would like to thank my friends and family for their continued support. In particular, I would like to thank my wife, Agatha, for her infinite love, patience, and dedication. I would like to thank all of the friends I have made during the graduate school process for providing me with enough relaxation to keep sane. And I would like to thank my family for trying their best to minimize the number of times they inquired about exactly when I planned to graduate.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

Mobile devices have become nearly ubiquitous. As of February 2010, there were 4.6 billion mobile phone subscriptions worldwide, a figure that has been steadily growing (The Associated Press, 2011). As the role that mobile phones play in our lives expands, so too does the need for intelligent software systems that aid in their utilization.

At their heart, cellular phones and other mobile devices are inherently personal. In contrast to traditional land-line telephones, cellular phones are often operated by and associated with a single individual. This association is potentially very powerful, as it allows the device's software systems to analyze and adapt to each user's specific needs. However, in many cases, this power has been underutilized. This work will examine and expand upon one of these underutilized areas: mobile device text entry.

The use of the mobile phone has expanded beyond simply telephone calling, moving into the realm of a general communication and life management device. One example of this is the rise of Short Message Service (SMS) text messaging. In 2010, 5 trillion text messages were sent worldwide (Informa, 2011), more than the number of voice calls placed from mobile phones. Additionally, internet access on mobile devices is becoming widely available, allowing users to compose email and search the web. All of these applications, and several others, require users to input text quickly and accurately.

However, text input on mobile devices is a non-trivial problem, due primarily to their small size. Because of the limited amount of space, mobile devices must economize in some manner. There are three ways in which this is undertaken in modern devices: 1) by using a small set of buttons and assigning several characters to each key, 2) by severely reducing the size of the buttons, and 3) by providing an on-screen software keyboard.

All of these approaches have advantages and drawbacks, and none are able to completely address the problem. Reducing the number of buttons allows each button to be reasonably sized, but may require users to hit each button several times in order to input a single character. Providing a full keyboard of reduced size buttons resolves this problem, but these buttons may be difficult to press unambiguously, and difficult to see for elderly users. Software keyboards are able to provide full sets of keys and are often able to produce larger buttons than their hardware counterparts, but they suffer by not being able to provide sufficient tactile feedback which is necessary for highspeed touch-typing.

Researchers have proposed several methods to help ease some of the text input problems caused by the reduced user interfaces of mobile devices. In one, targeted toward full keyboards, the system attempts to resolve problematic input as it is typed. Ambiguous input that arises from typing in between keys or hitting several keys simultaneously is resolved by examining sourounding characters (Goodman et al., 2002). Using knowledge of the language and general typing behavior, these systems are able to identify the character that the user was most likely attempting to type.

Another common approach is to employ a mechanism to predict the intended term from limited or erroneous input. The popular T9 "text on 9 keys" input method (Grover et al., 1998), available on many reduced key set mobile phones, is one such example. The T9 algorithm attempts to reduce the number of keystrokes a user must input by automatically detecting the intended character from an ambiguous keypress. This allows the user to type purposely ambiguous input, relying on the system to understand their intent.

Automatic word completion, often termed *autocompletion*, is yet another common text input aid. Autocompletion systems attempt to predict the term a user wishes to type after the user has only entered the first few characters. If the predicted term is correct, the user can potentially speed up their input by typing fewer keystrokes per word.

The final text input aid in common use on mobile phones is automatic word correction, or *autocorrection*. Autocorrection systems automatically change misspelled words into the

correctly spelled intended terms. This mechanism can help users avoid common spelling mistakes and can reduce the amount of time they must spend proofreading their input.

The work outlined here does not propose a new text input aid to replace these existing systems. Rather, it proposes to advance a set of user personalization-based improvements to the existing and well established methods, in order to expand upon their utility. A closer examination of individual user behavior has the potential to make each of these technologies more accurate and reliable, helping to further reduce the burden that small mobile device interfaces impose on systems that rely on text input. As we may wish to employ personalized methods for previously unseen users and handle cases in which user behavior changes with time, we ideally would like to adapt to user behavior online. To enable this, this work focuses primarily on overcoming a hurdle to system online adaptation and personalization, that of online data collection. This problem is one of deciding how to interpret an ever increasing input stream provided by the user and how to use this data to allow the system to gain a more complete and robust understanding of an individual user's behavior.

A generic approach to collecting personalization data is shown in Figure 1.1. We start with user input, where the user interacts with the system. We feed this input into some module of system assessment, which tells us whether the system acted correctly or whether it made a mistake. Over time, we are able to build a data set of cases that show when the system performed correctly and when it did not. Using existing model building methods, we can then use this database of correct and incorrect performance examples to build a new model that is tailored towards the individual user's behavior.

This entire process is reasonably straightforward, with the exception of the system assessment module. It must be able to take a piece of input from the user and determine whether the system acted correctly, without any explicit user intervention to inform it of the correct answer. It must be able to produce judgments with high accuracy so that the noise it introduces into the system is not higher than the benefits it produces. To maintain this high accuracy it must be able to remove instances that it is not confident about, without

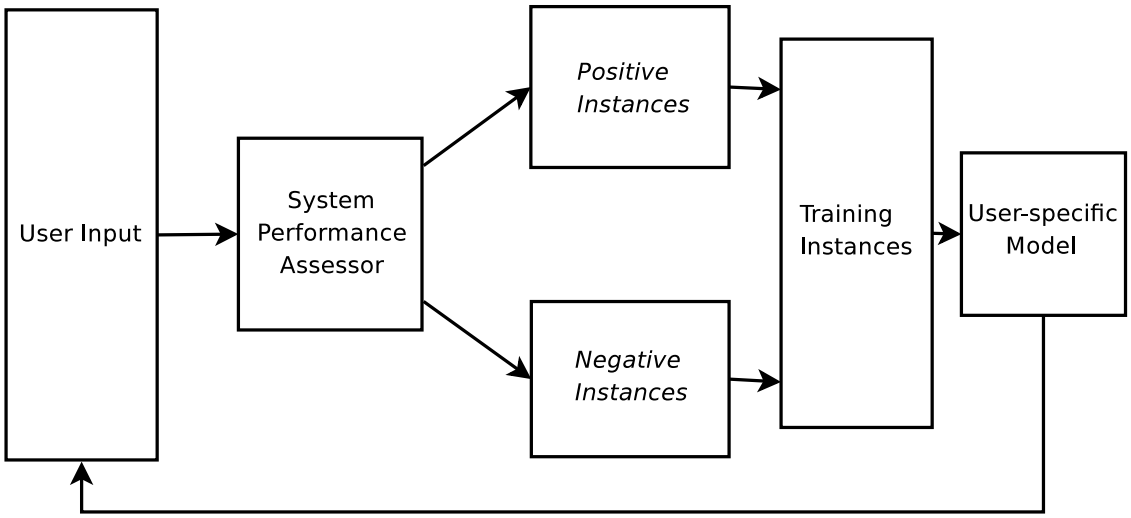


Figure 1.1: Generic approach to collecting data for a personalized system

removing so much data that no model can be learned. Addressing these concerns in the context of mobile device text entry aids and examining the effects of different methods on online adaptation is the primary focus of this work.

This work will examine online data collection and personalization for text entry aids at two different scopes: character-level and word-level aids. Character-level aids are those that are designed to help users input each individual character. Several character-level aids have been proposed, such as those which highlight (Paek et al., 2010) or resize (Al Faraj et al., 2009) keys to help users type faster. However, many of these overt aids can be distracting for the user, and have thus not seen widespread adaption. As such, this work will focus on improving the non-obtrusive character-level aid of key-target resizing (Goodman et al., 2002). Key resizing techniques change the size of each key on a virtual keyboard dynamically to make those keys that the user has a high probability of hitting next larger, and those which are unlikely to be hit next smaller. Because changing the visual appearance of the keyboard after every keystroke can potentially distract the user, key-target resizing techniques change the underlying hit area for each button without changing the visual appearance of the keyboard.

When a user touches the soft keyboard, key-target resizing algorithms incorporate knowledge of both the language and the touch position to predict which key the user intended to type. To do this, they must first build a model of both the touch behavior and the language. This is traditionally done offline, where general models are built to be used for all users. However, this may be insufficient, as individual users may exhibit different behavior. A few recent studies have hinted at the fact that personalizing these models will lead to increased prediction performance (Himberg et al., 2003; Rudchenko et al., 2011).

The previous work on personalization of key-target resizing models has several drawbacks, which this work improves upon. Previous work collected data through a tightly controlled experiment style; this work utilizes a more relaxed data collection style that retains necessary controls to insure experimental validity but allows for the collection of data that more closely resembles real usage behavior. Previous work considered only overall keystroke error rate as a performance metric; this work examines several different ways to quantify performance. Previous work only examined personalization on a single device; this work studies the effect across two different devices.

More critically, previous work has entirely failed to address the fundamental hurdle for personalization systems: data collection. Building a model of user touch behavior requires knowledge of what key the user intended to type with each press they make on the keyboard. This is not a problem for general models, which can be built from data collected in controlled laboratory settings where intent is known. However, this same methodology cannot be used in building personalized models, which must collect data online in real usage environments. In order to be able to use the data collected in this manner, there must be some mechanism to determine the fidelity of each keypress. That is, each time that a user presses the keyboard we must determine whether or not the key that the system predicts that the user pressed is actually the key that the user was attempting to press. This work examines information at the character, word, and discourse level to make this distinction.

Word-level aids are those that help users type individual words faster or more accurately.

This work will examine personalization in two word-level aids that are in frequent use in current generation mobile devices, autocompletion and autocorrection. These aids work similarly and are often conflated with one another. They both work on the notion that if the system can understand the user's intent for input that is either erroneous (autocorrection) or incomplete (autocompletion), it can help the user to better form their message. However, since different users have different lexicons and usage patterns, an autocorrection or autocompletion system that works well for one user may be a hindrance to another.

To overcome this, this work proposes that personalization techniques be applied to autocorrection and autocompletion. As before, in order to enable personalization we must be able to assess the fidelity of input in an online manner. This work proposes two tasks that enable online data collection: 1) dynamically assessing whether an autocorrection or autocompletion system has made a mistake and 2) determining what the correct behavior should have been (e.g., which word should it have corrected to).

Once an autocorrection or autocompletion system has the ability to assess its own performance it has access to a number of personalization methods. This work discusses and shows the feasibility of several of these personalization methods. In one, the system utilizes knowledge of its overall performance to apply more aggressive correction policies for some users and more conservative policies for others. In another, the system uses the assessment data to expand its internal dictionary, tailoring its vocabulary to the specific user and helping it to avoid making the same mistakes twice.

The rest of this thesis is organized as follows: In Chapter 2, I present the relevant background work necessary to understanding the current state-of-the-art and the work outlined in further chapters. The next four chapters present the online data collection and personalization work done to address the problems outlined above. In the first two chapters, I propose methods to utilize online data collection strategies for personalization to improve the input of each character as it is being typed by resolving ambiguous input in real time. In the next two chapters, I propose online data collection strategies based on self-assessment

that allow personalization to improve the existing text input aids of autocompletion and autocorrection. Finally, Chapter 7 summarizes the contribution of this work and takes a look forward at the future of personalization and adaptation techniques in this and other domains.

CHAPTER 2
Background

This chapter provides the necessary background material for understanding the work and discussion in the rest of this thesis. This chapter discusses the state-of-the-art in the field of mobile device text entry, including a more in-depth discussion of those text entry aids introduced in the introduction. It examines the challenges posed by mobile devices to the text entry problem, and examines how current generation text entry aids attempt to address them. It also gives a brief overview of text entry evaluation metrics. Once an understanding of the field and its challenges have been established, it examines attempts at personalization and system self-assessment in related domains. But first, the following section sets the stage with a brief discussion of historical work on text entry.

2.1 Text Entry Approaches

The field of text entry is by no means new and by no means restricted to mobile device text entry. Prior to the rise of mobile devices, text entry systems were already one of the primary means of interacting with personal computers. And prior still to the rise of the personal computer, the typewriter necessitated some manner of text entry system. The primary solution for both typewriter and later personal computer text entry was the QWERTY keyboard (Sholes et al., 1868), as shown in Figure 2.1.

Although the QWERTY keyboard layout has been widely adopted, it is generally believed to have a sub-optimal key arrangement (MacKenzie and Tanaka-Ishii, 2007). In the over 100 years in which the QWERTY layout has been standard, several alternative keyboards have been proposed. One of the earliest and most (relatively) successful alternatives was the

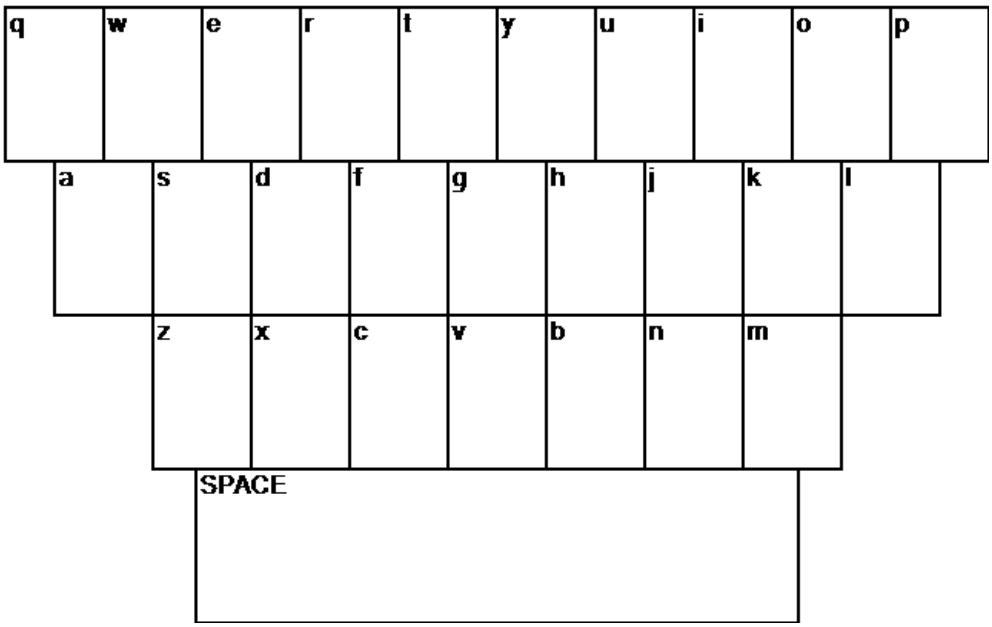


Figure 2.1: Layout of the QWERTY keyboard. Sourounding non-alphabetic characters may change depending on implementation.

Dvorak Simplified Keyboard (Dvorak and Dealey, 1936). The Dvorak design drew insight from studies of physiology and rudimentary studies of letter frequency in an attempt to improve key positioning. Nonetheless, the layout failed to see significant adoption.

The QWERTY keyboard remained the dominant input method after the move from typewriters to computers. However, the rise of mobile devices created new opportunities for alternative text entry methods. Critically, this allowed for and necessitated the development of *software keyboards* (sometimes referred to as *virtual keyboards* or *soft keyboards*), keyboards implemented in software rather than hardware. This meant that alternative keyboard layouts could be easily examined, which led to a rekindling of interest in layout design.

One of the earliest commercial alternative keyboard layouts was the Fitaly keyboard (TextwareSolutions, 1998). The fitaly keyboard was designed for typing with one hand, with a stylus. Although current touchscreen mobile devices allow both one-handed and two-handed input, stylus-based mobile devices generally required one-handed input. The design

principals behind the fitaly keyboard included the placement of commonly used keys close to the center of the keyboard and the inclusion of two large space keys.

MacKenzie and Zhang (MacKenzie and Zhang, 1999) took a more scientific approach to keyboard design, studying several aspects of typing behavior, including models of the language and of movement behavior. To model language they studied letter digraphs (commonly referred to as bigrams in other sources of literature), which models the probability of a two character sequence from existing corpora. To model typing behavior they used Fitts' law (Fitts, 1954), an information theoretic model of human movement behavior which gives the expected movement time between two keys. Fitts' law, as it is commonly written in text entry literature (MacKenzie, 1992), is defined as follows:

$$MT = a + b \log_2 (\frac{A}{W} + 1) \tag{2.1}$$

where MT is the movement time, W is the width of the keys and A is the amplitude of movement (the distance from the center of one key to the other along the axis of movement). The constants a and b vary depending on the form of movement, and can be empirically determined from data. Several studies have been conducted to calculate these constants for different modes of input, including mouse (Card et al., 1978) and stylus movement (Mackenzie et al., 1991).

Using these models of typing behavior MacKenzie and Zhang were able to design a keyboard that outperformed the QWERTY and fitaly models, which they dubbed OPTI. Although the OPTI layout drew inspiration from models of input behavior, the actual placement of keys was still done largely by trial and error. To improve upon this Zhai et al. (2000) used random walks to find a keyboard that minimized Fitts' law movement time based on digraph probabilities. This keyboard was named the Metropolis keyboard. Simulation experiments suggested that the Metropolis layout would outperform the OPTI and FITALY layouts.

Several other input methods that do not follow the standard keyboard paradigm have also

been proposed. Much of the early work focused on handwriting and pen-gesture recognition (Tappert et al., 1990; Hu et al., 1996; Schomaker, 1998). Another proposed input method was shapewriting (Zhai and Kristensson, 2003; Kristensson and Zhai, 2004), which allows users to input text on a touch screen keyboard without picking up their finger or stylus between characters. Input via eye gaze has also been considered (Frey et al., 1990). At present, none of these things have seen widespread adaption and as such will not be discussed in detail here. See MacKenzie and Tanaka-Ishii (2007) for a review.

2.2 Text Entry Challenges

Mobile devices impose several constraints on users that were previously negligible or nonexistent in desktop computer text entry. Although the size of the device is one of the most critical differences, the experience is also effected by several user and situation dependent factors, such as the viewing angle and the size of each user's thumb.

Computational Limitations. One of the most straightforward limitations of mobile devices relative to typical personal computers is the lower amounts of processing power and memory. It is important that the entry mechanism is not overly taxing on the system, as users expect it to react swiftly to their commands. Thankfully, most approaches to text entry do not require a significant amount of processing power or memory. Nonetheless, for more sophisticated methods such as key-target resizing (discussed in Section 2.3.1) this may still be an critical limitation.

General Size Related Constraints. The small size of mobile devices causes several problems that effect text input and constrains the design of the input mechanism. At the hardware level, there are 3 commonly utilized solutions: (1) reduced key keyboards, (2) full keyboards with small keys, and (3) onscreen text entry. Each of these solutions creates additional problems that hinder fast and accurate text input.

Reduced key keyboards are keyboards that have a smaller number of keys than the character set they allow a user to enter. To cover the entire character set, several different

Figure 2.2: Common 12 key keyboard used for mobile device text input.

characters must be associated with a single key. A commonly utilized input method on reduced key keyboards is the multitap method, in which tapping the same key several times in succession produces different characters depending on the number of times the button is tapped. This method requires several keystrokes for each character entered, slowing down text entry. For instance, to enter the word *fox* on the reduced keyboard shown in Figure 2.2[1] using the multitap method a user must enter the sequence 33366699.

The second option, keyboards with small keys, removes the ambiguity seen in the reduced key keyboards. However, to achieve this they must reduce the size of each key to be very small. This has the effect of making each individual key harder to see and press. Additionally, because many more keys have to be accommodated than on a reduced set keyboard, more space must be devoted to the keyboard, reducing the size of the screen or increasing the overall size of the device.

[1]The image in Figure 2.2 was produced by Wikipedia users Silsor and Marnanel and is distributed under a Creative Commons BY-SA 3.0 license.

The final option, onscreen text entry, has a number of advantages. Because no extra space is necessary for keyboard buttons, the screen can be large, which can be a boon to applications that require or are enhanced by a large view, such as web browsing and video playing. Additionally, the keyboard layout is dynamic; it can easily be switched between layouts appropriate for different languages and tasks, and easily hidden when not needed. Onscreen entry also allows alternative input methods such as shapewriting and gesture input. However, input on software keyboards is often more error prone than other methods, due to the lack of tactile feedback and the inherent ambiguity in the touch points.

Lack of Tactile Feedback. One of the primary problems with soft keyboards (and to a lesser extent hardware keyboards with small buttons) is that it is difficult for users to know when they have selected a key without focusing their eyes on the keyboard. This is because soft keyboards lack the tactile feedback provided naturally by a hardware keyboard, which gives users a haptic signal that they have pressed a key. Haptic feedback is critical in touch typing, as it orients the user without them having to examine the keyboard (Rabin and Gordon, 2004). The lack of tactile feedback on soft keyboards makes it more difficult for the user to know when their finger has slipped off of the intended key (Hoggan et al., 2008) and thus forces the user to spend more time focusing their gaze on the keyboard area (Paek et al., 2010).

Parallax. Parallax is a visual effect whereby an observer viewing an object from different positions will arrive at different conclusions about its actual location. On mobile devices, parallax effects can change the perceived position of a button and the user's finger, which may lead users to select different points at different angles. Even when the viewing angle is constant, parallax effects may cause users to consistently miss an intended target (Holz and Baudisch, 2011).

Occlusion. Another problem faced by keyboards with small keys (both software and hardware) is occlusion. That is, when a user attempts to acquire a small target their finger blocks their view, making it hard to select the exact point the they intended. The small keys

on full keyboards can be occluded by the user's fingers while typing, potentially resulting in misselection. Occlusion may also make it harder for users to confirm the selection they have made, resulting in a higher frequency of uncorrected errors. Vogel and Baudisch Vogel and Baudisch (2007) argue that occlusion is one of the main problems that lead to high input error rates.

Fat Finger. One commonly referred to explanation for the high error rates on touch screen devices (and to a lesser extend on small button hardware devices) is the "fat finger" problem (Balakrishnan and MacKenzie, 1997; Vogel and Baudisch, 2007). Although it is a problem for fingers of any size, the fat finger problem is so named because it is exacerbated as the size of the user's finger is increased. According to this hypothesis, the softness and size of user's fingers causes touch points to diffuse out over a large area, leading to a blob of noisy input. Several attempts to acquire the same target will result in different points because of subtle differences in the way the finger's area diffuses over the touch screen. Because of this, fat finger errors are often seen as noise that we cannot correct for, putting a lower limit on the size of buttons we can hope to accurately select.

Perceived Input Point. An alternative hypothesis for the variance seen between attempts at selecting the same point is the perceived input point model (Vogel and Baudisch, 2007; Holz and Baudisch, 2010). In this model, differences in acquired point are attributed primarily to 2 factors: the internal model of point selection utilized by the user and the angle at which they acquire the touch point. When the user makes a selection, they have an internal model of their selection behavior, which may not correspond to the model that the system uses to determine the selected point. Many users perceive that the point they are selecting is near the tip of their finger (Benko et al., 2006), but touch screens generally register the middle of the touched area as the intended point. This creates a disconnect between what users think they are selecting and how it is interpreted by the system. Because different users may have different internal models, between user variance is expected.

Holz and Baudisch (2010) generalized this model to consider the angle at which user's

approach the target. Their hypothesis was that the differences between the perceived and the registered points were systematic when you control for user and angle, and can thus be corrected for. To test this, they asked several users to attempt to select the same point on a touch pad multiple times, changing the pitch, roll, and yaw of their finger after a few selections. The results were consistent with their hypothesis. Users differed in the extent to which angle changed the points they selected, but when user and angle were controlled for the amount of variance was small. This suggests that if we know the angle and the the user (and have the appropriate models for their touch behavior), we should be able to select the point that the user intends with high accuracy. Unfortunately, modern mobile devices do not have the ability to detect the angle at which they have been pressed, meaning that the variation attributable to angle is still seen as noise.

Language Model Pitfalls. Language models are a key aspect of many of the text entry aids discussed in the next section, as well as many other fields that deal with language such as automatic speech recognition (Jelinek, 1976), spelling correction (Kernighan et al., 1990; Toutanova and Moore, 2002), and machine translation (Brown et al., 1993; Koehn et al., 2007). Although the term language model could be thought of generally to include any model of human language employed by the system, it is most often used in the literature to refer to statistical n-gram language models (discussed in Section 2.3.1). Language models give the system some notion of how likely a given character or word sequence is to appear in the language, and their overall contribution to automatic linguistic interpretation has been substantial. Nonetheless, the clumsy or insufficiently constrained application of language modeling can lead to some unintended consequences.

The problem arises when we consider that there are many use cases for text entry on mobile devices, and the user may be apt to type very different things in each of these cases. For instance, a model trained on perfectly grammatical sentences may not be appropriate for web search, where users often input a string of keywords rather than a grammatical sentence. Similarly, other use cases such as web address input would be expected to have

unique models. A smaller but more difficult to address problem arises from variation within one application; a user writing an e-mail to a boss is likely to employ a different writing style than they use when writing e-mail to friends. Similar problems can arise when a bilingual user attempts to type concurrently in both languages. None of these problems are enough to outweigh the significant gains provided by language modeling, but they do need to be addressed.

2.3 Text Entry Aids

This section outlines some of the solutions that researchers have developed to address the text entry challenges outlined in the previous section. It is divided into two general sections covering character-level and word-level text entry aids, respectively. Character-level aids are those text aids that work at a lower level of discourse processing to attempt to help the user input each character they select. Word-level aids attempt to consider the larger context to ensure that each word or phrase displayed corresponds to what the user intended.

2.3.1 Character-Level Entry Aids

One possible solution to the ambiguity caused by small key sizes is to simply make the keys bigger. However, making the overall size of the keyboard larger is generally not practically possible. In the case of hardware keyboards, this means either reducing the size of other components or expanding the dimensions of the device. For soft keyboards, it means taking screen real estate away from other widgets. However, because soft keyboards can be changed dynamically, there is another option: only increase the size of buttons when the user is likely to select them next. This is the approach taken by the BigKey keyboard (Al Faraj et al., 2009).

BigKey uses character level bigram probabilities estimated from corpora (Mayzner and Tresselt, 1965) to predict which key the user is likely to hit next. It then expands the size

of the four most probable keys by a fixed amount, such that the most probable key is larger than the second most probable key, etc. By doing this, it makes those keys that the user is likely to select larger and easier to acquire. Results of a small study presented by Al Faraj et al. (2009) suggest that the BigKey approach was able to improve both typing speed and accuracy over a standard keyboard that did not enlarge keys.

Similar work was performed by Himberg et al. (2003). In this work, users were given sequences to type on a 9-digit number pad. Every time the user pressed a key, the shape of the key would update based on the user's touch point. The update process moved the center of the key towards the new touch point, gradually warping the shape of the keyboard to accommodate the user's touch behavior. Although the approach is relatively simple, this represented the first attempt in the literature of updating the touch model for online keyboard personalization.

Despite the initially promising results, there are several drawbacks to this type of approach to key resizing. On the BigKey keyboard, The choice of a fixed number of keys resized to a fixed width allows for fast implementation, but may not accurately reflect the underlying probability distribution. In some cases the most probable key will be much more likely than the second most probable, and in others the top several keys may have similar probabilities. This approach also creates problems when two of the enlarged keys are next to one another, as their areas may overlap. As for the approach taken by Himberg et al. (2003), the update process could be skewed by users attempting to select different points to accommodate to the changed keyboard, which would mean that user behavior would be constantly changing because of the keyboard changes. Additionally, users found the constant changes to key sizes distracting. A related line of work has attempted to solve these problems by performing key resizing more covertly.

Key-target resizing algorithms work on the same principle as key resizing algorithms such as BigKey, with one important difference: the changes are not shown to the user. Instead of making the displayed buttons larger, key-target resizing algorithms expand the

underlying hit area of probable keys, while leaving the display unchanged. This eliminates the problems associated with dynamically updating the keyboard area, such as user distraction and overcorrection. In order to rid itself of these issues, covert resizing trades some of the benefits of key resizing, most notably its ability to highlight for the user characters they are likely to want to select next. Nonetheless, despite the fact that the user does not know that resizing is taking place, they are still able to reap some of its benefits, in that those keys that are most probable are also the easiest to hit.

Although the simple resizing mechanism of BigKey could also be applied to key-target resizing, state-of-the-art key-target resizing algorithms implement several improvements. These algorithms reason that instead of only resizing certain keys, we should resize every key, and instead of resizing by a fixed amount, keys sizes should be dependent on their relative probability. In addition, in computing this probability, we should not only consider the likelihood of each key based on the language, but also based on some model of typical typing behavior as well.

The common formulation of modern key-target resizing work is presented by Goodman et al. (2002), who derive a probabilistic model of typing behavior, based on a noisy channel formulation. Their reasoning is as follows: when a user attempts to type a sequence of characters on a soft keyboard, the system receives this as a series of touch points. To translate these touch points into the intended character sequence we must calculate the most likely character sequence given the touch points:

$$\arg\max_{c_1,...,c_n} P(c_1, ..., c_n|t_1, ..., t_n) = \arg\max_{c_1,...,c_n} P(c_1, ..., c_n)P(t_1, ..., t_n|c_1, ..., c_n) \tag{2.2}$$

where $t_1, ..., t_n$ is the sequence of touch points typed by the user and $c_1, ..., c_n$ is a sequence of corresponding characters[2]. The probability $P(c_1, ..., c_n)$ is the language model, which

[2]Equation 2.2 can be derived via Bayes' rule: $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$. Because we are finding $\arg\max_{c_1,...,c_n}$, we can drop the denominator, $P(t_1, ..., t_n)$, which will be constant across $c_1, ..., c_n$.

captures how likely the sequence $c_1, ..., c_n$ is to occur in the language. This probability can be calculated as:

$$P(c_1, ..., c_n) = P(c_1) * P(c_2|c_1)... * P(c_n|c_1...c_{n-1}) \tag{2.3}$$

However, this is difficult to calculate. For instance, To calculate the probability of the final letter of the previous sentence ($P(e|$*However, this is difficult to calculat*$)$), we need to be able to calculate the probability of every possible sequence of 39 characters in the language. Even if we restrict our character set to 30 characters, enough for the letters in the English alphabet and a few punctuation marks, this gives us an impractically large number of probabilities to calculate ($4.05 * 10^{57}$), which we can not hope to accurately estimate from existing corpora. Instead, we make the *n-gram assumption* that a given character in the sequence is only dependent on the previous $n-1$ characters:

$$P(c_i|c_1, ..., c_{i-1}) \approx P(c_i|c_{i-(n-1)}, ..., c_{i-1}) \tag{2.4}$$

As n grows, the number of probabilities that must be computed expands exponentially. Because of this, word-level language models are often restricted to a small n (e.g., 2 or 3) due to the number of possible word combinations. The relatively small number of possible symbols allows character level n-grams to be larger; Goodman et al. (2002) use 7-grams. To account for unseen character sequences, modified Kneser-Ney smoothing is performed (Chen and Goodman, 1998; Kneser and Ney, 1995)[3].

The second probability we must estimate, $P(t_1, ..., t_n|c_1, ..., c_n)$, models user touch behavior. Because no corpus of letter-touch point pairs was available, Goodman et al. (2002) collected their own. However, collecting touch behavior is problematic because it requires users to type on the keyboard prior to the model being built. One option would be to use no key-target resizing when collecting data, but this may lead to users attempting to adapt

[3]Smoothing is a important aspect of language modeling, but unfortunately an in-depth discussion of it is too far afield from this work. See Goodman (2001) for a review.

their behavior to the more restrictive keyboard. Instead, Goodman et al. (2002) use a more permissive data collection methodology: they ask users to type a certain phrase and (because they now know what key the user intends to hit) count as correct all points that are "close" to the intended key[4].

From their data collection, Goodman et al. (2002) were able to observe several properties of touch behavior: 1) the average touch point for each key was not the center of the key, but a point slightly lower and shifted towards the center of the keyboard, 2) the variance in x and y coordinates differed, 3) touch distributions did not fall along the axes of the keyboard (they were rotated slightly), and 4) the sides of long keys (e.g., spacebar) had different variances. It should be noted that this study used stylus-based touch input on a small corner of a tablet PC, so these observations may not hold when using other input methods, such as thumb or finger input, or on actual mobile devices.

Based on their observations, Goodman et al. explored several different models in their pilot study, but found that none of these models outperformed modeling the touch behavior based on a standard bivariate Gaussian. The probability of each point given a character is estimated by modeling the distribution as a bivariate Gaussian distribution and the probability of a sequence is modeled as the product of the individual point probabilities. Note that we are again making a simplifying assumption that the current touch point is not dependent on previous touch points:

$$P(t_1, ..., t_n|c_1, ..., c_n) = P(t_1|c_1) * P(t_2|c_2)... * P(t_n|c_n) \tag{2.5}$$

By calculating the probability this way we are able to incorporate 3 of the 4 above observations in our model. The only one not accounted for, the difference in variance on different sides of long keys, is likely to be the least important; bigger keys are generally easier for the user to hit anyway.

[4]Goodman et al. (2002) do not define what constitutes close in their data collection. Later work (Gunawardana et al., 2010) consider any touch point within the intended key or any of the adjacent keys to be correct.

To test their methods, Goodman et al. performed a user study comparing their key target resizing algorithm to a static keyboard with no resizing. Users had to complete several rounds of testing, in which they were given a phrase to type on the keyboard. After a small phase of a few rounds to give users time to get used to the keyboard and experimental setup, users were asked to complete several rounds on both the static and target-resizing keyboards. To simplify calculation of error rates, users were not allowed to correct their input mistakes. Their results suggested that a keyboard with key-target resizing could reduce input error rates by a factor of 1.67 to 1.87.

Gunawardana et al. (2010) introduced several minor but important improvements to the key-target resizing methodology. Firstly, they transform the probability calculation into a calculation of the most probable letter, rather than the most probable sequence. The sequence calculation has two distinct disadvantages. The first disadvantage is that it can potentially change a sequence on the fly. That is, when new characters are typed by the user the algorithm may examine the sequence and decide that a different sequence is now more probable, resulting in a change in previously displayed characters. Initially, this seems like a preferable characteristic for the keyboard to have, since the introduction of new evidence may make the user's intent more clear. However, the problem is that this behavior defies user expectations. Users may examine the text field only periodically, and they may not expect previous characters they have written to change. This point is discussed further in Section 2.3.2.

The other problem with sequence-level computation is simply one of device resource limitations. Performing sequence-level computation takes more memory, processing power, and time than word-level processing, and all of these things are at a premium. If computation is too expensive, there may be a noticeable delay between a user selecting a character and it appearing on the screen, which is likely to be distracting to the user. Although resource limitations are likely to be alleviated somewhat as devices become more powerful, at the present time these factors still weight heavily on design decisions.

Thankfully, transforming equation 2.2 into a word-level equation is fairly trivial. Because we already have made simplifying assumptions about what each probability is dependent on, we can simply extract these probabilities for our word-level computation:

$$\arg\max_{c_i} P(c_i|t_i) = \arg\max_{c_i} P(c_i|c_{i-(n-1)}, ..., c_{i-1})P(t_i|c_i) \tag{2.6}$$

The second important feature added by Gunawardana et al. (2010) is key anchoring. Key anchoring ensures that the areas directly at the center of the keys are always associated with that key. This is meant to handle an unintended consequence of key-target resizing: because the language model probability is so high for some sequences, it may expand the size of the corresponding key to such a degree that adjacent keys are difficult to hit. Because the algorithm does not show the resizing to the user, they may not realize that this has occurred. Key anchoring largely alleviates this problem by always allowing the user to select the correct key when their presses are centered on it.

Lastly, Gunawardana et al. introduce a slightly modified evaluation procedure. As before, users were given phrases to type "as quickly and accurately as possible". However, instead of having users type on keyboards with different resizing algorithms, all users typed on a keyboard that used the permissive keyboard style used for data collection by Goodman et al. This provided both the touch sequences produced by the user and the corresponding intended keys. Using this information, evaluation can be done offline by calculating the key predictions that a resizing algorithm would have produced on the data.

The phrase input methodology used in key-target resizing training and evaluation is motivated primarily from its use in studies that compare differing keyboard layouts, as described in Section 2.1. MacKenzie and Soukoreff (2003) discuss the motivation behind this methodology and provide a set of these phrases for testing. When you wish to compare two different layouts, you must have users type on each to discover their relative merit. While an experimenter could simply have each user type for a while on both and compare the performance, without carefully controlling the input there is likely to be a high variance between sessions

that is not attributable to differences in layout. Some of the likely confounding factors would be differences in the words they chose to write (e.g., how easy they are to spell) and the amount of time each person spends thinking about what they are going to say. Additionally, letting users decide what to type makes it difficult to calculate performance, which needs a gold standard text to compare to (see Section 2.4.2). By asking the users to copy phrases, experimenters attempt to control for these factors. This increases the *internal validity* of the experiment, the chance that the experiment's conclusion is attributable to the factors that the experimenter wishes to test.

Conversely, the *external validity* of the experiment refers to the generalizability of the results to other situations. Controlling factors tends to improve internal validity, but may have an adverse effect on external validity. For instance, the majority of real world use cases for text input do not involve copying text, so it is questionable how well the phrase copying methodology can generalize to the average use case. MacKenzie and Soukoreff (2003) conclude that, for comparing two systems, the internal validity gained by using the phrase copying methodology outweights the external validity lost. While this seems to be a well reasoned argument for phrase copying for layout comparison, many of their points do not directly apply to key-target resizing research. As such, Chapter 3 argues that within the context of key-target resizing we can relax our experimental methodology to improve external validity without an adverse effect on internal validity.

2.3.2 Word-Level Entry Aids

Several text entry aids have been proposed that work at the word level. Unlike character-level aids, word-level aids may interpret and change more than a single character at one time. They are generally less concerned with helping ensure that users hit the correct key, and more concerned with helping users enter words fast and non-erroneously.

One class of text entry aids, often called *predictive text aids*, attempts to speed up text entry by interpreting ambiguous input. These aids are targeted primarily towards mobile

devices with reduced-key keyboards, as the small number of keys necessitates that each key is associated with several characters. The default multitap input method often requires users to press each button several times to produce the desired input, slowing down the overall input speed. Using a predictive text aid, users can input ambiguous keypresses and let the system interpret their intent. The most widely deployed example is the T9 "text on nine keys" system (Grover et al., 1998). With this aid, users input words by hitting an ambiguous key only once for each character that they intent to input. For instance, to input the word *good* using the T9 method, a user would enter 4663 on the keyboard in Figure 2.2. This sequence of keys could correspond to several different words (e.g., good, hood, home), so T9 picks the most likely term from its dictionary based on the language and the context. In the event that the system misinterprets the input, users can select the correct interpretation from a ranked list of possible words.

Another commonly deployed word-level aid is automatic word completion, sometimes referred to as word prediction or autocompletion. Word completion systems attempt to increase text input speed by predicting the word the user intends to type before the user has finished typing it. Word completion has the potential to reduce the number of keystrokes needed to input a word to less than the number of characters in the word. Because of this, it has seen adaption is a variety of different uses cases (Bast et al., 2008; Greenshpan et al., 2009; Khoussainova et al., 2010), but is particularly useful for instances where the users input ability is challenged or impaired (Garay-Vitoria and Abascal, 1997).

Like key-target resizing, autocompletion systems rely heavily on a statistical representation of the language provided by n-gram language models. Although the earliest systems used unigram models (1-gram), later systems used larger values of n to consider the wider context. Given the prefix of the word that has been typed by the user thus far, we examine words that begin with this prefix to find the one that has the highest language model probability in the current context. If we wish to return several candidate results, we can simply rank them by their language model probabilities.

Although n-gram language modeling is at the root of the autocompletion procedure, several modifications have been proposed to improve overall performance. Several models have been proposed to incorporate additional linguistic constraints using information from part-of-speech (Fazly and Hirst, 2003), syntax (Renaud et al., 2010), and semantics (Li and Hirst, 2005). Topic modeling has also been examined (Trnka et al., 2006) to help the language model better handle a variety of use cases. Some models also attempt to handle cases where users may have misspelled their initial input.

A similar but distinct text entry aid is *automatic word correction* (autocorrection). As the name suggests, autocorrection systems attempt to automatically correct words that users have misspelled. Autocorrection technology grew naturally out of work in the well-studied field of spelling correction. The basic premise is the same; misspelled terms are identified and corrections are proposed. The core difference is that autocorrection systems must make a decision about a misspelled word and its correction and implement it without intervention from the user.

Modern spelling corrections systems are based on a noisy channel model (Kernighan et al., 1990; Mays et al., 1991). The problem formulation is as follows: given a misspelled word form t (as identified by checking it against an internal dictionary), find the correct word c that corresponds to what the user intended to type. To do so we select the most probable word from our internal dictionary, by calculating $P(c|t)$ for each word:

$$\arg\max_{c_i} P(c_i|t) = \arg\max_{c_i} P(c_i)P(t|c_i) \tag{2.7}$$

As in equation 2.2, there are two probabilities that we must estimate: $P(c_i)$ is the language model as we have seen previously and $P(t|c_i)$ is the error model. The language model estimates the likelihood of c_i appearing based on the sourounding context and is calculated via the statistical n-gram models discussed previously.

The error model estimates how likely it is that the given misspelled word form would arise from an attempt to type the word c_i. To estimate $P(t|c_i)$, Kernighan et al. (1990) observed

that many spelling mistakes are only a single editing operation away from the intended word. To take advantage of this, they calculated their error model probability by examining only cases in which the misspelling could be transformed into the candidate word by a single editing operation. The error model probability was then estimated by the probability of the corresponding edit operation, as estimated from corpora.

Later work expanded the error model to allow more than a single edit operation by calculating the *edit distance*, weighted by the relative probability of each operation (Brill and Moore, 2000). The edit distance, also called *minimum string distance* or *Levenshtein distance*, between two strings is the minimum number of operations required to convert one string into another using a closed set of operations (Levenshtein, 1966; Damerau, 1964). In the standard version of the algorithm, 3 operations are permitted: *inserting* a character, *deleting* a character, or *substituting* one character for another[5].

Several additional modifications to the original framework led to advances in spelling correction systems. In addition to allowing an arbitrary number of editing operations in their error model, Brill and Moore (2000) allowed edit operations that involved multiple letters in a single operation. For instance, consider replacing the sequence *ph* with the single character *f* (e.g., spelling "physiology" as "fysiology"). Using typical edit distance operations, this would be considered 2 edit operations, but as these sequences share a pronunciation, it may be more accurately modeled as a single error. To account for this, Brill and Moore (2000) explored all possible partitions of one or more characters and selected the partition that minimized the weighted edit distance between the word and the misspelling.

The decision by a user to transcribe "physiology" as "fysiology" is likely to be a cognitive error, not a typographical one (Kukich, 1992). That is, the user made this mistake because they did not know the correct spelling and attempted to spell it phonetically, not because their finger accidentally touched the wrong key. Cognitive errors often result in misspellings that have similar pronunciation to the word that the user was attempting to type. To

[5]Some early work (e.g., Kernighan et al. (1990)) also allowed swapping two adjacent characters to count as a single operation.

account for this, work by Toutanova and Moore (2002) expanded the Brill and Moore (2000) spell checking model to take pronunciation into account. To do this they performed the error correction procedure on phonetic representations of words, rather than their standard spellings.

Once a spell checking mechanism is in place, autocorrection systems still must decide which cases to automatically correct. Although one option would be to automatically correct every word that does not appear in the internal dictionary, this approach is likely to be overzealous. Ahmad and Kondrak (2005) analyzed out of vocabulary terms from search query logs and found that only 9% were misspellings; others were proper names, neologisms, medical terms and foreign words. Even a carefully constructed dictionary will miss some of these terms. As changing words that were correct is likely to be a more damaging mistake than failing to change a term that is incorrect, autocorrection systems must be careful to not over-correct. Therefore, one option is for autocorrection systems to only correct if the system has a high confidence that a mistake has been made (Whitelaw et al., 2009).

Some systems, such as word processing, may employ both automatic correction and standard spelling correction. In these cases, out of vocabulary words can either be flagged as incorrect and left for users to initiate spelling correction or automatically fixed by autocorrection. Although having user initiated spelling correction as a fallback allows autocorrection to be more conservative, it may be of little use in mobile applications. The limited text entry systems of mobile devices can make manually correcting input slow and difficult, and users often do not proofread input in typical mobile device use cases such as text messaging. Because of this, there are incentives for mobile device autocorrection mechanisms to be more aggressive.

2.4 Evaluation Metrics

Measuring text input performance can be seen most generally as measuring two properties: 1) text input speed and 2) text input accuracy. These 2 properties are often at odds with

one another, as inputting text more swiftly often has the potential to increase error rate as well. Several methods have been proposed to measure speed and accuracy, as well as other properties of input efficiency. This discussion will only cover common metrics that are relevant to this work. For a more complete overview of metrics that have been proposed, see (Wobbrock, 2007).

2.4.1 Measuring Text Input Speed

Although there have been several different methods proposed for measuring text input speed, they are all based on the same simple idea of measuring the number of characters input per unit time. We can compute the input rate in *characters per second* (CPS) as follows:

$$CPS = \frac{(|T| - 1)}{S} \tag{2.8}$$

where T is the transcribed string and S is the total input time of the string in seconds. We subtract one from the character count to account for the fact that the recorded time starts when the first key is pressed, which does not include the time needed to locate and select the key. This metric is often reported in the derived form of *words per minute* (WPM):

$$WPM = \frac{60CPS}{5} \tag{2.9}$$

It should be apparent that we multiply by 60 to convert seconds into minutes, but the reason for dividing by 5 may be less clear. This is based on the convention that we should count every 5 characters in the input string as a word (Yamada, 1980). Having a standardized notion of word length allows for easier comparison across tasks where true word length may vary.

2.4.2 Measuring Text Input Accuracy

Although somewhat more nuanced that calculating input speed, calculating input accuracy is still fairly straightforward. Nonetheless, several different methods have been proposed. The metrics outlined here calculate accuracy via error rate, and thus lower values are most desirable. One simple measure of error rate is *keystrokes per character* (KSPC) (Soukoreff and MacKenzie, 2001):

$$KSPC = \frac{|IS|}{|T|} \tag{2.10}$$

where T is the transcribed string and IS is the entire input stream. The input stream contains every keystroke typed by a user, including mistaken characters and backspace keystrokes that do not appear in the final transcribed string. Thus, each time a user corrects their input, the length of the input stream increases. Note that if no errors were produced, for standard input on a qwerty keyboard the length of the input stream will equal the length of the transcribed text, giving us $KSPC = 1$.

It should be noted that there are 2 different notions of keystrokes per character that appear in the literature. The one reported here is a *dependent* measure (Soukoreff and MacKenzie, 2001). Implicit in the measure is the suggestion that if no mistakes are made, one keystoke produces one intended character. However, for certain input methods this may not be the case. For instance, a keyboard with only a few keys, such as the 12 key keyboards common on many mobile phones, may require a user to hit a key several times to input the intended character. Thus the number of keystrokes required may be a function of the input method independent of the input of the user. This different notion of keystrokes per second is referred to as a *characteristic* measure (MacKenzie, 2002). Although our discussion here will only consider the dependent measure, it is important to not conflate the two.

There are a few drawbacks to KSPC as a performance measure. One potential issue is that it is somewhat fickle in regards to the number of errors produced and how they

are corrected. Consider these two input streams representing two attempts to correct a misspelling of the word *people*:

1. *peeple <<<< ople*

2. *pee < ople*

In this example, the token < represents the backspace key. In both cases, the user starts to misspell the word as *peeple*, an instance in which only a single character is incorrect. However, in case 2 he realizes his mistake immediately after typing the incorrect character, whereas in case 1 he realizes it only after completing the word, causing him to backspace through several correct characters to fix the error. Although both instances only contain one typing error, example 1 has a much worse KSPC ($KSPC = 2.33$) than example 2 ($KSPC = 1.33$).

Another drawback of the $KSPC$ performance metric is that it may not capture all sources of error. In particular, if a user does not correct a mistake, $KSPC$ may not detect the input as erroneous. To handle cases in which errors are present in the final string transcribed by the user, we need an additional metric, *Minimum String Distance* (MSD) error rate.

The Minimum String Distance algorithm utilizes the edit distance algorithm used by spelling correction systems. If the text that a user intended to type is known, minimum string distance can be used to detect typing errors. Consider the following example:

- *Intended string*: "Call me Ishmael."

- *Typed string*: "Cal me Ishmiel."

Each typing error in the example can be corrected with only only edit operation. If we wish to transform the typed string into the intended string, we need to insert the character l and substitute the i for an a, giving us a minimum string distance of 2. Because the minimum string distance measure is symmetric ($MSD(A, B) = MSD(B, A)$), it does not

matter which string we decide to permute. We can now convert MSD into an error rate measure[6] (Soukoreff and MacKenzie, 2001):

$$MSDErrorRate = \frac{MSD(A,B)}{max(|A|,|B|)} \tag{2.11}$$

where A and B are the intended string and the transcribed string, respectively. MSD error rate gives an indication of the number of uncorrected errors in the string, but does not measure corrected errors. Thus, to get an understanding of the total error rate in instances in which both corrected and uncorrected errors may be present it is necessary to look at both MSD error rate and KSPC. Unfortunately, these two measures are not easily combined to give an overall measure of error rate. They do not measure errors on the same scale; Values for MSD error rate range from 0 to 1, while values for KSPC range from 1 to infinity.

2.5 System Self-Assessment

One of the main goals of this work is to attempt to examine and solve the online data collection problem for mobile device text input aids, in order to enable these aids to be dynamically personalized. As collecting data online with the intent of improving performance often requires the system to understand its current performance, many of the methods introduced in this work (particularly those in Chapter 5) rely on system self-assessment. Now that we have introduced the relevent aids, we will more closely examine what it means for a system to assess its own performance.

The idea behind self-assessment is simple. When a system makes a decision it has an

[6]The form presented here is the original formulation of MSD error rate from Soukoreff and MacKenzie (2001). In later work (MacKenzie and Soukoreff, 2002) the authors pointed out that this formulation does not handle all cases well, and suggested that the denominator should be the mean of a set of alignment strings that represent possible causes of the error. As this value is always less than or equal to $max(|A|,|B|)$ the corrected version is guaranteed to give a lower error rate. However, the formulation given is still the most widely used in the relevant work.

effect on the world. In some cases the effect produced by the system will be desirable (based on some predefined metric) and in some cases it will not be. As the system wishes to produce desirable outcomes, we can say that all actions taken by the system that produce desirable outcomes lead the system to a success state, and all other actions lead it to a failure state. However, in many systems the final state will be partially or entirely unobservable by the system, so it will not know whether its action has led to success or failure. Self-assessment can thus be thought of as the task of a system attempting to discover the success conditions and details of the state it is in after an action has been taken.

What it means for the system to assess its own performance depends strongly on what the system intends to accomplish. Because of this, system self-assessment procedures can look very different across disciplines. As such, we will restrict our discussion here to previous attempts at self-assessment of natural language processing systems.

One of the most common applications of self-assessment in natural language processing systems is in the field of spoken dialogue systems. Although the term spoken dialogue system could generally be used to refer to any system that utilizes spoken dialogue, in practice self-assessment has only been relevent in the context of task-oriented scenarios in which a human user interacts with a system via spoken dialogue. Generally, these scenarios have the human user attempting to use the system to complete a task. In these cases, the user knows what the goal is and the system has the means to provide it. The task is generally centered around the user attempting to convey their intent to the system so that it can take the necessary steps towards task completion.

One typical early example of such a system is the "How May I Help You?" (*HMIHY*) system developed by AT&T (Gorin et al., 1997). The HMIHY system was designed to be an open-ended call routing system. Users call the system hoping to perform a certain task, such as discovering the cost of a long distance phone call or to charge a call to a credit card. The system opens the dialog by prompting the user with the phrase "How may I help you?". The user then communicates their goal to the system via natural spoken dialogue

input. The system attempts to understand the user's dialogue contribution and continues to prompt the user for additional information or clarification until the task is achieved.

Although an unproblematic dialogue might proceed in the above manner, there are several instances in which the system may fail. For instance, the speech recognizer may produce incorrect output, making it difficult for the system to understand the system's intent. Alternatively, the natural language understanding unit in the system may misinterpret speech that was well understood by the speech recognizer. There are multiple points of potential failure involved in producing a single dialogue turn. Given this, it is likely that problematic situations will arise in many dialogues. Because problematic situations can aggravate the user and lead to task failure, the system would benefit from identifying these cases.

Problematic situation detection in the HMIHY system is modeled as a classification problem (Walker et al., 2000). Using a dataset of about 4700 manually annotated dialogues, a rule-based classifier was trained to differentiate between situations that were deemed to be problematic (task not completed successfully) or unproblematic. About 36% of the dialogues in the corpus were labeled as problematic. Five feature sources were used to train the classifier:

1. *Acoustic/ASR Features* captured aspects of the speech recognition output, such the number of words recognized or the input modality expected by the recognizer.

2. *NLU features* captured information provided by the natural language understanding module, such as confidence measures corresponding to each of the tasks that the user could be trying to perform.

3. *Dialogue manager features* captured features related to dialogue management such as the last prompt produced by the system and whether or not it was a reprompt.

4. *Hand-labeled features* included human produced transcripts of each utterance and demographic information such as the age and gender of the user.

5. *Whole dialogue features* captured relevant information that spanned the entire dialogue, such as the total number of reprompts over the course of the dialogue.

Using this feature set, the HMIHY system was able to produce accuracy of around 92% using the entire dialogue. Using automatic features only produced accuracy of around 87%. Since the system wishes to predict problematic situations early so that it can adapt to it, it is more meaningful to examine the performance of the system when only the first few exchanges have been observed. Using information from only the first 2 exchanges in the dialogue the HMIHY system produced accuracy of around 80%.

The HMIHY feature set is fairly indicative of those features used by other systems, with 2 notable omissions: prosody and dialogue acts. Prosodic features can be useful because problematic dialogues may cause an emotional response in the user such as annoyance or anger which may be expressed prosodically, or because the user may attempt to enunciate more slowly or clearly (Swerts et al., 2000; Litman et al., 2006). Because the user is providing a speech signal to the system, prosodic features are readily available. Dialogue act features are useful because they give the system a better idea of what the user is trying to achieve with each utterance (Prasad and Walker, 2002; Hara et al., 2010). Although harder to get than prosodic features, dialogue acts can be identified reasonable well in strict task-oriented dialogues such as those seen in many spoken dialogue systems.

Spoken dialogue systems are not the only application domain in NLP that can benefit from self-assessment techniques. Another example is conversational question answering systems, in which the system attempts to interactively answer questions posed by the user (Chai et al., 2006a). Identifying problematic situations in these cases can be more difficult, as conversational question answering systems lack some of the feature sources of spoken dialogue systems, such as prosody. Additionally, since the user may form all of their dialogue turns as information seeking questions, they are likely to give little to no indication that a mistake has occurred. Work by Chai et al. (2006b) used a variety of similarity metrics to compare user queries to system answers and previous queries as a means to find cases where

user intent was not fully met, achieving classification accuracy of around 74%.

2.6 Personalization

This section gives a quick overview of personalization attempts in the literature. Like self-assessment, looking at personalization in general is overly broad and not particularly relevant to the work presented here. As such, this section will give a brief overview of personalization concepts and work, focusing on cases in the natural language processing literature. As we will see, what it means to personalize a system is extraordinarily application specific.

This work has used and will continue to use the term *personalization* to describe the process of tailoring a system's behavior to an individual user. However, personalization is similar to (and often conflated with) the related concept of *user modeling*. User modeling is the task of understanding the behaviors and preferences of an individual user that interacts with the system. As one might imagine, there are many different types of user differences that could be modeled. Work in this area ranges from such seemingly unrelated tasks as recommending movies to users based on previous preferences (Resnick et al., 1994; Konstan and Riedl, 2012), tailoring tutoring systems to different learning styles (Michaud and McCoy, 1999), and discovering user-specific information retrieval habits (Brajnik et al., 1987; Ghorab et al., 2012). One important task in user modeling is constructing the user model itself, and in this respect the field also has considerable overlap with self-assessment work. Similarly, user modeling can be thought of as analogous to the initial information gathering and representation stages of personalization (Gauch et al., 2007), with little focus on the final implementation and execution.

Early work on user modeling and personalization for natural language processing applications focused primarily on one of two areas, content planning and plan recognition (Zukerman and Litman, 2001). Content planning is done primarily for the purposes of natural language generation. Paris (1989) modeled user domain knowledge to modify the length and content of responses from a question answering system which explained the content of patents.

Zukerman and Mcconachy (1993) modeled information about a users beliefs, attitudes, and the inference rules they were likely to apply to modify the output of a system designed to produce descriptions and similes. Work in the medical domain used information gathered by the physician and by questionnaires to tailor system responses to individual user symptoms (Carenini and Moore, 1994). All of this work assumed that the user model was given, and did not address the problem of inferring it from the user-system interaction.

Early work on user modeling and personalization in plan recognition was focused on natural language understanding tasks. Plan recognition is the task of understanding the goals of a user or how they intend to execute these goals and is thus a task in which user modeling is intuitively useful (Carberry, 2000). Systems are designed to have a list of possible user goals and actions. Using this the system attempts to place the user's current action within an action sequence that eventually leads to a goal state (Allen and Perrault, 1986). As there are likely to be several viable hypotheses for the user's intended plan, the system must attempt to narrow the hypothesis space to only the most likely. Work by Wu (1991) and Cohen et al. (1991) attempted to actively understand the user's plan by explicitly querying the user for plan information. Ardissono and Sestero (1996) used stereotypical information about the user and information from previous user actions to weed out unlikely plans.

Much of these early attempts at personalization were in highly structured, rule-based systems where user knowledge was taken as a given. Although these systems were able to produce reasonable results in the fairly narrow domains they were designed for, they were brittle to the inclusion of new domains or knowledge and did not address how user information could be obtained. Around the turn of the century, the rise of statistical techniques and the availability of large data collections changed the landscape for user modeling and personalization approaches (Webb et al., 2001). Since this time, the use of techniques such as collaborative filtering to relate the behaviors of a newly encountered user to those of previously seen users has become the norm. While these techniques can unquestionably be powerful, they can raise privacy issues (Toch et al., 2012) and are not applicable to appli-

cations in which a large amount of data about other users is unavailable or too difficult to consult in real time. In particular, the approaches that will be discussed in this work involve what is sometimes referred to as "client-side" personalization (Mulligan and Schwartz, 2000), in which we only have access to the limited information on the client device, which does not have information about other users.

CHAPTER 3

Adaptation Strategies for Character-Level Text Entry Aids

The next two chapters outline work aimed at improving character-level text aids via adaptation and personalization. This chapter focuses on a critical aspect of this process, developing strategies that will allow systems to collect the data necessary to build adapted models. In particular, the work outlined here focuses on key-target resizing (see Section 2.3.1). There are several aspects of key-target resizing which make it an ideal use case. First, it is non-obtrusive. One of the problems encountered by many text entry aids is that they are distracting or annoying to the user (Himberg et al., 2003; Paek et al., 2010). Because key-target resizing does its work covertly, it does not run the risk of drawing the user's ire simply because it is different from the interaction paradigm they are accustomed to.

Second, key-target resizing is already being implemented on commercially available mobile devices, such as the Apple iPhone[1]. This point is critical because many theoretically sound aids never make it beyond the level of research prototype. Although many different text entry aids have been proposed, a comparably few of them are adapted for use on generally available devices. The fact that key-target resizing has seen use in commercial systems suggests that it may have reached the point where general models have reached a level of performance thought of as "good enough" for large-scale adaption. It is cases such as this that are most relevent in testing the merits of personalization.

Finally, related studies (Himberg et al., 2003; Rudchenko et al., 2011) have hinted at the potential of personalized key-target resizing models. These studies perform laboratory-based experiments that confirm that personalized models can in fact produce improved performance

[1] http://www.apple.com/iphone/

over generalized models. However, they fail to address one of the most critical aspects of building a personalized model: how do we get the data we need to train a model for each user? The work outlined in this chapter addresses this heretofore unexamined problem.

As outlined in Section 2.3.1, there are two different models that contribute to key-target resizing, and thus two models that are candidates for personalization. The first model, that of the language, would be difficult to personalize. The large number of symbols in the language means that n-gram language models need a large amount of data to produce an accurate approximation. For instance, Gunawardana et al. (2010) used 8.5 million characters to build their key-target resizing language model. Even if a personalized model could be made with 1% of this data, it would mean that we would have to wait until the user typed 85,000 characters (about 17,000 words) before we could build our model. This means that personalization would be impractical in all but the most extreme cases.

The second model, the touch model, is a much more promising candidate for personalization. Most critically, effective touch models can be trained on smaller amounts of data. Rudchenko et al. (2011) showed that personalized models built on 2500 characters (about 500 words) could outperform general models trained on 5 times as much data. For comparison, this is about 50 text messages of average length, which means that the average text messager would produce enough training data in about a week with texting alone (based on statistics of text message use provided by Battestini et al. (2010)). This should be quick enough to make the gains produced by personalization worth the short wait.

The next section outlines the motivation behind building personalized key-target resizing models. Section 3.2 gives the relevant hypotheses and an outline of the work done to test them. The data collection methodology is discussed in Section 3.3. Section 3.4 outlines several online data collection strategies and how they are evaluated. This chapter ends by evaluating each of the proposed strategies in their ability to collect adaptation data quickly and accurately, setting the stage for the next chapter which will examine personalized models built from these methods.

3.1 Motivation

It is hard to overstate the advances in natural language processing and related areas that have come from building large, generalized models from aggregate data. The use of large quantities of data from many different writers or speakers has been able to facilitate the building of statistical n-gram language models that give good approximations of the likelihood of words and phrases appearing in the language. But at the same time, the generality of these models can be their pitfall. A single general language model trained on many different styles of language (e.g., casual speech, formal speech, formal writing, text messaging, web search, e-mail, URLs, etc.) is unlikely to handle each individual case as well as individual models trained for just that type of input. Personalization is an expansion of this idea: instead of just cutting up our models to handle different writing styles, we slice them one step lower to make models that are robust to differences in vocabulary and writing style between individual users.

However, as we have said, language models are impractically large to personalize at present. So we will instead focus solely on personalization of the touch model. The purpose of this section is to convince the reader that personalized touch models should be considered not just desirable, but necessary for accurate key-target resizing. To do so, several factors will be outlined that are likely to affect the touch points input by a user. In each case we will see that while general models are blind to these differences, personalized models are able to account for them in full or in part.

Differences In Perceived Input Point. As discussed in Section 2.2, some of the noise observed in touch input accuracy may arise from differences in the cognitive model applied by the user when attempting to select a key. Some users may believe that the point that they select is near the tip of their finger as it touches the screen, while others may imagine it to be near the center. For a given user, the touch model will be able to learn this model from the observed data, and adjust accordingly. However, this is difficult for a general model. Since

the touch model is estimated by a bivariate Gaussian distribution, it will be unimodal, but because the data is being produced by several different cognitive models, it will not be.

A similar problem is seen with the variance of the model. Holz and Baudisch (2010) showed that the angle at which a target was approached had differing effects on user's cognitive models. For some, the acquired point changed significantly with the angle of approach, while others were nearly unaffected by it. If we had knowledge of the angle, we could learn to correct for it. Unfortunately, because we have no known way to determine the angle of approach in current generation mobile devices, this difference will be detectable by the system only as a difference in variance between one user and another. Thankfully, with personalization we are able to build separate models for users who exhibit differences in variance that are attributable to angle of approach.

Physical Differences Between Users. Users come in all shapes and sizes, and the size of the user can have an effect on their typing behavior. Differences in the size of the user's hands in particular may lead to behavioral differences. Users with larger hands may have higher error rates because their fingers register a larger area of the screen while typing. As above, these higher variances can be observed by personalized models, but not easily incorporated into a general model. The size of one's hand may also affect the style in which they hold the device and, relatedly, the distance they must reach to type a key. These differences affect the angle at which they strike the keys, affecting the distribution of their touch points.

Other physical variations between users may also have an effect on their typing behavior. For instance, differences in arm length or vision may change the distance and angle at which users hold the device. Differences in manual dexterity may affect input speed and accuracy.

Typing Style. Users may vary in both their physical and cognitive typing styles. Physical differences involve which digits they choose to type with. For instance, a user may choose to input text with one or two thumbs, or one or several fingers. The wider thumbs may be more inaccurate than the more slender fingers. Additionally, the choice of one vs. two thumb

input will affect the angle at which users approach the keys (e.g., consider reaching for a key on the far left side of the keyboard with the right thumb vs. the left thumb). If a user switches frequently between styles, personalization will not be able to help their accuracy, nor will it hinder it. However, personalized models will be better able to adapt to a user who predominantly uses one style.

Cognitive differences between users involve how careful they are when typing. Timid typers who carefully select each letter may have lower variances than fast typers who are more prone to accidentally type outside of the bounds of an intended letter. As before, we can only expect personalization to better model this difference for certain users, those that do not display significant intra-user variation (i.e., users who do not frequently change how carefully they type).

On The Fly Adaptation. Unlike static general models, learning a personalized model on the fly allows the device to adapt to changes in user behavior. For instance, consider a user who initially starts out typing on their mobile device using several different physical input styles, but eventually settles on a preferred style. A continually updating personalized model will adjust its behavior accordingly to adapt to the new style, while a static general model will not.

Differences Between Devices. Although not strictly related to the user, differences the arise from properties of different mobile devices can also be handled via personalization. Previous works on key-target resizing have restricted their studies to only examine a single device. However, this potentially misses significant differences that can arise from differences in device. Although this could be addressed by training device-specific general models, this would require a new laboratory experiment to gain training data with each new device. This is also complicated by the fact that the soft keyboard is not always developed by the hardware manufacturer. For instance, the Android operating system[2], which runs on devices produced by several different hardware manufacturers, provides a built-in soft keyboard.

[2]http://www.android.com/

One of the most straightforward device differences is in the dimensions. The dimensions of the device affect how users hold it, and as a result, how they type on it. Notably, the dimensions of the device may not even be fully under the control of the manufacturer if a user chooses to place the device in one of several commercially available cases. Although these cases may only shift a user's grip by a few millimeters, virtual keyboard buttons are small enough on mobile devices that this shift could be significant. The uncertainty about whether a user may augment their devices dimensions in this way greatly complicates the production of device-specific general models, but is easily handled by personalized systems that continuously adapt online.

Touchscreen differences between devices may also influence user's typing behavior. Different devices may vary in the size and resolution of the screen, meaning touch models learned on one device may not be directly applicable to other devices. The touchscreen technology may also vary (e.g., resistive vs. capacitive) which may lead the hardware to interpret similar presses differently (Lee and Zhai, 2009).

Although we have made the distinction between *device differences* and *user differences*, the effects of the device and of the user are not so easily decoupled. A user's decision about how they hold a device is likely to affect their typing behavior, but this decision is influenced by both user characteristics such as hand size and device characteristics such as dimensions. Thankfully, no decoupling is needed; a personalized model trained on a particular device for a particular user adapts to the quirks inherent in the interaction of the user and the device. As such, we should not think of personalized models as adapting to a particular user, but to a particular *user-device pair*.

3.2 Touch Model Training

To understand the inherent difficulty in training a personalized model, we must take a closer look at the training process. We wish to build a touch model such that, for any touch point (x, y), we can calculate the probability that this touch event was meant to select any

given key on the keyboard. To do this, we need to estimate the distribution of touch events for every key separately. For instance, to calculate the distribution of touch points for the character e we must collect a data set of touch points such that the user's *intent* was to select that character. Once we have collected enough of these points, we can estimate the distribution, which we assume to can be modeled reasonably well by a known parametrized distribution, the bivariate Gaussian.

So in order to train a touch model, we need each data point in our training data to have, at minimum, 2 things: the (x, y) coordinates of the touch point and the key which the user intended to hit. Now consider what information the system typically has access to when a user types on the keyboard. It has access to the (x, y) touch point that the user selected. It may also have access to additional characteristics of the touch event, such as the approximate size of the touch and how much pressure was applied. It knows which key it registers as selected by the user, via whatever process (key-target resizing or otherwise) it uses to determine this. It does not know, however, whether or not the key it selected was actually the key that the user *intended* to hit, and therein lies the problem. In order for the system to build a touch model it must know which key a user was actually trying to hit for a given touch point, but under normal circumstances it does not know this.

Laboratory-based experiments have attempted to get around this by following the text copying experimental paradigm commonly used to compare keyboard layouts (See Section 2.3.1). By dictating the text that the user must write, the experimenters also dictate their intent. This allows the experimenters to associate every touch event, even those that select the incorrect key, with an intended key. In this way the carefully constrained text copying method can be used to build generalized touch models. Personalized models can also be built for the few people who participate in the study.

Unfortunately, the text copying paradigm is not appropriate for building personalized models for most of the user base. Having every user of a mobile device come in to participate in a laboratory experiment to collect personalized data is certainly not a feasible solution.

Although deploying a text copying game such as the one developed by Rudchenko et al. (2011) may be able to provide personalized models for a larger portion of the user base, it is still likely to reach only a small percentage of the overall user population. If we wish to build personalized models for every user, we must develop a new data collection methodology.

One possible solution, similar to that used by Rudchenko et al. (2011), is to somehow encourage or cajole users into providing data in a structured format where their intent is known. However, this method has several drawbacks. First, collecting enough data to build a personalized model takes time. Dictating that a user must spend several hours interacting with a text input game before they are able to use their device for text entry tasks is not a reasonable solution. Second, such a constrained methodology does not allow the personalized model to be continuously updated, losing one of its premier advantages over generalized models.

The preferred data collection methodology should be online. After all, users engage in text entry tasks frequently, so to ignore online data collection is to ignore a plethora of useful data. In order to utilize this data we must answer the core question "how do we determine the intent of the user's key presses during natural, online text entry?". Answering this question is a critical aspect of online data collection, which itself is a critical aspect of adaptation and personalization.

There are two main stages to this process. In the first, we must devise a new experimental data collection methodology that better simulates true user behavior and accounts for the issues inherent in online data collection. This methodology must be formulated such that it it is externally valid with regard to natural online data collection, while still holding enough variables constant to be internally valid. This methodology is outlined in the next section.

The second stage of the process is to build models to assess the fidelity of incoming data points. This involves attempting to address such questions as: "for a given touch point, can we determine the key which the user intended to hit?" and "can we determine when we are too uncertain about the intent of a touch event to include it in our data?". Models and

methods for addressing these questions are given in Section 3.4.

3.3 Data Collection

This section discusses the decisions that allowed for the collection of a corpus of key-target resizing data that could be used to examine the issues of interest (online data collection strategies, personalization, adaptation). It should be noted that "data collection" in this context refers to obtaining the data necessary to perfrom the laboratory experiments to empirically examine the issues of interest; this section does not discuss the online data collection strategies used to collect data to build personalized models.

There were several advantages to the text copying methodology for keyboard layout experiments. These include:

- Constraining as many variables as possible was necessary to compare separate input rounds with different keyboard layouts.
- Specifying what users typed removed variance in thinking time between trials, making entry speed comparisons easier.
- Constraining how users were able to edit the text made error calculations easier.
- Specifying what users typed minimized certain types of editing operations.
- Specifying what users typed allowed the experimenters to calculate error rates.

While some of these advantages are still relevant to key-target resizing research, a few of them can no longer be seen as advantages. Tightly constrained experimental parameters made sense when comparing across sessions. However, the simulation style introduced by Gunawardana et al. (2010) allows for multiple key-target resizing methods to be tested on the same data. Because of this, there is no need to constrain variables for between-session comparison.

Similarly, the effect of thinking time is no longer significant. When comparing keyboard layouts, an accurate measure of input speed is important. Allowing users to write whatever they wanted would introduce variance between trials based on how long users spent mentally constructing their entries. However, evaluating on the same data means the simulation style of evaluation for key-target resizing allows us to hold input speed constant and examine what we are most interested in, error rate. Therefore, removing thinking time can no longer be seen as advantageous.

The other three advantages are still relevant to key-target resizing work. Most work that used the text copying paradigm allowed users to correct errors using only the backspace key (Soukoreff and MacKenzie, 2003; Wobbrock and Myers, 2006), although some allowed no error correction whatsoever (MacKenzie and Zhang, 1999). Although it would be possible to calculate error rate while including edit operations such as text selection and deletion or text pasting, these mechanisms do complicate things. Although allowing no text editing whatsoever is unrealistic and draconian, limiting users to just the backspace is a reasonable rule to enforce experimental consistency. It should be noted, however, that in this case contraining error correction too heavily is potentially detrimental, because we need to use the cues from error correction in order to collect data online (further explored in Section 3.4).

Similarly, specifying what users must type did allow for some nice experimental controls. In freeform text entry, users may edit the text they input for a variety of reasons. They may delete a sentence or word that they have written not because they have made a mistake, but because they have decided to rephrase it. Additionally, in conversational text input such as text messages or chat, users may decide to change a partially written contribution to address new incoming dialogue from the other participant. By constraining users to text copying we effectively remove these editing types, ensuring that any editing we see is intended for error correction.

Knowing user intent is necessary for error rate calculation. Specifying what users must

type dictated the users' intent for them, allowing experimenters to easily calculate error rate. Even though in typical usage scenarios user intent is unknown, an unconstrained text input methodology where user's intent is not dictated must be designed so that gold standard user intent is obtainable. Without this we cannot compute performance for key-target resizing algorithms or online data collection strategies.

As we can see, while the text copying methodology retains some of its advantages when applied to key-target resizing, several of them no longer apply. We should also acknowledge that this methodology has several drawbacks, including:

- The typical use case for mobile device text entry does not involve text copying. The typing style employed when copying text in laboratory settings is unlikely to reflect typical usage behavior.
- Removing inherent aspects of unconstrained text input such as thinking time may hurt external validity.
- Constrained setups where intent is known do not accurately reflect real use cases where it is not known.
- Design does not allow for easy comparison of different hardware devices.

A new methodology should address as many of these drawbacks as possible, while still attempting to retain the advantages imparted by the old methodology. To this end, this work introduces the *semi-guided chat input methodology.*

The semi-guided chat input methodology collects data via instant message chat. For each session, two users participate in the study simultaneously. Each user is given a mobile device and is told to communicate with the other user via a simple chat program provided to them. Users are told that they may chat about whatever they wish, but that the experimenter will provide them with topics if they are unable to keep a constant conversation going. The experimenter monitors the conversation and interjects with a new topic whenever the

conversation lulls, to ensure that the participants chat continuously for the duration of the experiment.

Several controls are implemented to enforce internal and external validity. The two participants engaged in each study session must be acquainted and able to converse freely with one another prior to the experiment. This contraint helps to better mimic typical usage behavior and also attempts to minimize lulls in the conversation. Although users are given a choice of their preferred physical typing style, to maintain consistency they are asked to use the same style throughout the experiment.

Many of the constraints are imposed by the chat program. Mobile devices commonly allow users to switch between landscape and portrait views, which affects the size of the keyboard, and as a result, the touch points. This necessitates that separate models be built for the landspace and portrait keyboards. This process is the same for each keyboard, and as such examining both keyboards would require twice as much data to be collected in the course of a limited experiment. To control for this, the chat program only allows input in portrait mode.

As in the text copying paradigm, editing operations are simplified to allow use of the backspace key only. Cursor movement, text selection and text cutting, copying, and pasting are all disabled. Similarly, the chat program does not employ any other text entry aids (e.g., autocompletion, tactile feedback). Although the inclusion of these features is possible, the simplification in interpretation gained from their removal is beneficial in ensuring internal validity.

As an experimental testbed for text entry aids, chat has a number of advantages. Chat is informal and conversational, similar to text messaging (Ling and Baron, 2007) and twitter (Ritter et al., 2010), both of which are commonly used mobile phone applications. Because it does not dictate what users should write it introduces several previously ignored aspects of unconstrained text entry, such as abbreviations and out-of-vocabulary words. It also helps to manage the tradeoff between input speed and accuracy. In the text copying methodology,

users were told to copy the text "as quickly and accurately as possible", but the experimental setup did nothing else to attempt to have users balance speed and accuracy in a realistic way. In the chat methodlogy, this tradeoff is handled naturally by two competing user desires. Because it is important for conversational contributions to be timely, users have an incentive to input and commit their contributions fast. Conversely, users wish to be understood, and contributions with too many spelling mistakes are less likely to be clear. Knowing these two facts, users of chat form each dialogue contribution such that it is as fast and understandable as possible.

3.3.1 Determining the Gold Standard

In order to evaluate how well an online data collection strategy is able to divine user intent, we must first know the true, "gold standard" values for user intent. Because the semi-guided chat input methodology does not dictate intent, we do not have ready access to this information. This section discusses procedures for determining the gold standard intent during data collection.

Although we have primarily been discussing typing errors as though they constituted one single type of error, this is not in fact the case. Errors differ in the type of mistake that caused them; they may be typographic or cognitive (Kukich, 1992). Typographic errors occur when the user attempts to select the correct key but their touch point is inaccurate and does not land in the area that the system associates with that key. These are the errors that we hope to reduce by improving key-target resizing touch models. Conversely, cognitive errors occur when the user misspells a word because they mistakenly believe that this is the way the word is correctly spelled. Cognitive errors must be handled differently than typographic errors, because the user's intent is different.

Consider an example in which a user misspells the word *divide* as "duvide". If the user attempted to hit the character i but his finger accidentally landed on the character u, this is considered a typographic error. The intent of the user was to hit the the character i, and

the system has misinterpreted it. Conversely, if the user hit the character *u* because they believed that this was the correct way to spell *divide*, then this was a cognitive error. In this case the user's intent was to hit the character *u*, and thus the system has interpreted their key press correctly. In general, cognitive errors should not be considered as errors for the purposes of training and evaluating a key-target resizing model, as the intent of the user aligns with the system's interpretation.

Keeping this distinction in mind, we must also attend to differences in how users handle errors after they have been made. Users may discover and correct the errors they make before committing their contribution, or they may not. This distinction between corrected and uncorrected errors is well established in the literature and was the motivation for the decision to disallow all forms of error correction in some early studies (Matias et al., 1996; MacKenzie and Zhang, 1999). Both corrected and uncorrected errors must be considered when determining the gold standard. We will look at each case separately.

Corrected Errors. Corrected errors can be readily detected in the input stream by looking for editing operations. Because the outlined methodology disallows most editing operations, we only need to look for instances in which the backspace key is pressed. Editing operations often, but not always, signal that the user is correcting erroneous input.

In building the gold standard, corrected errors can be manually classified by human annotators. These annotators use their own judgment to determine whether an error is cognitive or typographical, and therefore what the intended key was. Although it may initially seem that this task is daunting, cognitive and typographic errors have certain telltale traits that can help annotators identify them. For typographic errors, the incorrectly selected key must be relatively close to the intended key as the user is unlikely to be so inaccurate in their typing that their key press lands several characters away from its intended position. In the case of cognitive errors, the misspelled term and the intended term are likely to be phonetically similar (Toutanova and Moore, 2002). In a relatively small number of cases, such as the *divide/duvide* example above, this distinction may not be clear. Annotators are

asked to mark these cases as ambiguous. Ambiguous cases are not be considered during evaluation.

Annotators must also identify cases in which editing operations are used for purposes other than error correction. Consider this input stream:

- *I don't think that is correct<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<that's wrong*

In this instance, the user inputs an error-free contribution, but then decides to rephrase it. This causes them to initiate an editing operation for reasons unrelated to error correction. Thankfully, the difference between the initial input and the final contribution is stark enough that these cases should be easily identified.

Each user's input stream was annotated by two separate annotators. Cases in which the two annotators do not agree were considered ambiguous and therefore not considered during evaluation. To judge the reliability of the annotations, inter-annotator agreement can be calculated using Cohen's κ statistic (Cohen, 1960):

$$\kappa = \frac{p_o - p_c}{1 - p_c} \tag{3.1}$$

where p_o is the proportion of observations in which the two annotators agree, and p_c is the proportion that they would be expected to agree by chance. The probability of chance agreement is estimated from the distribution of the classes in the observed data.

Inter-annotator agreement was very high on the dataset used in this study (Section 3.5.1), $\kappa = 0.995$. As the level of agreement expected and needed can vary substantially between domains and tasks, there is no universally agreed upon κ value that constitutes "good" agreement. However, in a few cases previous work has suggested certain thresholds that they consider to demarcate reasonable agreement. One of the most influential examples is Landis and Koch (1977), who suggest that $\kappa > 0.8$ should be considered near perfect agreement. This threshold may seem to be too low for our annotation task, as it is critical

that we have a reasonably non-ambiguous annotation schema. Nonetheless, our observed κ value is so high that it would be considered excellent agreement by any reasonable measure, which should assuage any fears that our evaluation results would be tainted by a poorly defined gold standard.

Uncorrected Errors. Uncorrected errors are potentially more difficult to classify, as they do not have the added information provided by editing operations. Additionally, while all corrected errors must have an accompanying edit operation, there is no clear marker that shows when an uncorrected error has occurred. Although uncorrected errors often result in words that are not in the vocabulary, not all do. This is complicated further by the fact that in informal input such as chat, where acronyms and abbreviations may be common, many out of vocabulary words are not erroneous. Thus, it is difficult for an outside annotator to know when uncorrected errors have occurred.

To account for this, study participants are asked to correct the errors in their transcript at the end of the experiment. Since the intent is correct on cognitive errors, participants are asked to only correct errors that were caused by mistyping. Of course, it is possible that after a lengthy experiment, participants no longer remember which cases were caused by typographic errors. To attempt to minimize this, a pilot study was carried out to attempt to access how long users can type and still retain the ability to correct their mistakes. Data from this study suggested that users could type for over an hour and still be able to confidently find and correct erroneous input, if asked to correct their mistakes directly after typing. Additionally, independent annotators examined those errors pointed out by users and confirmed that they could possibly be typographically produced (e.g., substitutions of characters far away from one another on the keyboard were considered unlikely to be typographic).

3.3.2 Setup Specifics

What has been outlined thus far is the general form of the semi-guided chat input methodology, which can be applied to any study of key-target resizing or similar text entry aids. It is purposely vague about certain design implementation details, as these can be varied between studies while still adhering to the general framework. This section gives specifics about these design decisions for the current study of online data collection.

In the current experimental setup, two participants were given mobile devices and asked to sit on opposite sides of an opaque median. Users are asked to only communicate via the mobile devices. The median helps to discourage other communication between participants, both verbal and non-verbal.

An experimenter sat to the side of the median so that they were able to observe both participants. During data collection, the experimenter's tasks were threefold: 1) they must monitor the conversation and interject with new topics when users are not conversing, 2) they must ensure that they experimental constraints are being followed (e.g., stopping participants from changing their physical typing style and from conversing without using the device), and 3) they must answer any questions that participants may have.

The experimenter must have a large and diverse set of topics prepared in order to ensure that the participants are able to converse for the entirety of the experiment. To account for this, the experimenter was provided with a set of 70 topics used in data collection for the SWITCHBOARD corpus (Godfrey et al., 1992)[3]. These topics cover a wide variety of issues, ensuring that the experimenter is always able to provide participants with topics that they are willing and able to converse about.

The chat program consists of a simple interface which displays the keyboard, a box for text entry, and the text of previous conversation turns. The chat program connects to the other

[3]Because SWITCHBOARD collected speech data from participants over the telephone, the participants are referred to "callers" in topic descriptions. For the experiment outlined here, the term "caller" was replaced by the term "person".

(a) Standard QWERTY (b) Symbols

Figure 3.1: Layouts of the standard QWERTY and symbols keyboards used in the data collection study.

device via a bluetooth connection. To best mimic a real chat program, the implementation is based on an open source bluetooth chat program[4].

Previous studies often restricted users to a keyboard with only the letter keys and a few punctuation keys. While this may have been a reasonable simplification when intent was dictated, in a more open input scenario it is more realistic to give users a larger number of possible input options which closely resemble the keyboards they would encounter on currently available devices. To account for this, the soft keyboard implementation used in this experiment was based on an open source soft keyboard implementation[5]. Some views of the keyboard are shown in Figure 3.1.

During data collection the keyboard was programmed to register keys using a static non-resizing algorithm. That is, the keyboard was set up such that the key dimensions shown to the user are the same as those that are associated with the key by the system. Although it may be desirable to use a system that already implements a key-target resizing algorithm built on a general model during data collection, this would require us to already have data to train a general model prior to the experiment, which we did not. Additionally, we may wish to use a static model because building a general model for every hardware device may not be practical, as outlined above.

[4]http://developer.android.com/resources/samples/BluetoothChat/index.html
[5]http://developer.android.com/resources/samples/SoftKeyboard/index.html

Figure 3.2: Mobile Devices used in the data collection study: HTC EVO (left) and Google Nexus S (right)

The system logs information about each key pressed by the user. For the purposes of this experiment, it may suffice to log only the (x, y) coordinates of the key press and what key the system registers as cooresponding to the key press. However, to benefit further analysis, several other items were logged, including the size and pressure of the key press, and the time at which the key press occurred.

Although it is hypothesized that differences between devices may effect the accuracy of general key-target resizing algorithms, this hypothesis has not been empirically validated previously. To examine this, we used two separate mobile devices during data collection, the HTC EVO[6] and the Google Nexus S[7]. These are both state-of-the-art devices which run the same operating system (Google Android) and have similar specs in terms of processing power, but which differ in the size of both the device and the touchscreen. The devices are

shown in Figure 3.2.

Previous work by Rudchenko et al. established that personalized models could be build from training datasets of about 2500 key presses per user. To ensure that enough data was collected for each user, the input rate of users in our pilot study was examined. This study suggested that 2500 key presses could be collected in about half an hour of continuous input. Because this estimation is based on a small pilot study, and because the variance in input rate between users is likely to be high, it was decided that collection sessions should be one hour long. Additionally, because we wish to examine data assessment strategies that will not retain all of the data seen as training data, we need more than the bare minimum amount of data.

3.4 Adaptation Strategies

We initially examined the utility of 3 simple methods designed to enable adaptation via online data collection (Baldwin and Chai, 2012b). Each of these methods considers different levels of processing to attempt to manage the trade off between the amount of data considered to be available for training and the amount of noise in the data.

Conservative Method. The conservative method makes very little inference about the fidelity of the data; it only includes data that the user explicitly provides fidelity judgments on. Unfortunately, users do not provide these judgments often. The only point at which user's fidelity judgments are overt is during editing operations. Consider the following input stream:

- *wring <<< ong*

In this example, the user mistypes "wrong" as "wring", and uses the backspace key to correct it. In doing so, the user provides us with eight touch points for training: five points

[6]http://www.htc.com/us/products/evo-sprint
[7]http://www.google.com/nexus/

from before the edit operation and three after. We can assume that the three letters typed after the edit operation were the intended characters for the three characters that were deleted. This gives us six points with their intended key. We see that the first attempt to type the character *o* was interpreted by the system as an attempt to type the letter *i*, and that the first attempts to type *n* and *g* were successful. A small amount of inference tells us that we can probably trust that the first two characters, *w* and *r*, were also correctly interpreted by the system. After all, if they were not, the user would have had no reason to stop the editing operation mid-word and would have instead changed the other incorrect characters.

Unfortunately, not all error correction operations are as clear as the above example. For instance, if the user accidentally omits a keypress or includes an additional unintended keypress, the intended keys input after the editing operation (*ong* in the example above) will not align with the keys they wish to correct (*ing* above). While it may be possible to examine all alignments and find the most likely one, these cases are often ambiguous, so this may result in a higher percentage of misrecognized characters. As the goal of the conservative method is to provide training data with as little noise as possible, these cases are simply disregarded. To accomplish this, the conservative method only considers an error correction operations to be trustworthy if it can find an alignment between the characters typed after the editing operation and those typed before that differ only be adjacent keys. For instance, the *wring/wrong* example given above would be captured because the strings only differ by a substitution of *o* for *i*, two keys that are found adjacent to one another on the QWERTY keyboards used in the experiments.

This constraint also helps the conservative method disregard two other potential sources of noise: cognitive errors and editing operations unrelated to error correction. As characters in cognitive errors should be phonetically but not necessarily spatially similar to the intended characters, the adjacency constaint should remove most cognitively generated errors. Additionally, as a rephrasal or restart is very unlikely to only differ from the previous input by

adjacent characters, these case are also effectively disregarded.

Because edit operations are undertaken on a comparatively small percentage of the input stream, the conservative method throws out the majority of the data. By doing so, it is trading recall, the percentage of points from the input stream that we include in training, for precision, the percentage of those characters that we return whose intent was correctly determined. By focusing on high precision, we minimize the amount of noise in our training data, which is critical for building accurate models. We do so at the expense of time, however. Because we only use a small percentage of the data, we must see a lot more data before a useful model can be built.

Character-Level Method. The character-level method is the converse of the conservative method, in that it is extremely liberal in deciding what data we can trust. This method assumes that the user is careful about what they type and is prone to proofreading their text input and correcting mistakes. If this is the case, then we can trust that all input that is not involved in an edit operation was interpreted correctly by the system. All those characters that were involved in an edit operation can be handled in the same manner as in the conservative method. In doing so, we are able to include every single character from the input stream in our training data, excluding those that are involved in editing operations that the conservative method does not capture (e.g., cognitive errors and restarts).

It may seem unreasonable to assume that all data not corrected by the user is good. Indeed, this style of data collection makes no attempt to remove uncorrected errors in the text, and is thus almost certain to add noise to the data. However, because previous work (Wobbrock and Myers, 2006) suggests that uncorrected errors are highly outnumbered by corrected errors, the amount of noise that is generated by such a method may be outweighted by its ability to retain a larger percentage of the data.

The character-level method makes the opposite tradeoff from the conservative method, sacrificing precision for recall. This allows personalized touch models to be trained fast, but the accuracy of these models may suffer as a result. The success of such a method is likely

to be effected by both the user's cognitive input style and the domain. For instance, some users may be more prone to proofreading and correcting their errors than others and some domains may require users to proofread more carefully than others.

Word-Level Method. The word-level method attempts to find a middle ground between the conservative and character-level methods. In the word-level method we again handle words involved in edit operations as we did in the conservative method. We also follow the logic of the character-level model to assume that most of the data not edited by the user was correctly interpreted by the system. However, the word-level method only trusts those words that it knows; that is, it only considers words that are in its internal vocabulary to be correct, while disregarding touch points from words it is not familiar with. For characters not found in words such as spaces and punctuation, the word-level method acts like the character-level method, trusting the systems prediction. The assumption made by the word-level model is that words that it is unfamiliar with are more likely to be typing mistakes than those that it knows. Although there is still the possibility of real-word spelling errors (Mays et al., 1991; Hirst and Budanitsky, 2005), removing out-of-vocabulary words is likely to help reduce noise in the data.

3.5 Evaluation

Two different evaluations are needed to assess the quality of the the online data collection strategies outlined above. In one, the evaluation directly examines how accurately the algorithm predicts the true intent of the user, and how much data it is able to cover. In the second, the algorithm is judged by how well it can be utilized by the personalization systems it is meant to help. The focus of this section is on the first type of evaluation. The second case will be handled in the next chapter, where, to determine the effect of each method on the final touch model, we build personalized touch models on the data collected via each method and compare the performance of key-target resizing using these touch models with general and gold standard personalized models.

Once we have determined a gold standard, evaluating the data collection strategies directly is reasonably straightforward. For each touch point in the input stream (excluding editing operations), a collection strategy will either predict an intended key or determine that it cannot make a judgment about that character. To determine the extent to which the data returned by a given collection strategy can be trusted, we compute precision:

$$precision = \frac{correctIntents}{returnedIntents} \tag{3.2}$$

Where $correctIntents$ is the number of intended keys predicted by the algorithm that match the gold standard, and $returnedIntents$ is the total number of touch points in the input stream that the algorithm returns an intended key for. We can compute an error rate for the returned predictions by subtracting precision from one. This tells us the amount of noise that is added into the data by the assessment algorithm.

Although maximizing precision is important, it does us little good if we cannot get enough data to train a model. To assess this we look at recall:

$$recall = \frac{returnedIntents}{TotalNumberOfChars} \tag{3.3}$$

Where $TotalNumberOfChars$ is the total number of characters in the input stream, excluding backspace characters. This gives us a measure of what percentage of the overall data is used by the algorithm.

As mentioned above, there is a clear tradeoff between precision and recall. An algorithm that predicts intents conservatively will likely have high precision and low recall and one the predicts liberally will likely have high recall and low precision. To understand which algorithm is best, we must understand the way that this tradeoff affects the performance of the personalized model that it is being used to build. However, before this is possible we must have have an understanding of the typing data we are working with and the performance of each collection strategy on this data.

	Average
Messages Length (chars)	24.7 ± 22.4
Messages Sent Per User	141 ± 67
Total Keystrokes	5097 ± 1111
Text Keystrokes	4126 ± 963
Backspace Keystrokes	642 ± 268

Table 3.1: Statistics of the chat dataset

3.5.1 Data Set

The data collection study was carried out as described above (Section 3.3). A total of 8 users (3 female) participated in the study. Participants ranged from age 20-25. Three users owned or had previously owned mobile devices with soft keyboards, but no participants had experience with the devices used in the study.

Table 3.1 shows some aggregate data statistics. Further statistics broken up by individual user are given in the appendix. As shown, users hit an average of 5097 keystrokes over the course of one hour. Of these, about 80.9% were keystrokes that made a character appear in the text field; the other keystrokes were backspaces and keyboard modification keys such as "shift". The backspace key was hit an average of 642 times. Dialogue turns averaged about 25 text characters each, although there was significant variation between users. This is relatively short compared to previous studies of text messaging; one recent large-scale text messaging study (Battestini et al., 2010) reports an average length of 50.9 characters per message. One thing that we did not attempt to model here was the financial cost of sending a message. Because the monetary cost of text messaging is calculated on a per message basis, there is an incentive for users to send one long message instead of several short ones. This may have accounted, at least in part, for the message length differences.

Differences in message length could potentially effect the performance of key-target resizing, as the algorithm inherently treats early characters different than later ones. In particular,

because the language model takes into account the last 6 or 7 keystrokes input by the user, it will be the most useful when it has seen at least that many characters. In contrast, for the first character in the sequence it does not have as much information to differentiate between different key hypotheses. As many different letters are common at the start of sentences, the initial probabilities are less likely to be highly differentiated. On the other hand, when there in enough context, certain language model probabilities are likely to be much higher than the alternatives (e.g., $P(h|thoug)$). This means that the earliest keys in a message are likely to have more ambiguous language model probabilities, and thus rely more heavily on the touch model when differentiating alternatives.

In terms of pure quantity, one hour of continuous input using this methodology is more than sufficient to collect a data set comparable to the one used by Rudchenko et al.. This is not to say that we can expect to build a personalized model within an hour of a user picking up the device; the collection methodology is intended to be fast, so real user input will likely not provide us with data so swiftly. We will examine the question of how long it takes to build a model in a real usage scenario in Section 4.2, once we have established which collection methodology is superior and its overall retention rate. Whether or not this collects a reasonable distribution of data points is another matter we must examine.

Some other points of note:

- Users made heavy use of the backspace key, which accounted for 12.6% of all keys hit, or about one backspace keystroke for about every 6.4 text keystrokes. This would seem to suggest that error correction was prevalent (although not all uses of backspace were related to error correction)

- There seemed to be a notable difference in message style between users, hinted at by the large variance in message length. Some users preferred to send several short messages in situations were a different user may have chosen to sent a single long message.

To better understand the data, let's take a closer look at the keypresses of one individual user. Figure 3.3 shows the keypresses produced by user #4. There are several things we can

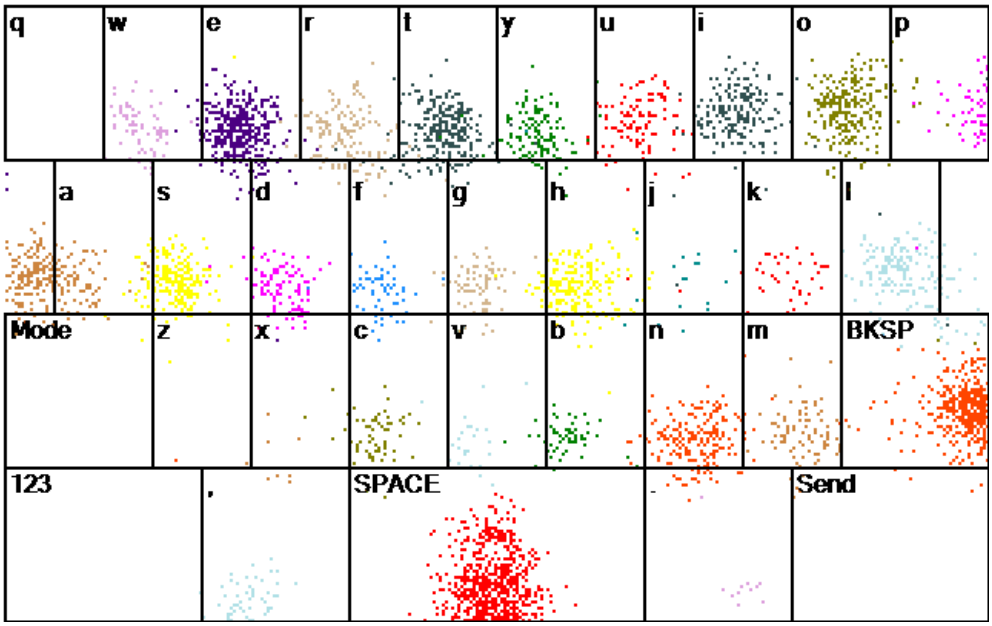


Figure 3.3: Keypress data from user 4, color coded by user intent. For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.

observe from the visualization. One quirk that jumps out immediately is that the centers of the keypress distributions generally do not fall at the center of the corresponding keys. Instead, the center of the distribution is systematically shifted downward toward the bottom of each key. If this shift is consistent for a given device independent of user, it can be trivially accounted for by adjusting each incoming keypress by a fixed amount, without the need for key-target resizing[8]. However, if key-target resizing is performed, no manual adjustment is necessary, as the KTR algorithm will account for the offset when training its touch model.

On the horizontal axis, the trend is less consistent. Again, it appears that the center of the distribution generally does not fall at the center of the key. Instead it is shifted to the left or the right, depending on the key. The trend of the shift seems to be away from a central point, but not the direct center of the keyboard. Keys to the left of the y, h, and

[8]The static keyboard used in the collection experiment did perform a small fixed adjustment.

b keys (inclusive) have their distributions centered to the left of the static key center, while those keys on the right of this line tend to have their centers shifted to the right. By looking at the demographics of this user we can begin to guess why we see such a phenomenon. User #4 is a left handed user that typed on the keyboard with both thumbs. It is likely that the user typed keys near the edge of the keyboard with the closest thumb, and preferred the dominant hand for keys in the middle. If this is indeed the case, then we can observe that the user employed a mental model that tended to consistently underestimate the horizontal distance to the key center.

When examining the variance, there does not seem to be a clear pattern. For certain keys, such as the *e*, *o* and *space* keys, the distribution varies primarily along the vertical axis, while others (e.g., *a, h, n*) vary primarily horizontally.

It is notable that the results given here are almost entirely contradictory with the observations given by Goodman et al.. While we did observe that the means of the touch distributions were shifted lower on the keyboard, keypresses seemed to be pulled away from, not towards, the center. While the variance in x and y coordinates did differ, for many keys they were surprisingly close. Of course we are only examining a single user here, but the differences in outcome are stark enough to deserve mention. It is likely that these differences can largely be attributed to differences in experimental conditions: Goodman et al. used stylus input on a desktop computer screen, not finger input on a mobile device.

One final point about the data distribution in Figure 3.3 must be addressed: the amount of data gathered per key varies widely. For instance, while frequent keys such as *e* and *space* have a large number of associated touch points, rare keys like *z*, *x*, and *q* have very few. This of course is to be expected; the frequency of letter usage in the English language varies widely, so any collection of natural English usage should show this pattern. Nonetheless, this can potentially present a problem when training key-target resizing models since certain keys may not have enough training data to accurately model user input behavior. This problem can be further compounded by the fact that rare keys may also have rare language model

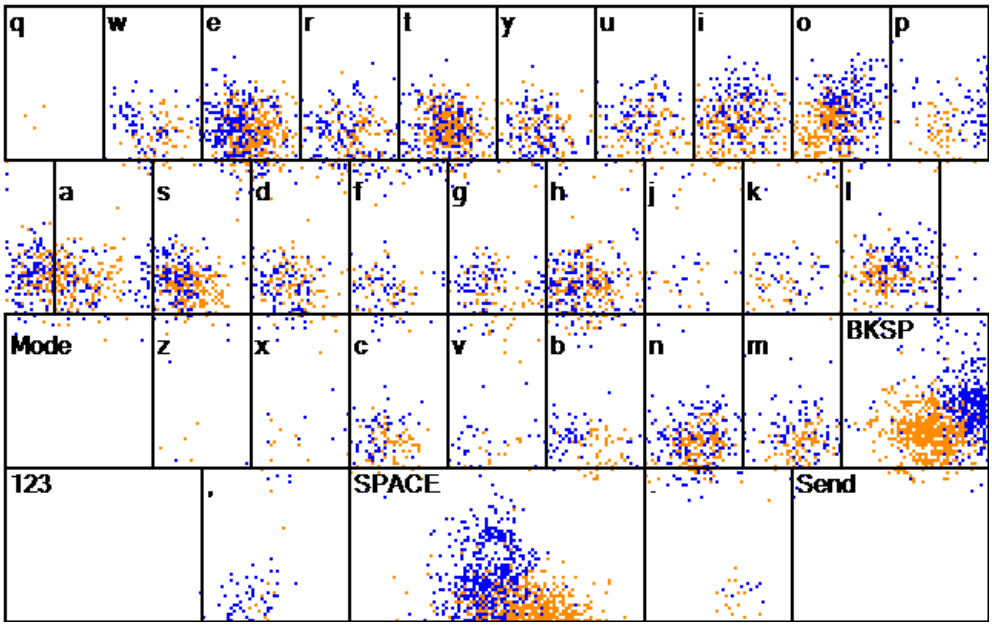


Figure 3.4: Keypress data from user 4 (blue) and user 8 (orange).

probabilities. This could lead to a situation in which it is nearly impossible for a key-target resizing algorithm to correctly predict the user's intent when they are attempting to select rare keys. Unfortunately, a systematic problem such as this is not easily highlighted when looking at the raw performance numbers. Because rare keys occur so infrequently relative to common keys, a key-target resizing algorithm could potentially have high accuracy (i.e., low keystroke error rate) even if it predicted the rare keys incorrect every time. Because of this, our analyses will need to take a look at per key accuracy to get a full picture of how the key-target resizing algorithm is performing.

Because this work is more concerned with between user variation than between keys for a single user, it would be beneficial to compare data from two separate users. Figure 3.4 shows the keypress data of user #4 from Figure 3.3, as well as data from user #8. Both users typed on the same device, the Nexus S. To enhance readability, all keypresses of user #4 are shown in blue and all key presses of user #8 are shown in orange, regardless of intended key.

Figure 3.4 helps to give some indication that personalization might be worthwhile. That

is, because the figure clearly shows that there is significant between-user variance, this suggests that modeling touch points at the user level is potentially more beneficial than building an aggregate general model. Thus, the data in Figure 3.4 supports the need for personalization. For certain keys there is a clear difference in behavior between the two users. One such example can be seen in the *e* key. The mean of the *e* key's distribution is shifted to the left for user #4 relative to the key center, while user #8's distribution is shifted to the right. This pattern persists for the sourounding keys; the *w* and *r* keys show similar shifts. This means that an aggregate model of these two users would have a more central mean and a wider variance. This would lead the model to see much more ambiguity than actually exists, leading it towards mistakes that could be more easily avoided by a personalized approach.

Another clear example is the space key. Here the variance of user #4's distribution is principally along the y axis, while user #8 distribution varies predominantly along the x axis. While we may expect to see user #4 make a keypress that is ambiguous between the *space* and *v* keys, these distributions are well separated for user #8. A combination of these two distributions would be poorly represented by a single bivariate Gaussian distribution, and would again lead to a system that saw ambiguity when none existed.

Given that our data set shows a potential role for personalization (consistent with previous work), we should expect that a key-target resizing algorithm using a personalized touch model trained on known intents would outperform one using a general model. Of course, since a system cannot know intents when collecting online, we still do not have a good sense of whether these gains will persist using noisy data collected via the proposed assessment algorithm. But before we can see the effect of these assessment mechanisms on key-target resizing we should examine their raw collection statistics, to see how much data they can retain and how accurately they predict the user's true intent.

User	None	**Conservative**		**WordLevel**		**CharacterLevel**	
	Precision	Precision	Recall	Precision	Recall	Precision	Recall
1	0.934	0.990	0.152	0.989	0.828	0.986	0.904
2	0.952	0.989	0.093	0.994	0.812	0.986	0.959
3	0.909	0.992	0.186	0.997	0.834	0.995	0.897
4	0.965	0.987	0.098	0.997	0.904	0.997	0.948
5	0.958	0.988	0.094	0.997	0.812	0.988	0.947
6	0.928	0.984	0.125	0.996	0.773	0.977	0.915
7	0.871	0.996	0.148	0.957	0.819	0.943	0.921
8	0.908	0.994	0.054	0.964	0.652	0.963	0.711
Overall	0.930	0.991	0.123	0.989	0.812	0.982	0.907

Table 3.2: Precision and Recall for the 3 data collection methods for all users.

3.5.2 Adaptation Strategy Evaluation

Each data assessment method was run on the input streams of each user and compared to gold standard logs to determine precision and recall. Results are shown in Table 3.2.

The first column in the table shows the precision of the static model that was used by the system during data collection. This represents the case in which no attempts are made to discover the intent of or filter out data we are uncertain about. If we simply trust the intents provided by the system we obtain precision of 0.930. Put another way, the naive approach in which we build a dataset that reflects how the system currently acts produces about 7% noise, on average. Because we trust every point seen by the system, the recall of the static model is always 1 (not shown in Table 3.2). However, it is important to note that the cases in which we are regarding as noise are likely to be some of the most critical cases to model. That is, since we wish to improve overall performance, it is imperative that we can discover cases in which the system did not act optimally and adjust accordingly.

As shown, the conservative method had the lowest recall of all methods, as expected. On average, the conservative method captures a little over 12% of all data typed by the user, meaning that it would require over 8 times as much data as a perfect collection method that

knew the intent of every keypress. As expected, the average precision of the conservative method was good, with an error rate of under 1%.

The results of the word and character level methods show more promise for timely data collection. On average, the character level method was able to capture 90.7% of the data with an accuracy rate of 98.2%. In comparison, the word level method captured about 10% less data than the character level, while reducing error rate by an absolute 0.7% (38.9% relative reduction). However, it is unclear from this analysis what effect this difference in error rate will have on systems trained from this data and how significant this reduction in error rate is relative to the reduction in collection rate. This will be examined in more depth in the next chapter.

One additional point about user error correction behavior is hidden in the table. Because the character level method trusted every point except those that were in editing operations but thrown out by the conservative model, we can see what percentage of editing operations the conservative model actually retained. On average, the character level model threw out 9.3% of the data, while the conservative model retained 12.3%. This means that 21.6% of the overall data was involved in an editing operation of some kind, but 43.1% of all editing operations were discarded, either because they were determined to be non-error related (e.g., restarts) or because their intent could not be confidently determined.

The numbers in Table 3.2 also highlight some important between user differences. The overall input error rate of each user varied, from as low as 3.5% to nearly 13% in one instance. Because non-error related editing operations are disregarded by all methods, users that used many restarts or other editing operations not related to error correction (e.g., user 8) had lower overall recall rates.

The results given here suggest that it is possible to use the simple online data collection strategies proposed in this chapter to retain many of the touch points produced by the user (in the word and character-level cases) without introducing a large quantity of noise into the data. The next chapter takes a look at how this effects the personalization process.

CHAPTER 4
Key-Target Resizing Personalization

This chapter takes the adaptation strategies outlined in the previous chapter and uses them to build and evaluate personalized key-target resizing methods. By doing so, we are able to get a practical understanding of the utility of such methods. One primary question examined in this chapter is whether or not personalized models collected online can outperform general models trained on aggregate data from multiple users. Additionally, we will examine how these models compare to non-adaptive personalized models trained on gold standard data. Further analysis gives insight into how long it takes to produce a viable personalized model via online data collection and how the performance of personalized models compares to general models for each individual key. Hybrid models are also proposed and evaluated, with the goals of producing greater gains in performance and handling some of the shortcomings of the personalized models. Finally, this chapter ends by briefly describing a few additional strategies that were attempted but proved to be unsuccessful.

4.1 Evaluating Personalized Key-Target Resizing on Noisy Data

To evaluate the collection methods in a real-usage scenario, we used the data collected by each method to train key-target resizing algorithms. The training procedure was as described in previously (Sections 2.3.1 and 3.2). About 1 million characters from text messaging data from the NUS SMS Corpus[1] was used to train the language models. Because the size of key anchors must be empirically learned from data which may add a confounding factor to our

[1] http://wing.comp.nus.edu.sg:8080/SMSCorpus/

analysis of touch model behavior, key anchoring was not implemented.

In this evaluation, each algorithm is run on the collected input streams to extract the predicted intents. The output the the algorithm is then used to train a personalized touch point model. The simulation procedure introduced by Gunawardana et al. (2010) is then used to calculate the error rate of the personalized model (using the metrics outlined in Section 2.4.1). Using this methodology, we can compute in what way the tradeoffs inherent in each of the online data collection strategies will effect the final performance of a personalized key-target resizing algorithm.

Leave-one-out cross validation was used for evaluation. Because context is important to key-target resizing, one full dialogue turn was held out as a testing instance for each round, rather than one single button press. To better compare to previous work and to account for the limited amount of data, key-target resizing was only evaluated on keys that appeared on the alphabetic keyboard (English alphabet characters, period, comma, and spacebar). During data collection users were allowed to switch to other keyboard layouts which contained numbers and additional symbols (such as the one shown in Figure 3.1b); these keypresses were included in the context seen by the key-target resizing algorithm, but not considered for evaluation.

A total of seven different data collection methods were examined. Two methods use aggregate data from other users, while the other five methods explore personalization with training data sets extracted via different data assessment methods. These methods differ only in the dataset used to train the touch model. The methods are as follows:

- **Aggregate Methods:**
  - *General Method.* An implementation of the typical non-personalized methods used in previous work. This method uses aggregate gold standard data from all other users to train the touch model.
  - *Device-specific Method.* Similar to the general method, but only uses gold-standard aggregate data from users who typed on the same device as the the testing user.

	Aggregate		Personalized				
User	General	Device	No Filtering	Conservative	Character-Level	Word-Level	Gold
1	4.8%	5.1%	5.2%	10.4%	3.7%	3.8%	4.3%
2	4.8%	4.2%	4.7%	10.4%	3.9%	3.8%	3.6%
3	7.9%	7.7%	9.6%	12.6%	7.4%	7.1%	7.0%
4	2.2%	2.2%	2.7%	10.1%	2.0%	2.0%	2.3%
5	2.6%	2.4%	3.5%	9.7%	2.5%	2.0%	1.9%
6	5.1%	5.5%	6.7%	12.6%	6.2%	4.8%	4.6%
7	4.4%	4.5%	5.6%	11.1%	4.4%	4.6%	4.5%
8	5.7%	5.8%	6.5%	24.4%	6.3%	6.4%	6.2%
Avg.	4.8%	4.7%	5.6%	12.2%	4.5%	4.3%	4.3%

Table 4.1: Keystroke error rate for each user using touch models trained on data collected using a variety of methods.

- **Personalized Methods:**
  - *No Filtering Method.* Baseline method that assumes that every keypress was interpreted correctly by the static model during data collection. Uses every keypress (by the current user) to train the model.
  - *Conservative Method.* Trains the touch model on the dataset extracted using the conservative inference method described previously.
  - *Word-level Method.* Trains the touch model on the dataset extracted using the word-level inference method described previously.
  - *Character-level Method.* Trains the touch model on the dataset extracted using the character-level inference method described previously.
  - *Gold Standard Method.* Trains a personalized model using gold standard (human annotated) intents.

Table 4.1 shows keystroke error rate for all runs and all users. As shown, the general method performs fairly well with an average error rate of 4.8%. Surprisingly, the device specific method does not significantly outperform the general method, suggesting that the

differences between devices (for different users) may not be significant or systematic enough to effect error rate. However, it should be noted that this represents a small study of only a few users and only two devices, so this result should be thought of only as a preliminary examination of between device differences. Furthermore, to truly examine the effect of device on typing behavior would require the study to control for user, which was not done here.

The conservative method performed significantly worse than all other personalized methods, included the unfiltered method. There are two factors that are likely to contribute to this result. The most significant factor is simply a lack of data. Because the conservative model only captures about 12% of the data, it only has a few hundred keypresses to examine given the current dataset, spread among all keys. This is likely to be too sparse to train accurate models. In order to see the true effect of the conservative method without data concerns would likely require data collection times of 5 to 10 hours per user, which is out of the scope of this study. The other factor that may have effected the conservative method is the distribution of points that it collects. Because it only retains points involved in an error correction operation, it is likely to collect a dataset that suggests that the original system had a much higher error rate than it actually did, potentially skewing the data towards erroneous cases.

The character-level method outperformed the general model by a small but statistically significant (McNemar's test, $p < 0.01$) margin. The average error rate of the character-level model was an absolute 0.3% (relative 6.3%) lower than the general model. The word-level model performed best overall, with a average error rate of 4.3%. This represents a 0.5% reduction in absolute error rate in comparison to the general model, or a 10.4% relative reduction. Noteably, the word-level model performed just as well on average as the gold standard model, suggesting that it is possible to build models on automatically extracted training data that compare favorably to models trained in controlled settings.

4.2 Discussion

Although the reduction in error rate as a result of personalization may seem small, in practice it is pretty significant. Because the character input rate of text entry is high, a small reduction in error rate can lead to significant performance improvements, and the differences in systems will quickly become apparent under heavy usage conditions. An early study of soft keyboard text entry (MacKenzie and Zhang, 1999) reported text entry rates of 28 words per minute for inexpert users on the QWERTY keyboard. At this rate, the 0.5% absolute difference between the error rates of the word-level and general models would mean that the general model makes one additional mistake for every 1.4 minutes of continuous input.

It should be noted that the error rates reported here for both general and personalized models are generally lower than those reported by other key-target resizing work, although the few studies that have given error rate measurements thus far have not been entirely consistent with one another. Gunawardana et al. report error rates between 8% and 12% on different test sets, but do not give individual user error rates. Rudchenko et al. report error rates between 5%-21% from various users. Goodman et al. only report error rate ratios, not raw error rates.

There are several factors that may have led to these error rate discrepancies. Most notably, the differences in data collection methodology likely had a non-negligible effect. The static keyboard used in data collection for this study may have caused users to change their typing behavior to be more conservative, resulting in error rates that are lower than true error rates. Similarly, the permissive "perfect" key-target resizing methods used in previous work likely had the opposite effect, which may have lead to overestimated error rates. Additionally, allowing users to see when they have made a mistake and allowing error correction operations may have had an effect on overall accuracy, as users may more actively adjust to improve performance. However, despite the discrepancy in value, this work still validated the findings of previous work, in that general key-target resizing outperformed

static key targets and personalized key-target resizing outperformed general models.

The word-level model was able to achieve performance on par to the gold standard model, while still retaining a high percentage of data during data collection. Because of this and because the word-level model outperformed the other models, the rest of this chapter will use the word-level model as the default means of comparison when discussing ways to improve online adaptation. It may at first seem suprising that the word-level model can perform as well as, or for some users slightly better than, the gold standard model. It is important to note that the gold standard is only the reference upper bound for evaluating the accuracy of data collected by the three approaches. Since key-target resizing includes both the language model and the touch model, and the relative weights applied to balance the effects of these two models, the gold standard may not always lead to the best performance in key-target resizing.

Given that the word-level model is the highest performing of our online data collection strategies, we should examine what it would take to implement it on an average system. Thankfully, the word-level method can be performed simply using information that should already be available to any current generation mobile device. Since the system must interpret each key touch, it certainly has access to the stream of keypresses input by the user, which is sufficient to handle those cases involved in an error correction operation. It should also have a readily available internal dictionary, as this is necessary for other text entry aids offered by mobile devices, such as automatic word completion and spelling correction. With this information already at hand, a system should be able to perform the word-level data collection method with minimal overhead.

As mobile device text input is frequent, the high retention rate of the word-level method should be more than sufficient to make training online personalized models practical. However, it is as of yet unclear how long the data collection process must be carried out before accurate personalized models can be trained. While Rudchenko et al. showed that their chosen data set of 2500 keypresses per user was sufficient for personalization, no previous

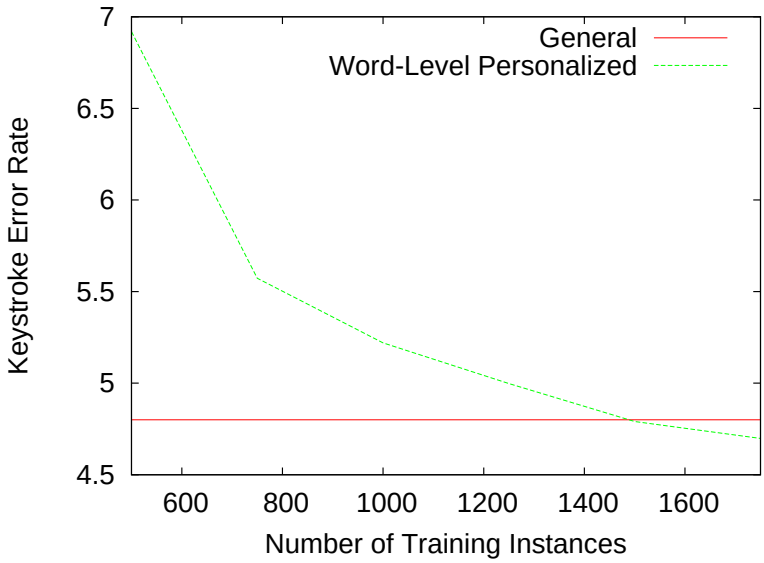


Figure 4.1: Average performance of personalized models trained on datasets of various sizes.

work has given concrete numbers about the minimum amount of data that is needed to train personalized models that outperform general models. Put another way, we wish to discover the critical point at which we have collected enough data such that a personalized model will perform at least on par with a general model. To estimate this, we varied the amount of training instances available to the word-level model and examined error rate. Results are shown in Figure 4.1.

The solid red line in Figure 4.1 represents the average error rate of the general model. As shown, the results suggest that a surprisingly low amount of data is necessary to reach comparable performance. On average, about 1500 keypresses were necessary in order to meet the performance of the general model. The 81.2% average retention rate (recall) of the word-level collection method means that the user must type about 1850 characters for the system to capture enough data to build a model that performs at least on par with the general model.

Given this, we can use data from previous studies of mobile device usage to estimate

how long data collection would take for an average user. A study of text message usage by Battestini et al. (2010) suggests that the average text message is 50.9 characters in length and the average text message user sends about 7 text messages per day, for an average rate of around 356 characters per day from text messaging alone. This means that using the word-level method a system could get enough data to train a personalized model in 5.2 days of normal usage with only text message data. In fact, the actual collection time is likely to be lower because these character counts only include those characters that make in into the final message, excluding all other characters in the input stream (e.g., corrected errors and restarts). Additionally, pulling data from additional sources of text entry, such as email or web search, will reduce the collection time further. It seems clear from this data that training a personalized model requires a reasonably small time investment.

Of course, this does not mean that no help could be given to the user for the 1850 or so initial keypresses in which no personalized model is present. One straightforward way of handling this would be to provide a user with a general model to use intially, and then replace this with the personalized version once a sufficient amount of data had been collected. Unfortunately, this would mean going through the time and expense of building an aggregate touch model that is only going to be used in the initial few days of the devices lifespan, and then discarded. For a single user this may be a small gain, but such an expense is not unreasonable for a device manufacturer who expects to ship thousands or millions of devices. Nonetheless, there is an alternative, one that could squeeze as much worth out of an aggregate as possible, and hopefully even improve upon the performance of relying solely on a personalized model.

Up until this point we have been discussing personalized and aggregate models as two distinct alternatives. However, combining these approaches may prove to be beneficial. The analysis from Figure 4.1 divided the lifetime of a user's text input into two stages: an initial stage in which not enough data is available to build a personalized model (resulting in personalized models that perform worse than the general model) and a secondary stage

in which the personalized model outperforms the general model. While we could default to a general model for the first stage and replace it with the personalized model for the second, we could instead combine the models to implement a slower transition; we could start with a pure aggregate model and add personalized information as it comes in, slowly increasing its importance until we eventually end up with a pure personalized model.

This method has potential benefits for both the first and second stages of text input. In the first stage we can incorporate personalized information earlier, potentially increasing performance over an aggregate model without waiting to see 1850 characters. Even after we reach the second stage where we have collected enough data for a full personalized model we may still benefit from drawing information from the aggregate model, as such a model performs better on rare keys that the personalized model still lacks sufficient data to examine. In the next section we will discuss and evaluate possible hybrid methods.

There is another reason why we may feel that hybrid methods are necessary. The data in Table 4.1 suggests that the overall performance of personalized models is greater than general models, but this does not tell the whole story. In order to get a better understanding of the performance of each system we should examine not only the overall performance, but the performance of each system for each separate key. These results are given in Table 4.2, along with the number of times each key appeared in the data set. The table suggests that a reasonable model can be built from a relatively low number of keys: most keys with over 150 keystrokes have error rates around 5% or lower. As expected, the frequency of the keys roughly follows the power law, resulting in the most frequent keys appearing far more often than the least frequent keys. This means that it will take significantly more time to collect data for the least common keys than the most common keys. For instance, across the one hour data collection session users hit the *space* key an average of 645.5 times, while they only hit the *q* key an average of 2.75 times. While it may be unclear how many keys presses are needed to build a good model for a given key, it is clear that is more than 3, as the word level model performs abysmally on the least frequent keys *q* and *z*, correctly predicting the

intent on only 3 of 22 *q* cases and not once out of 23 *z* cases.

This behavior is clearly not ideal. Systematic errors such as this can potentially be the most damaging. Afterall, a key selection algorithm that works poorly across several different characters is annoying, but an algorithm that is incapable of ever selecting certain keys is nearly unusable. Additionally, since the least common keys appear so infrequently, these systematic mistakes are not clearly projected in the overall error rate; the word-level model outperforms the general model, even without ever predicting the intent of a *q* keypress correctly. The good news about systematic mistakes is that if we are able to understand why they occur, we can correct for them. One such example of this is the key anchoring idea proposed by Gunawardana et al. which can indirectly help to alleviate this problem, but in our discussion of combined models we will examine a further idea that attempts to pull data from the general model to improve the performance of infrequent keys.

There is one anomaly in Table 4.2 that bears mentioning: the letter *u*. Despite being the 10th most frequent letter will over 1300 instances in the corpus, the error rate is much higher than any other key for the general model, and than all but the infrequent q and z keys for the word level model. There are several factors that may contribute to this: 1) the placement of the *u* key on the QWERTY keyboard relative to other keys may make it more error prone than other keys; for instance it is directly adjacent to the *i* key, and these keys often occur in similar language model concepts. 2) the general placement of the key towards the center of the keyboard may have made it more error prone than outer keys. 2) The text messaging language model used may have had some effect on overall performance, as the *u* key (and the adjacent *y* and *i* keys) appear in different contexts in text messaging data than in other text data; in particular, they are frequently used as phonetic substitutions in text messaging (*u* substituted for *you*, etc.). Regardless of the cause, the poor performance of the *u* key does not appear to be directly related to the personalization procedure, as it's error rate is high for both the general and personalized models.

Key	Count	General Model	WordLevel Model
Space	5164	0.6%	0.4%
e	2973	3.5%	3.3%
t	2320	2.9%	3.7%
o	2104	4.9%	2.9%
a	2084	3%	1.6%
i	1920	4.2%	4.2%
s	1727	4.1%	2.7%
n	1596	3.8%	3.7%
h	1578	3%	3.1%
u	1316	34%	33.1%
r	1273	5.6%	4.8%
l	1241	1.5%	1.2%
d	956	5.9%	4.3%
y	800	4.2%	4.1%
m	768	5.3%	3%
w	688	5.1%	4.4%
g	595	5%	3.7%
c	591	5.1%	3.2%
b	508	4.9%	4.9%
p	451	4.4%	2.9%
f	400	7%	6.2%
k	398	4.3%	3.5%
.	306	2.9%	2.6%
v	232	5.6%	6.5%
,	166	2.4%	3.6%
j	106	15.1%	13.2%
x	68	16.2%	22.1%
z	23	13%	100%
q	22	18.2%	86.4%

Table 4.2: Keystroke error rate for each individual key for the word-level and general models. Keys are ranked by their frequency in the data set.

4.3 Combined Methods

Although it is easy to suggest that combining the general and personalized models could improve performance, this leaves us will the non-trivial task of deciding how exactly the models should be combined. There are several different methods that seem intuitive, and it is not clear without empirical information which one would be superior. Given this, instead of picking an arbitrary method and analyzing the results, we will compare a few different methods.

Our data analysis suggests that the personalized model performs poorly on the most infrequent keys, as it has insufficient data to accurately model their distribution. Our intuition might be then that the ideal combined model would use the ample data from the general model to ensure that a sufficient number of key presses were available for each key. This can be done by using a small sample of general model keypresses as an initial seed to the personalized model. That is, instead of the current method of building a personalized model which starts with no training data whatsoever and slowly builds a training set as the user types, the seeding method would start with a set number of keypresses provided by the general model for each key, and then proceed to collect personalized data as normal. In the early stages of data collection when there are few personalized keypresses for each key, the model will naturally rely more heavily on the seed instances provided by the general model. Once the user has been using the system for a while and a large number of personalized points exists for each key, the small number of seed points will have a fairly negligible effect on the overall touch model. This method should naturally adapt to the differences in key distributions: the most frequent keys will quickly build up enough personalized data to make the seed points obsolete, while the least frequent keys will have the seed points to rely on during the much longer period in which they must wait to build an accurate personalized model.

It seems logical that the size of the seed should be reasonably small; large enough to give

at least a mediocre estimation for a key's distribution on its own, but small enough that once enough personalized data exists it exerts little influence. The results in Table 4.2 give us some insight into how large the seed should be. While the general model substantially outperforms the personalized model for the least frequent keys, the personalized model starts to show comparable performance for the x key (average of 8.5 keypresses) and starts to consistently outperform the general model by the period key (average of 38.25 keypresses). This gives us a rough estimate of the number of points needed per key to build an accurate model, and gives us a starting point in deciding how large our seed should be. It is unclear whether the ideal seed size would be at the high or low end of this range, or at some point in between. Of course, rather than speculate which would be most intuitive, we can, should, and will examine the matter empirically.

Although the seeding method provides a simple and intuitive way to overcome the lack of data for infrequent keys, it does have some potential shortcomings. Firstly, while it may help us start the personalization process earlier, it has the potential to actually require more data than the pure personalized model before real gains can be seen. This problem arises because we must wait for the number of personalized points to because so frequent that they make the effects of the seed negligible. So while 15 points may have been sufficient to build an accurate personalized model with no seed, with a seed of 15 points we would have to collect many more personalized points to sway the distribution towards the personalized model. Additionally, combining the seed points with the personalized points may cause problems for middle cases, where there are roughly as many seed points as personalized points. At this point the distribution of all points may be poorly modeled by a bivariate Gaussian, as the points come equally from two reasonably distinct distributions. Related to this problem is the fact that the impact of the seed points could linger far longer than intended, skewing the model even if the number of personalized points is far greater than the number of seed points. This is because once we have collected enough data to estimate a tight distribution of personalized touch points, the seed points are essentially outliers to this distribution.

The second method of building a combined model would be to use a backoff mechanism. That is, let's use the personalized data when we can, but when not enough data exists we default to the general model. This decision can be made on a per key basis. For instance, assume we encounter a keypress that falls in between the q and w keys and that we have enough data to build a personalized model for w but not q. In this case we would estimate the probability of generating the touch point given the character is w with the personalized model, but use the general model for q. More formally, let x_key be the number of training data points for a given key. Then our backoff mechanism calculates the touch model probability as follows:

$$P(t_i|c_i) = \begin{cases} P_{per}(t_i|c_i) & \text{if } x \geq c \\ P_{gen}(t_i|c_i) & otherwise \end{cases} \tag{4.1}$$

where $P_{per}(t_i|c_i)$ and $P_{gen}(t_i|c_i)$ are the probabilities of the touch given the character for the personalized and general models, respectively, and c is the backoff threshold. The backoff threshold determines when we have enough data to trust the personalized model. Similar to the seed count, we will examine several backoff thresholds to determine which produces the best results.

The backoff method handles the potential problems with the seeding method by segregating the two models. For a given key, we pick either the general or personalized model, and thus do not have the problem of potentially contaminating our personalized model with unwanted points. The backoff model is not without its problems, though. In particular, using two different generative mechanisms for different keys is potentially problematic. For instance, we observed in Figure 3.4 that user 8 and user 4 had some significantly different distributions for certain keys. If we were to use user 4's data for some keys and user 8's data for others, we would get a completely mangled understanding of which key was most reasonable for any given point. While the backoff model does not do anything quite this egregious, combining the general model with a personalized model in this way could have

a similar effect. One potential problem is that the variance of the key distributions is not the same. Because a key distribution in the general model combines the distributions of several different users with different means and variances, it will generally have a higher variance than the corresponding key's distribution in the personalized model. Trusting the personalized model for one frequent key and the general model for an adjacent infrequent key has the potential to overpredict the infrequent key, because the higher variance of the general model will make the infrequent key seem more reasonable in borderline cases.

One potential solution to the drawbacks of the backoff model is to backoff the prediction for the entire model, not each individual key. That is, if we determine that backoff is necessary, we use the entire general model, but if backoff is not necessary, we use the entire personalized model. As our backoff threshold was previously based on the amount of training data available for a key, we must use a different threshold here. Since we want to backoff any time there is potential that we do not have enough training data for the key with the true intent, we should backoff any time it seems plausible that this is the case. Since it becomes more and more unlikely that the true intent corresponds to a key the farther away you get from that key on the keyboard, a reasonable backoff threshold would backoff if any keys close to the touch points lack sufficient data. How "close" should be defined in this case is unclear, but a metric such as "within one key's distance from the key's static position" seems reasonable.

Unfortunately, even the full backoff model has a potentially crippling drawback: it might back off far more often than the ideal case. For instance, consider a point that fell between the distributions of the a and s keys. Since a and s are common keys, we will quickly have enough data to make a personalized judgment about which of these two keys is most likely. However, we can not entirely rule out the possibility that this keypress was meant to hit one of the nearby infrequent keys such as z or q, both of which are in relatively near to this point. So in this case we would make the decision to backoff to the general model, even though the personalized model would be better equipped to differentiate between the two most likely

keys. In this way we are delaying the utilization of the personal model for frequent keys for much longer than is probably necessary. A user will probably provide enough data for a personalized model of the a key within the first day of operating the mobile device, but using the full backoff model they would have to wait the weeks or even months it might take to get sufficient data for adjacent z and q keys before this model is exploited.

The final method we will consider works similar to the backoff model in that it leaves both the general and personalized models intact, with no direct modification. Instead, it uses a linear combination of the probabilities produced by the two methods as its final probability. In other words, to compute the final touch probability for a given touch point and character, we get:

$$P(t_i|c_i) = \alpha P_{per}(t_i|c_i) + (1 - \alpha)P_{gen}(t_i|c_i) \tag{4.2}$$

where α is an empirically determined weighting factor. This means that each model gets a weighted vote as to which key is most likely. The α value can be optimized by holding out one user's data from the general model to use as a personalized instance, and finding the α value that minimizes error rate. Several α values can be trained for different amounts of data available to the personalized model. In this way we can learn a (likely low) α value for early cases where we have little personalized data and should put our trust in the general model, and a larger α value to weight the personalized model more heavily once it has matured.

The advantage of the linear combination method is that it has the potential to allow personalized information to be gradually interjected in the model as the amount of training data rises, leading to a smooth and natural progression from trusting the general model entirely to relying primarily on the personalized model. The disadvantage of the linear combination method is that it does not really address the per-key problem; the α weight is the same for all keys in a given run, so eventually the personalized model will have matured to the point where it makes the general model obsolete. When this occurs, we will likely be stuck once again with the undesirable behavior observed in the personalized model: we

Combination Type	Seed/Backoff Threshold	Keystroke Error Rate
Backoff	5	9.05%
	10	8.99%
	20	9.02%
Full Backoff	5	4.28%
	10	4.28%
	20	4.31%
Seed	5	4.31%
	10	4.27%
	20	4.20%
Linear Combination	N/A	4.15%
Word Level Only	N/A	4.30%
General Model Only	N/A	4.76%

Table 4.3: Keystroke error rate for various combined model methods.

will do better on the most frequent keys, but get the most infrequent keys that we lack data for wrong. In theory, we might also implement a per-key linear combination scheme similar to the key-level backoff model, but doing so would lead to a model with much the same disadvantages as that model.

4.3.1 Combined Method Evaluation

The combined models were implemented as described above. In all runs the data extracted by the word-level method was used to train the personalized part of the model. The word-level model was chosen because it outperformed the other online data collection strategies, and because using the gold standard model may have unrealistically represented the contribution of the personalized model in the overall combined model. Evaluation was done via leave one (message) out cross validation, as it was for the separate model data presented in Table 4.1. The results obtained when all available training data (everything extracted by the word level model and whatever data is provided by the general model, which is method dependent) are given in Table 4.3. The keystroke error rate values reported in Table 4.3 are overall error rate values, comparable to the values reported in the last row of Table 4.1. The performance

of the separate models for the word and general models from Table 4.1 and repeated in Table 4.3 to ease comparison.

As shown, the performance of the seed, linear combination, and full backoff models perform on par with the word-level personalized model. Conversely, the backoff model performed poorly. This is likely due to its previously identified shortcomings. Although some differences seem to suggest that greater performance might be obtained by using the combined model, none of the error rates reported in Table 4.3 show a statistically significant difference from the word-level model. As improving overall performance was only tangentially related to our motivations for combining the models, this result is acceptable as long as we can identify a combined method that is able to solve one of our other two problems. Of course, if the overall performance is significantly lower than the performance of the personalized model then this method would be of little use to us. Thus, we can effectively eliminate the backoff method from further consideration.

The seed and backoff models were fairly robust to differences in seed and threshold size, as varying the size of the seed or backoff threshold within a reasonable range had little effect on the overall performance. Of course, varying the size more widely did impact the performance negatively (not shown in the table). Having the seed be too high (e.g., 50) would make the combined model mimic the general model more closely, reducing its overall performance. On the other hand, having a negligibly small seed resulted in essentially the same model as the personalized model. Similarly, a small threshold for the full backoff model meant that backing off was quite rare, leading to the model to act as if it were only the personalized model, and a relatively large threshold meant that the model backed off quite often, trusting the general model in cases when it would have been performed better if it trusted the personalized model instead.

The results in Table 4.3 suggest that we could adopt the the seeding, full backoff or linear combination methods once we have obtained a sufficient amount of data without hurting our performance. Given this, we will examine the utility of each in handling our two problem

Key	Count	General	Word-Level	Full Backoff (5)	Seed (20)	Linear Combo
,	166	2.4%	3.6%	3.6%	3.6%	1.8%
j	106	15.1%	13.2%	13.2%	11.3%	12.3%
x	68	16.2%	22.1%	23.5%	11.8%	17.6%
z	23	13%	100%	60.9%	13%	73.9%
q	22	18.2%	86.4%	86.4%	13.6%	63.6%

Table 4.4: Keystroke error rate for the combined models for the 5 least frequent keys in the data set

cases: 1) improving performance for infrequently seen keys, and 2) improving performance before we have a sufficient amount of data to build a personalized model.

To address our first concern we must again look at the results by individual key. Table 4.4 reports the error rates for the 5 least frequent keys in our data set. As the results for different seed or backoff levels were similar, only a single representative seed and backoff level are shown (backoff threshold of 5 and seed of 20). The results for the word-level and general models are again given for comparison.

As shown, all three of the combined models were able to at least partially achieve the goal of improving performance on infrequent keys by helping the personalized model in cases where it lacked sufficient data to make an intelligent decision. However, the seeding method is the clear winner, as it is the only combined method that is able to approach and even improve upon the general model's performance on the least frequent keys. While the linear combination and full backoff models are still favorable to the personalized model alone, they still have error rates of over 50% when the user's intent is to select the z or q keys, which is not the improvement we were hoping for. Even though neither model performs as well as we would have hoped, neither performed so bad that they should be entirely removed from consideration. Afterall, the result is not entirely unexpected. While we were unsure about the performance or the full backoff model, this is consistent with our hypothesis about the linear combination model; its area of strength should be in improving overall accuracy and reducing collection time, not improving performance on infrequent keys.

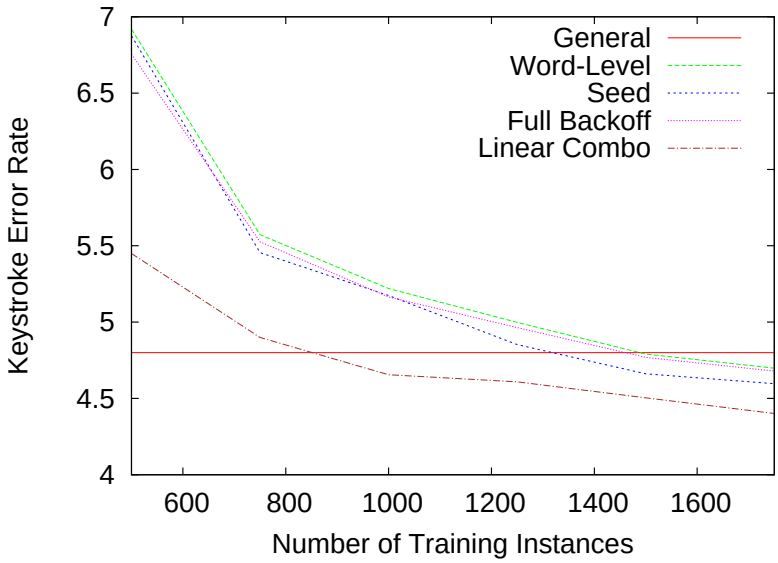


Figure 4.2: Average performance of hybrid models trained on datasets of various sizes.

To compute the necessary amount of data each model needs to collect before it can be used, we can train a model of each type on varying amounts of data and compare the performance to the general model, as before. Figure 4.2 shows the results of this procedure. The performance of the general and word-level model from Figure 4.1 are given as comparison.

As shown in the figure, we once again have a case where one model is clearly superior to the others. In this case, the linear combination model outperforms all other personalized and hybrid models at all levels of training data. It substantially reduces the amount of training data needed to see improvement, from about 1500 instances with the word-level model only to around 850 instances. This translates into a collection time of about 3.0 days on average, cutting off over two full days from the collection time of the word-level model alone. Additionally, the linear combination method produces a statistically significant increase in performance over the word-level model at all data levels shown in the graph. The linear combination model allows not only better performance, but better performance faster. However, as noted in Table 4.3, the word-level model slowly catches up, to the point were

the difference between models is no longer statistically significant when all data is included. Nonetheless, the linear combination model's ability to produce error rates at the same level or better than the word-level model means that the choice between the hybrid model and pure personalized model is clear (assuming a general model is available to build the hybrid model).

One oddity of the graph worth noting is that the performance of the linear combination model is worse than the general model when a small amount of training data is present (less than 850 instances). While it should not suprise us that the personalized portion of the model would need time to collect enough data to show improvement, one might initially hypothesize that the linear combination model should do *no worse* than the generel model. Afterall, if when training the α value we determine that we cannot find an value that produces performance greater than the general model we can simply set the α value to 0, resulting the linear combination model disregarding all personalized information. This makes the linear combination model exactly equivalent to the general model, meaning that its performance would be exactly the same. It is likely that the model did not meet this supposed threshold at lower amounts of training data due to overfitting. That is, the model overgeneralized on its small amount of data, believing that it could make use of some personalized information before it was truly reliable, resulting in performance gains during threshold training that were not kept when evaluating on unseen data.

Although overshadowed by the performance of the linear combination model, the other two hybrid models deserve mention. The full backoff model seemed to again be able to outperform the word-level model (although not with a statistically significant difference), but performed the worst out of the three hybrid models. Although using the full backoff model could improve performance, that fact that both other hybrid models outperform it in both areas in which we wish to improve suggests that further consideration of the full backoff model is unwarranted.

The seeding method showed a slightly more substantial increase in performance. The

seeding method crosses the performance of the general model at around 1300 instances, allowing it to collect enough data in 4.5 days on average, a modest gain of less than one day difference from the word-level model. This result is notable in part because it was hypothesized that the seeding process might actually slow down the personalization process rather than accelerate it, due to the presence of data points from the general model requiring more personalized points to be seen before true personalization could be achieved. The results given here happily contradict this hypothesis. It is likely that the gains in performance associated with alleviating cases of insufficient data overpower the potential losses in performance associated with mixing general and personalized points.

While hybrid models were able to help in both of the two different aspects we wished to improve upon, each case had a different conclusion as to which model was superior. The seeding method performed best for infrequent keys and the linear combination method produced the greatest decrease in data collection time. Which model is preferred would ultimately be a design decision that rested on how those implementing the system judged the importance of each area of improvement. If we are most concerned with ensuring that we do not have any keys that we routinely perform poorly on, the the seeding method would likely be most appealing. Alternatively, if it is decided that it is important to adapt early and increase overall performance whenever possible, then the linear combination method would seem to be the proper choice. It should be noted that, as with the choice between general and personalized models, this may itself be a false choice; the most appealing model may be produced by some combination of the two hybrid methods themselves.

4.4 Other Experiments

This section gives a quick summary of two other data collection methods and modifications that were examined, but were not successful in improving performance over those methods outlined above. In one, we attempt another assessment method, one that tries to further improve the accuracy of the word-level method by taking a wider context into account.

In the other, we attempt to improve the overall performance of the learned touch models by removing outlying points that may skew the result. A tangentially-related attempt to improve performance by examining touch model bigrams is also described.

By requiring that a word typed by the user be in our vocabulary, the word-level method removes many of the typographic errors in which we are unclear about the user's intent. However, in certain less frequent cases the user produces typographic errors that result in words that are in the dictionary. These real-word spelling errors are harder to detect, but if they are left in the training data they have the potential to add potentially harmful noise to the data. To combat this, we experimented with a method that used a word-level trigram language model[2] to rate the plausibility of the given word appearing in its current context. We labeled this method the *discourse-level method*.

Discourse-Level Method. The discourse level method builds upon the word-level method, but attempts to further remove potential sources of noise, placing it between the word-level and conservative methods in respect to the precision and recall tradeoff. In particular, the discourse-level method incorporates all aspects of the word-level method but it also attempts to detect and remove spelling errors in words that were in the vocabulary. It does so by considering the trigram probability of the word sequence, similar to methods of real-word spelling error detection (Mays et al., 1991; Wilcox-O'Hearn et al., 2008). Although in the ideal case it would be able to find the intended term for a real-word spelling mistake, real word spelling correcting systems do not have high enough accuracy and are likely to introduce as much noise as they remove. As such, the discourse model simply removes words whose trigram probabilities do not meet be a certain empirically determined threshold.

The discourse-level model was tested following the same procedures used in the other

[2]This language model calculates the probability of a word given the two previous words (as opposed to our character-level language model that calculated the probability of a character given the last several characters). This is the style of language model used in many NLP applications such as speech recognition and machine translation. Despite the similar nomenclature, there is no relation between word-level language modeling and the word-level data collection strategy outlined previously.

evaluation runs. The language model was build from the AQUAINT corpus, a large corpus of about one million documents from newswire text collected from 1999 to 2000[3]. Several different probability thresholds were examined. As expected, the discourse level method had a lower recall than the word-level method. Unfortunately, this did not result in noticeably different precision values. With lower recall but similar precision, the touch models built from the discourse-level method consistently resulted in higher keystroke error rates than those built from the word-level model.

Although the idea of reducing noise by removing real word spelling errors seemed intuitive, there are a few flaws that likely lead to the discourse-level method failing to overcome the word level method. The first is simply that the problem of real word spelling errors may have been overstated. While it is true that real word spelling errors do occur and that if left uncorrected in the text they would result in added noise for the word-level model, they are infrequent enough that their effect may be negligible. Furthermore, detection is difficult, resulting in many false positives. This means that a large amount of data was removed that did not need to be, resulting in a model that is sacrificing recall much more than the ideal case.

Another problem with the discourse-level method is that it had little room to improve. The word-level touch models proved to be robust enough to noise that they performed on par with the gold standard models, so even if the discourse-level method was able to consistently improve precision this may not have been enough to significantly reduce keystroke error rate. Because of this, it proved more fruitful to focus on methods that addressed the limitations of the collection rate (e.g., the combined methods discussed above) rather than those that attempted to address the performance issues.

Another potential noise reduction method that was considered was outlier detection and removal. Because it is not itself an online data collection strategy, outlier detection can be applied to training data extracted via any of the previously outlined mechanisms. The

[3]http://www.ldc.upenn.edu/Catalog/docs/LDC2002T31/

motivation for outlier detection is similar to that of the discourse-level model; we hope to remove noisy points that skew the final touch distribution. If run on top of training data extracted via an online data collection strategy, outlier detection may discover noisy points that the method missed. Additionally, one potential advantage of outlier detection is that it can remove points that had their intent correctly predicted by the system, but nonetheless are poor exemplars of the class. For instance, a user may have their finger slip substantially from the intended key while typing. The word-level method may still correctly identify their true intent (e.g., if they perform an error correction operation), but since this point is far from the rest of the points associated with that key and was produced by a motor mistake which is unlikely to systematically reoccur, we would likely be better off disregarding it than skewing our distribution to attempt to incorporate it.

To detect outliers, we followed previous work by calculating the Mahalanobis distance (Mahalanobis, 1936) from the mean to each point. Those points with the largest Mahalanobis distance can be iteratively removed from the data (Hadi, 1992), hopefully removing unwanted noise. This process can be done repeatedly until an appropriate stopping condition is met. Although this is a simple outlier detection method, and others do exist, our case is rather straightforward compared to many outlier detection tasks. For instance, while newer methods are designed to be non-parametric and to handle higher dimensional spaces, we are already assuming a Gaussian distribution for our models and are only working in two dimensions.

To remove outliers, the outlier detection algorithm was run on the data collected via the word-level method, and touch models were then trained on this data. Several different runs were conducted using different stopping conditions. Unfortunately, the results showed that removing outliers did not have a statistically significant effect on the keystroke error rate. The reasons for this are likely similar for the hypothesized reasons for the discourse-level model's failure: the model was already shown to be robust enough to noise that removing additional noisy data had a negligible effect.

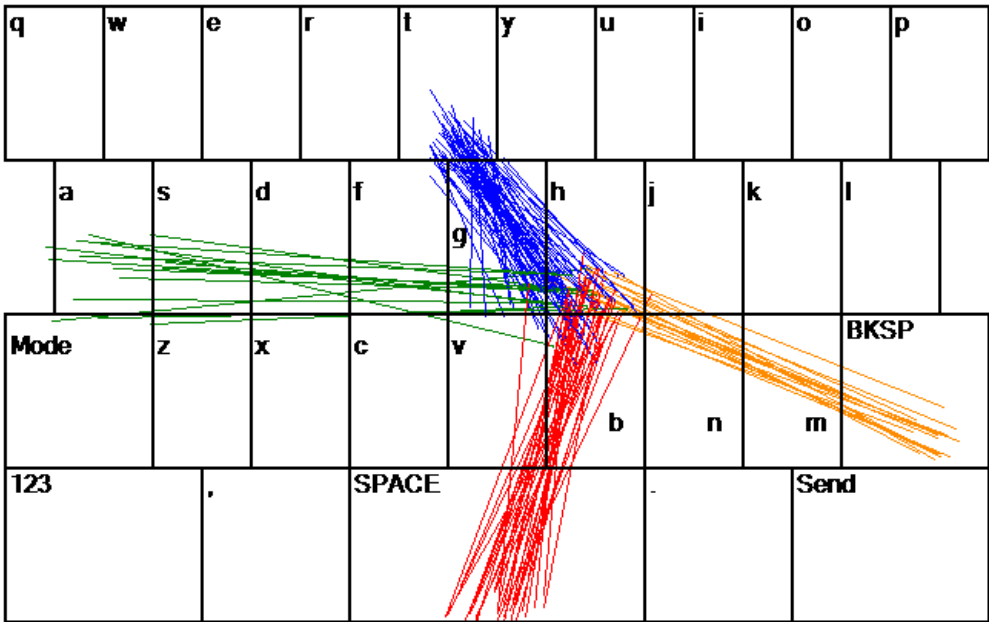
One final experiment attempted to discover if a greater range of touch model information

could be used to improve performance. All of the key-target resizing methods presented here and previously assume that a given touch point is independent of the previous touch points. However, this does not seem to be intuitively correct. Afterall, the distance from the previous key to the current key will vary based on which key was hit last, and Fitts' law suggests that the distance traveled from point to point has an effect on target selection behavior. Likewise, the direction of movement may effect the axis along which the keystrokes vary or the angle in which the user's finger hits the key. Given this, it seems worthwhile to at least entertain the idea that our independence assumption may be a poor one.

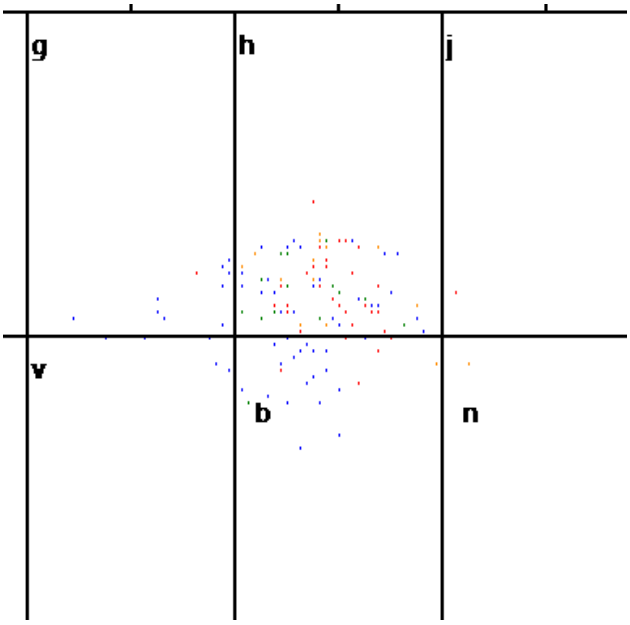
The visualization in Figure 4.3 shows an example of the differentiation that could be made by touch model bigrams. The figure shows data from user #2 in which the *h* key was the intended key. User #2 was chosen because they used single thumb input, which ensures that the movement vectors shown roughly correspond to the actual movements; with multi-finger input we have no information about which finger was used to select which key. To see if differences in distance traveled and angle effect the final touch point, the data shown gives touches in which the previous key was one of four different keys from various points on the keyboard (the *a*,*t*,*space*, and *backspace* keys). Figure 4.3a shows the movement vectors themselves, while Figure 4.3b shows the individual points only in order to enhance readability. The point colors in Figure 4.3b correspond to the vector colors in Figure 4.3a.

The utility of using touch model bigrams is not immediately apparent from the figure, although there does seem to be some amount of separation. For instance, the variance of the touch points originating from the *t* key (in blue) have a mean that is shifted to the left of those touch points associated with the *space* key (in red). The general trend, at least from this one small sample, appears to suggest that the center of the touch points for a given previous key tends to fall slightly farther away from that key than the average touch point. There is at least enough of an indication of between-key variance here to warrent further investigation.

Unfortunately, there is a problem that makes it difficult for character-level bigrams to be



(a) Movement Vectors



(b) Points

Figure 4.3: Movement vectors and touch points intending to select the *h* key, color coded by their previous keypress.

considered, and that is the amount of data required. As we have seen previously, the unigram probabilities of each key effect our ability to collect a sufficiently large training dataset in a timely manner, and this problem is exacerbated further when we consider bigrams. Most two-letter key combinations did not appear in our dataset, meaning that a bigram touch model that was built on this would likely be too sparse. To account for this but still incorporate information from the previous key, previous key information was encoded only by its primary direction of movement (north, south, east, or west), producing only four models per key.

Personalized touch models were built using this method. As with the other experiments in this section, the use of previous key information in the touch models did not produce a model that was able to outperform the previous personalized models, and in fact performed slightly worse than the general model. There are a few reasons why this may be. The first and most obvious conclusion is that the bigram information is just not differentiated enough to have an effect on the overall performance. An alternative hypothesis is that some aspect of this small investigation made it difficult to capture the true gains. It may be that the simple way of incorporating previous information was insufficient to capture the differences, or it may be that dividing up the limited amount of touch points into smaller models resulted in an insufficient amount of data. These doubts are enough to suggest that this idea should get further examination in the key-target resizing literature. However, this will conclude the discussion here, as touch model bigrams are only marginally related to personalization or adaptation.

4.5 Conclusion

The last two chapters examine the previously unexamined problem of online data collection for online adaptation of key-target resizing touch models. In order to train touch models for key-target resizing algorithms, a system must have training data that associates the key that a user intends to hit with the touch point that they selected. Because the true intent of each keypress made by a user is unknown to the system in natural usage scenarios, some amount

of inference must be applied to collect data online. This work introduced and analyzed four data collection methods that used varying degrees of inference to overcome this data collection hurdle.

To facilitate these experiments, a dataset of unscripted mobile device chat conversations was collected. This data was used to examine the tradeoff between precision (ability to predict the correct intent) and recall (data retention rate) for each collection method. A further analysis of the data took an in-depth look at the touch points produced by each user to gain insight into user typing behavior. A conservative online data collection approach that only drew inference from user error correction operations was found to introduce less than 1% noise, but was only able to retain 12.3% of data input by the user. The two methods that made additional inference about keypresses not involved in error correction were able to capture over 80% of the data, while still keeping the overall amount of noise in the dataset to under 2%.

To examine the effect that each data collection procedure had on the final key-target resizing models, the training datasets extracted by each method were used to build personalized touch models, and the error rates of systems employing these touch models were compared. The results suggested that a data collection methodology that makes inference based on vocabulary and error correction behavior is able to perform on par with gold standard personalized models, while reducing relative error rate by 10.4% over general models. This approach is simple, computationally inexpensive, and calculable via information that the system already has access to. Additionally, it is shown that these models can be built quickly, requiring about 5.2 days worth of text input by an average mobile device user.

In an attempt to further improve performance and reduce the data collection time, several methods were introduced that combined information from the personalized and general models. An analysis of the performance of these models suggested that a simple seeding method can give a small reduction in the amount of time needed to build a personalized model, while providing the additional benefit of improving performance on infrequent keys.

Additionally, the seeding method was able to provide a modest reduction in the amount of time the user must spend collecting data before the personalized model could be employed. Another hybrid model that predicted the intended key by taking a linear combination of the general and personalized models was able to produce a more substantial reduction in collection time. However, the linear combination model is not able to produce the same gains as the seeding method on infrequent keys.

Ultimately, the purpose of the last two chapters was to introduce several online data collection strategies for key-target resizing; to establish that careful application of these strategies would result in training data that could be used to personalize, continuously adapt, and improve the performance of key-target resizing methods; to give an in-depth analysis of user touch behavior as it pertains to key-target resizing; and to establish that an intelligent combination of personalized models built via online data collection strategies and general models built in the lab could further improve performance. The next chapter will introduce two very different text entry aids and establish how online adaptation and personalization can be applied in these cases.

CHAPTER 5

Adaptation Strategies for Word-Level Text Entry Aids

This chapter will outline work on self-assessment for word-level text entry aids, necessary for online adaptation and personalization. As we have done with key-target resizing in Chapter 3, we focus primarily on aids that have seen adaption in currently available commercial systems. In this chapter we will concern ourselves with two similar but distinct word-level aids, autocompletion and autocorrection.

The following section motivates the need for online data collection of autocorrection and autocompletion data via self-assessment. Section 5.2 outlines the self-assessment methods, and the data collection methodology is covered in Section 5.3. Next, Section 5.4 gives specific methodology and evaluations of the self-assessment procedures. Finally, Section 5.5 gives a brief discussion of the potential uses of these self-assessment mechanisms for personalization and adaptation, leading into a more in-depth analysis in Chapter 6.

5.1 Motivation

Although seemingly similar in nature, automatic word correction and automatic word completion are distinct in their goals. Completion systems attempt to reduce the number of keystokes a user needs to input to compose their message, which can have an effect on the overall text input speed. Conversely, correction systems attempt to reduce text input error rate.

Despite their difference in intended goal, autocorrection and autocompletion systems act similarly from a user's perspective. Both technologies modify what is typed without explicit

intervention from the user. In both cases, users may explicitly select a suggested completion if they notice it. If the user is monitoring the text closely, they may notice a completion or correction when in occurs. However, mobile device users often fail to closely monitor their input, either because spelling mistakes are tolerated in common use cases or because the lack of tactile feedback on soft keyboards means they must spend more time watching the keyboard rather than the text area (Paek et al., 2010).

The problem arises when the system provides an incorrect prediction. Consider Figure 5.1a, which gives an example of an autocorrection mistake[1]. Figure 5.1a shows a text message conversation between two participants. Speech bubbles originating from the right of the image are contributions produced by one participant (participant A), while those originating from the left are contributions by the other participant (participant B). After receiving good news, participant A attempts to reply by saying *good*. However, they chose to use the non-standard spelling *gooooooood* to show extra emotion, a technique commonly employed by text message authors (Baldwin and Chai, 2011). This spelling is not in the device's internal dictionary, so the system assumes that the user has made a mistake and attempts to perform automatic correction. The system picks the best word from its internal dictionary to replace *gooooooood*, which is *hooklike*. The user does not notice that this has happened before sending the message, and the next 3 messages in the conversation are spent correcting the error.

When participant B reports good news and receives the non-sequitur response *hooklike*, they are understandably confused. Because the participants must then expend additional dialogue turns to correct the confusion, the rate at which they can discuss the topic at hand is reduced. The goal of autocorrection is to reduce error rate and, by doing so, increase the reader's understanding, but in this case it has done just the opposite. Although it is hard to measure directly, it is likely that the cost of an autocorrection mistake is higher than the utility gained by an unproblematic autocorrection. A user may be able to understand a message that contained a spelling mistake, but is likely to be confused when this same

[1]All autocorrection and autocompletion figures in this chapter are copyright Break Media. Used with permission.

(a) (b)

Figure 5.1: Mistakes generated by (a) automatic correction and (b) automatic completion systems from the DYAC dataset (Section 5.3)

mistake is repaced by an incorrect word. As such, autocorrection systems must have high accuracy rates to be more of a boon than a detriment.

We should take a moment to understand how a mistake as egregious as that shown in Figure 5.1a can arise. At first examination, it seems that the terms *goooooood* and *hooklike* differ so greatly from one another that no reasonable system would make this mistake. Although this correction may at first seem implausible, with a little examination it is not hard to understand how the system reached this conclusion. On the qwerty keyboard, the g and h keys are adjacent, the o key is adjacent to the l, i and k keys, and the d key is adjacent to the e key. This example was produced on a soft keyboard, so each attempt to hit a key was recorded as a touch point on the keyboard. Soft keyboard key presses are ambiguous, so some of the user's key presses may have been more reasonably interpreted as one of the adjacent characters. We do not know for sure from the transcribed text of the example how the system would have interpreted the input without autocorrection; perhaps it would have still interpreted some of the key presses differently than intended (e.g., *hooolood* or *gooklooe*). Given this, it is plausible that a user that typed the sequence of touch points seen by the

system could have intended to type *hooklike*.

A similar phenomenon can arise when automatic completion systems fail. An example of this is given in Figure 5.1b. In this example, the user attempts to produce the phrase *my hairs wet*, but does not notice that they system has suggested an automatic completion. When the user hits the send button, the system interprets this as the user wanting to select the suggested completion, changing their attempt to type *wet* into *arthritic*[2]. As before, this results in a contribution that is not what the user intended, and they must spend additional dialogue turns correcting it.

As with automatic correction, when automatic completion systems fail they not only fail to improve performance, they are a detriment to it. However, unlike automatic correction, we can more easily quantify the cost. Automatic completion systems are evaluated using the notion of keystroke savings, how many (or what percentage of) keystokes are saved if the user takes the automatic completion rather than typing out the entire word (Carlberger et al., 1997). Let's assume for a second that the user truly intended to type *arthritic* in Figure 5.1b. Since the user takes the suggestion after inputting 3 characters, they save 6 characters by taking the automatic completion. However, since the completion was not correct, they were forced to spend an entire dialogue contribution of 85 characters to correct the mistake. Correcting the mistake cost the user 79 more characters than they had the potential to save if the completion was correct. Overall, the cost of a mistake in this case was over 14 times the number of keystokes they could have saved. If this is representative of a typical use case, it suggests that an autocompletion systems that is 93% accurate would lose as many keystrokes from its mistakes as it gained from its correct instances.

We see from the above examples that it is critical that autocompletion and autocorrection systems avoid mistakes. There are two principle ways in which this can be done. In one, the overall performance of the correction and completion systems is improved. Clearly, higher overall accuracy will lead to less problematic instances. The other alternative is to be smarter

[2]Similar logic to that given above for *goooooood* and *hooklike* can explain why the system may think that this is a reasonable completion.

about when we apply these technologies. If we can identify certain situations in which these systems are more prone to error, we can choose not to attempt completions and corrections in these instances. If these systems are able to automatically assess their performance, they have the potential to utilize both of these avenues of mistake avoidance.

Consider the problem of choosing when to perform automatic completion. Some users may carefully monitor their text and those suggestions provided by the system, and will thus be unlikely to accidentally accept a mistaken completion. Conversely, other users may focus on the keyboard when typing and may not notice when mistaken completions happen. The behavior of these two users suggests very different autocompletion strategies, but we do not know a priori which type a new user will be. To account for this, the system must actively monitor the user's reactions to its behavior to discover whether a more liberal or conservative completion policy should be implemented.

Similarly, if the system can assess its own performance, it can use this information to improve its performance overall. A system that can discover when it has made a mistake and what action it should have taken can use these instances as training data to build a more robust model. It is for these reasons that this work proposes to examine automatic system assessment of autocompletion and autocorrection systems.

5.2 Overview of Self-Assessment for Autocorrection and Autocompletion

Now that we have established that there is a potential need for system assessment, we need to examine what that entails. To do so, we first need to introduce some nomenclature.

First of all, we should establish that, from the system assessment point of view, autocompletion and autocorrection are nearly identical (Baldwin and Chai, 2012a). When the system performs correctly, it is not clear to an outside observer that any correction or completion has taken place at all. The end result in a correct autocompletion, a correct autocorrec-

tion, and correct unaided input is exactly the same: the character string that is recognized by the system is exactly the string that the user intended to type. Similarly, an incorrect autocorrection and an incorrect autocompletion result in functionally the same output: a word that the user did not intend to type. The words selected by the two systems would be different, but both are erroneous. Although there is some amount of nuance to each case, the proposed assessment techniques will generally not need to differentiate between an error that was produced by an automatic correction system and one that was produced by an automatic completion system. As such, unless otherwise noted, the rest of this thesis will use the term **autocorrection** to refer to any automatic augmentation of the user's character string, either by automatic completion or automatic correction.

Throughout the chapter, we use the term **attempted correction** to refer to any autocorrection made by the system; for example, *hooklike* is an attempted correction in Figure 5.1a and *arthritic* is an attempted correction in Figure 5.1b. Some attempted corrections could correct to the word that the user intended, which will be referred to as **unproblematic corrections** or **non-erroneous corrections**. Other attempted corrections may mistakenly choose a word that the user did not intend to write, which will be referred to as **correction mistakes** or **erroneous corrections**. For example, both *hooklike* and *arthritic* are erroneous corrections. We use the term **intended term** to refer to the term that the user was attempting to type when the autocorrection system intervened. For instance, in the erroneous correction in Figure 5.1a the intended term was *goooooood*, and in Figure 5.1b the intended term was *wet*.

Now that the terminology is in place, let's consider a few scenarios where self-assessment could be used to collect data for personalized performance improvement. If the system knows its past performance for a given user, it knows how well it is meeting that user's needs, and it can use this information to adopt a more conservative or liberal autocorrection policy. In order to know its past performance, the system needs to be able to automatically determine if a given correction attempt resulted in an erroneous or non-erroneous correction. It may

not even need to make this judgment for all attempted corrections, as long as it can estimate overall performance from a representative sample. Since the goal of the assessment is simply to gage the overall accuracy rate of the current system, identifying the intended term is not necessary; we only need to know that a mistake was made, not the specifics.

However, we must be able to discover a bit more information if we wish to improve the autocorrection model itself. In order to do this, we must perform data self-assessment, similar to the key-target resizing data collection given in Chapter 3; we need to obtain training data to personalize and improve a model. To train a model we must have access to not only what the system did, but what it should have done. This means that we must collect more information than just the system performance. That is, we must again know what the user actually intended for each autocorrection that we wish to use as training data. We must know how the system acted as well, but the system should already have access to this information. So, as before, for this type of personalization the primary item we must discover is what the user intended.

Now that we know what data is needed for each of the proposed methods of performance improvement, we know what information we must obtain through automatic assessment. For a given correction attempt, there are 3 primary things that we must determine:

1. Whether or not the correction attempt was erroneous (Necessary for both avenues of improvement).

2. What was typed by the user and the system's attempted correction (Necessary for improving the model).

3. The term that the user intended to type (Necessary for improving the model).

The methods proposed in this chapter attempt to address this self-assessment problem as a means of collecting relevant information for adaptation. However, before we examine the proposed methods we must address the problem of how to obtain a data set to train and test them.

5.3 Data Collection

In order to examine the problems outlined above and evaluate the proposed models, we need a dataset that contains both erroneous and non-erroneous attempted corrections. This is problematic, as a sufficiently large dataset of erroneous corrections is likely to be difficult to obtain in a laboratory experiment. The extent to which erroneous corrections appear is largely dependent on how frequently a given user monitors their input. If they monitor their input frequently, they may catch all or nearly all autocorrection mistakes before they happen. After all, this is part of our motivation for changing our autocorrection policy based on user. To obtain a large dataset of erroneous corrections we need data from users that do not monitor their text as closely, but we have no way of knowing prior to data collection which users these are.

Thankfully, there is another possible source of erroneous autocorrection data. The website "Damn You Auto Correct"[3] (DYAC) posts screenshots of text message conversations that contain erroneous instances of automatic correction or completion. The screenshots are sent in by users, and represent mistakes that arose in real text message conversations. The correction mistakes shown are generally ones that users found to be humorous or egregious.

To build a dataset of problematic instances, screenshots from the DYAC website were downloaded and transcribed. Manual annotators annotated each correction mistake in the dialogue with its intended term.

There is one caveat associated with this type of data collection: we cannot be entirely sure that the data is not falsified. Because screenshots can be submitted by anyone via the web, they could send in fabricated examples. The focus of the website on humorous screenshots may exacerbate this problem. Consider the screenshot shown in Figure 5.2. In this example, the word *help* is replaced by the word *whore*. Unlike the examples in Figure 5.1, it is difficult to understand how an autocorrection system could make this mistake. Most of the letters

[3] http://damnyouautocorrect.com/

Figure 5.2: Autocorrection example that may have been falsified

in the two words that would need to be confused for this mistake to be made are found far from one another on the keyboard. While the two words have some phonetic similarity, the differences are substantial enough to arouse suspicion. Our knowledge of how autocorrection systems work suggests that this mistake would not arise easily. Although it is still plausible that the instance shown in Figure 5.2 represents a true autocorrection mistake, the chance that it is falsified is too great to consider it. To remove these instances, the data collected from the DYAC website was manually filtered before being included in our dataset.

Although the DYAC dataset allows us to build a corpus of autocorrection mistakes that we would not normally be able to acquire without great difficulty, it does come with several disadvantages when compared to an ideal dataset for the proposed work. Most notably, the fact that the data is presented in screenshots rather than input stream logs limits our ability to interpret it. Even if we can determine from the screenshot where the correction mistake occurred and what the intended term was, we do not know the coordinates of the series of touch points that led to the mistake. Without this information we cannot build a set of training data to improve the system[4]. Thankfully, since we still can determine when the

[4]It should also be noted that even with input stream information we would not be able

system produced an error, we can still evaluate the self-assessment mechanisms, which is our primary goal.

An additional problem arises from the fact that we do not always know the user's intent. In the examples given in Figures 5.1 and 5.2 and in the majority of cases on the DYAC website the user explicitly corrects the mistake in the follow-up dialogue. Unfortunately, this is not universally true; some cases do not have the intent of the user explicitly stated. In many of these cases the intended word can be inferred from the context, but this is not universally the case. Because we cannot build a gold standard without knowing the user's intent, cases in which the user's intent can not be determined were removed from the dataset.

The arbitrary length of the dialogues in the screenshots is also unfortunate. Because the screenshot only shows however many text messages fit on the screen, we only see a portion of the overall dialogue. This means that while some screenshots may be cropped such that the entire conversation is seen, others may have only a small snippet of the overall context. In some cases only the dialogue after the correction mistake is shown, while in others most of the dialogue shown is prior to the correction mistake. In the ideal case we would be able to examine the context before and after a correction attempt in order to determine its fidelity.

There is a final (but very critical) limitation that we must address: the DYAC dataset contains *only* instances of problematic corrections. In order to examine the initial task of differentiating between problematic and unproblematic instances we must also have a comparable dataset of unproblematic instances. Thankfully, the chat dataset collected in the work presented in Chapters 3 and 4 should meet our needs in this regard.

As we have mentioned, unproblematic autocorrection instances result in dialogues that are indistinguishable from instances where no autocorrection is performed and the user has typed their intended word correctly. The data collected for the work in Chapters 3 and 4, which does not perform any autocorrection, provides many good examples of non-erroneous dialogue snippets. Because users are asked to correct their uncorrected errors, we know which

to examine personalization directly, as the data comes from a variety of users.

parts of the dialogue are problematic and which are not. To get a dataset of unproblematic dialogue snippets akin to the DYAC dataset, we can extract dialogue snippets from the unproblematic regions. To ensure that the regions extracted are as close as possible to the size of the snippets in the DYAC dataset, the snippets were sampled to follow a normal distribution with the same mean and standard deviation as the DYAC dataset.

5.4 Autocorrection and Autocompletion Performance Assessment

We examine the self-assessment task as two problems, corresponding to the two possible sources of improvement. To dynamically tune the number of autocorrection attempts to be more liberal or conservative based on a users input style, we explore the task of differentiating between erroneous and non-erroneous instances. To build training data to improve the overall model we explore the tasks of identifying the autocorrection attempt and intended term for problematic instances. As problematic instances have to be identified for the latter task as well, we will discuss the differentiation task first.

5.4.1 Differentiating Between Problematic and Unproblematic Instances

The first task that must be undertaken for autocorrection self-assessment is determining whether a given correction attempt is problematic or unproblematic. This can be modeled as a simple binary classification problem. Given an arbitrarily sized dialogue snippet that contains an autocorrection attempt, the task is to make a binary decision that determines whether the autocorrection attempt was unproblematic or erroneous.

The proposed method follows a standard procedure for supervised binary classification. First we must build a set of labeled training data in which each instance is represented as a vector of features and a ground truth class label. Given this, we can train a classifier to

differentiate between the two classes. For the purposes of this work we use a support vector machine (SVM) classifier.

Feature Set

Since dialogue-based self-assessment tasks have been well studied in the context of spoken dialogue systems, it seems reasonable to examine the feature sets employed in these tasks as a starting point. Unfortunately, several of the feature sources employed by spoken dialogue systems are not available to us for text message autocorrection. Most notably, any features that draw information from prosody or speech recognition performance are clearly inapplicable, as we are no longer dealing with speech. And as we are concerned with online assessment we cannot make use of any features that must be manually labeled.

Beyond this, differences in intended task and behavior between this and previous work eliminate other potentially useful feature sources. There are two main task-related differences. First, in spoken dialogue systems the domain and goal of the dialogue are highly constrained. A given dialogue system is designed for a specific task. This means that there is only a small subset of dialogue behaviors that a cooperative dialogue participant will engage in. For instance, if the goal of the system is to book a train ticket (Litman and Pan, 2002), the system can assume that the user's dialogue behaviors will pertain to this task, and not stray into discussions of meteorology or other unrelated domains. This allows the dialogue system to make task-specific design decisions such as employing restrictive grammars and frame filling techniques, the success of which can give it good indications of whether a problematic situation has arisen. In contrast, a text message autocorrection system does not have this luxury, because the user's input is not contrained to a single known domain.

The second differentiating point is that while spoken dialogue systems are human-computer dialogues, text message dialogues are primarily human-human. As human users understand that current generation dialogue systems have limitations, they may be apt to simplify their dialogue contributions when interacting with a computer system, further constraining the

task. Additionally, while the system is a dialogue participant in the spoken dialogue case, it is merely a dialogue helper in the autocorrection case. Being a dialogue participant allows the system to have some control over, and a reasonable understanding of, the dialogue state. It also helps the system associate user actions with task-specific dialogue acts. None of this information is easy obtainable by a dialogue aid that is not a direct participant.

A corollary of these facts is that it is harder for a pure dialogue aid to understand when a dialogue behavior exhibited by the system is in response to a system action. For instance, if in a spoken dialogue system we detect that the user is angry, we can take this as a reasonably strong indication that they are angry at the actions of the system. Conversely, there are many reasons that a user might express anger in a text message dialogue (angry at the other dialogue participant, venting about a completely unrelated situation), most of which are unrelated to the performance of the autocorrection system itself. In order to detect problematic situations in the autocorrection case, we must see some evidence that the dialogue behavior is directed at the system.

So what does this leave us with? Thankfully, the most relevant source for information about whether a problematic situation has arisen is the actual *content* of the dialogue, which we still have full access to.

In order to build a successful classifier, we must devise a feature set that is able to differentiate between problematic and unproblematic instances. Examining Figure 5.1, we see that there are a few dialogue behaviors that are associated with incorrect autocorrection attempts. While in unproblematic dialogues users are able to converse freely, in problematic dialogues users must spend dialogue turns reestablishing common ground[5]. These attempts to establish common ground manifest themselves in two primary ways in the dialogue: as confusion and as attempts to correct the mistake.

Autocorrection mistakes can result in a considerable amount of misunderstanding, as

[5]The notion of common ground refers to the shared set of beliefs that two dialogue participants have about the world. The process used by the participants to establish common ground is called grounding. For a more detailed explanation, see Clark (1996).

(a)

(b)

Figure 5.3: Examples of autocorrection mistakes causing confusion in the reader.

the message sent to the reader is not what the writer thinks that they have written. This confusion can manifest itself in a variety of ways. Figure 5.3 gives a few examples. In Figure 5.3a, the attempt to type *garage* is autocorrected to *grave*. Since there is no contextually salient grave, this confuses the reader, who replies "What grave?????". The use of multiple question marks helps the user express their overall confusion. The example in Figure 5.3a gives an example of a different approach to expressing confusion. In this example the reader responds to the autocorrection mistake by explicitly inquiring about the meaning of the term by asking "what does pickle up mean". However, in this case the user chose not to use punctuation, which is common in casual written dialogue. Thus, we can not rely solely on the presence of question marks to indicate confusion.

In order for common ground to be established, both participants must know the intended

meaning of the word that was mistakenly autocorrected. Thus, another telltale sign of a problematic instance is correction dialogue, where the user explicitly mentions their intended word. Each of the screenshots in Figures 5.1 and 5.3 gives an example of the autocorrected user stating their intended word, using a variety of methods. Of particular note is Figure 5.1a in which the user uses an asterisk to highlight the intended word, a common technique among text message users.

A common order to events can be observed in the grounding behavior of participants when a problematic autocorrection occurs. The first thing to appear in the grounding dialogue must be the correction attempt itself, as prior to this the dialogue is error free. The next message in the dialogue, if it is generated by the reader of the mistaken correction, is often an expression of confusion. The next message generated by the autocorrected user is often a statement of their intended word. Both examples in Figure 5.3 follow this basic format.

There are, of course, exceptions. In some cases common ground could be reestablished without any explicit statements of confusion. If the autocorrected user realizes the mistake quickly, they may correct their mistake before any confusion can occur. Additionally, although confusion is typically expressed by the reader of the mistake and correction is usually done by the writer of the mistake, both may be done by either parties. For instance, if the reader responds thinking that the correction mistake was written as intended, the writer may express their confusion. Similarly, if the reader is able to guess from the context that a mistake has occurred, they may attempt to correct it themselves (e.g., "I think you meant to type..."). Nonetheless, if we can identify cases that follow the common format, this can give us high confidence that the correction attempt was erroneous.

Using these intuitions, we can devise a feature set that will help us differentiate between problematic and unproblematic instances. To build this feature set, a set of 300 dialogues with autocorrection mistakes from the DYAC dataset were held out for development (not used in training or considered during testing). Table 5.1 shows the feature set. The table uses the term corrected user to refer to the user for which autocorrection was performed;

the other user is referred to as the uncorrected user. All features shown only consider the dialogue after the correction attempt. The feature set is divided into 3 categories of features, based on their intent.

Confusion detection features are those features that are designed to explicitly detect messages that show that one of the dialogue participants is confused. As mentioned, one cursory feature that could indicate confusion is the presence of a question mark. Additionally, users may often use a block of repeated punctuation to expressed their confusion, which is captured in the punctuation block feature. Otherwise, expressions of confusion are primarily expressed through certain dialogue patterns, such as "what does X mean" or "I don't understand".

Similarly, *correction detection features* are those features that are designed to detect attempts to reestablish common ground by explicitly stating the intended term for the erroneous correction. As with confusion detection features, many of the correction detection features are pattern features. One interesting phenomenon observed in our data is that users often explicitly assigned blame to the autocorrection system, so we included a feature that recorded when users make mention of autocorrection, spell checking, and other similar mechanisms. A few other features attempt to detect explicit attempts to correct the word by writing it in all capital letters or using an asterisk.

Dialogue flow features attempt to detect certain common dialogue patterns that are indicative of correction attempts. The immediate confusion feature attempts to detect the typical case in which confusion is expressed by the uncorrected user immediately after a mistaken correction has occurred. Similarly, the immediate correction feature detects instances in which the corrected user immediately realizes that a mistake has occurred and attempts to correct it. The general form feature builds on these two features to try to detect the typical dialogue form of error-confusion-correction discussed above. Additionally, we observed that autocorrection mistakes frequently appeared in the last word in a message, which was recorded by another binary feature. Finally, a word repetition feature detects how much the corrected word appears in the dialogue.

Name	*Description*
Confusion Detection Features	
QMark	True if uncorrected user's dialogue contains a question mark
Punctuation Block	True if uncorrected user's dialogue contains a sequence of ? and !
Word Pattern	True if any message contains confusion word patterns such as "what does X mean"
Correction Detection Features	
All Caps	True if word in corrected user's dialogue is written in all capital letters
*	True if dialogue contains word followed or preceded by *
Quick Correction	True if message after correction attempt contains only one word
Explicit Blame	True if dialogue contains mention of "spell checking", "autocorrect", etc.
Word Pattern	True if corrected user's dialogue contains correction word patterns such as "I meant X"
Dialogue Flow Features	
Immediate Confusion	True if first message by uncorrected user contains a confusion feature
Immediate Correction	True if first message by corrected user contains a correction feature
General Form	True if Immediate Confusion & immediate correction & order follows general form
Last Word	True if correction attempt is the last word in message
Word Repetition	The number of times the correction attempt word appears in the dialogue

Table 5.1: Feature set for differentiating between problematic and unproblematic autocorrection attempts

Features	Precision	Recall	F-Measure	Accuracy
All Features	.861	.751	.803	.790
-Confusion	.857	.725	.786	.775
-Clarification	.848	.676	.752	.747
-Dialogue	.896	.546	.679	.707
Baseline	.568	1	.724	.568

Table 5.2: Feature ablation results for identifying autocorrection mistakes

Using these features we can build a model to differentiate between problematic and unproblematic instances. Once we have identified which instances are problematic, we can then consider the problem of discovering the term the user intended to type.

Evaluation

To build our classifier we used the SVMLight[6] implementation of a support vector machine classifier with an RBF kernel. To ensure validity and account for the relatively small size of our dataset, evaluation was done via leave-one-out cross validation.

Results are shown in Table 5.2. A majority class baseline is given for comparison. The table includes values for overall accuracy and precision, recall, and F-measure for the positive class (problematic instances). As shown, using the entire feature set, the classifier achieves accuracy of 0.790. It also produces above baseline precision of 0.861, while still producing recall of 0.751.

Although standard F-measure is reported, it is unlikely that precision and recall should be weighted equally. Because one of the primary reasons we may wish to detect problematic situations is to automatically collect data to improve future performance by the autocorrection system, it is imperative that the data collected have high precision in order to reduce the amount of noise present in the collected dataset. Conversely, because problematic situation detection can monitor a user's input continuously for an indefinite period of time in order

[6]Version 6.02, http://svmlight.joachims.org/

to collect more data, recall is less of a concern.

A feature ablation study was performed to study the effect of each feature source. The results of this study are included in Table 5.2. For each run, one feature type was removed and the model was retrained and reassessed. As shown, removing any feature source has a relatively small effect on the precision but a more substantial effect on the recall. Confusion detection features seem to be the least essential, causing a comparatively small drop in performance when removed. Removing the dialogue features hurts accuracy the most and results in the greatest drop in recall, returning only slightly above half of the problematic instances. However, as a result, the precision of the classifier is higher than when all features are used.

5.4.2 Identifying the Intended Term

Note that one purpose of the proposed self-assessment is to collect information online and thus make it possible to build better models. In order to do so, we need to know not only whether the system acted erroneously, but also what it should have done. Therefore, once we have extracted a set of problematic instances (and their corresponding dialogues), we must identify the term which the user was attempting to type when the system intervened. Lets assume that the classification task in the previous section was successful, and we know which dialogue snippets contain a problematic instance. Since the system initiates the autocorrection itself, we can also assume that we know the attempted correction within a problematic dialogue snippet. We can then formulate the problem as follows: given a correction attempt c and the sourounding problematic dialogue D, find the word $w_i \in D$ such that w_i is the term the user intended to type when c was produced by the autocorrection system. In other words, given a set of erroneous correction attempts, EC the problem becomes: for every erroneous correction $c \in EC$, identify $w \in D$ such that $w = Intended(c)$.

However, unlike a typical classification problem, we are not trying to directly differentiate between the positive and negative instances, but find the single instance that is most likely

to be intended. We can do this because we have some external knowledge about the problem that is not inherently encoded in the typical classification problem: we know that there is one and only one single instance that is the intended term. We thus model this as a ranking task, in which all $w \in W$ are ranked by their likelihood of being the intended term for c. We then predict that the top ranked word is the true intended term. This method was chosen because running a typical classification method would result in a substantially unbalanced classification problem, which may lead to poor overall results.

One additional point should be mentioned. Because a mistake must happen before it is corrected, we do not have to consider all $w_i \in D$ as plausible intended terms; we can instead restrict our space to only include those words that follow the message that the mistake occurred in.

Feature Set

A diverse feature set was assembled to support the above processing, consisting of five different feature sources: contextual, punctuation, word form, similarity, and pattern features. This feature set was crafted from an examination of the development data. Several of the features are related to those used in the initial classification phase. However, unlike the classification features, these feature focus on the relationship between the erroneous correction c and a candidate intended term w. Table 5.3 shows our feature set.

Contextual features capture relevant phenomena at the discourse level. After an error is discovered by a user, they may type an intended term several times or type it in a message by itself in order to draw attention to it. These phenomena are captured in the *word repetition* and *only word* features. Another common discourse related correction technique is to retype some of the original context, which is captured by the *word overlap* feature ("pickle up"; "pick me up"). The *same author* feature captures the fact that the author of the original mistake is likely the one to correct it, as they know their true intent.

Punctuation is occasionally used by text message writers to signal a correction of an

Name	*Description*
Contextual Features	
Word Overlap	True if the same word adjacent to both c and w_i in the dialogue
Only Word	True if w_i is the only word in a message
Same Author	True if first occurrences of c and w_i produced by the same person
Word Repetition	Number of times correction attempt word appears in the dialogue
Punctuation Features	
*|!|=|?|"|‘	True if dialogue contains w_i adjacent to mark (separate features)
Word Form Features	
All Caps	True if w_i is written in all capital letters
Letter Repetition	True if w_i contains any letter repeated more than twice in a row
Out of Vocab.	True if w_i is not a known vocabulary word
Similarity Features	
Edit Distance	Minimum edit distance between w_i and c
Pattern Features	
Meant	True if dialogue contains pattern: meant to {say|write} w_i
Supposed	True if dialogue contains pattern: supposed to {say|be} w_i
Should	True if dialogue contains pattern: should {say|read} w_i
Wrote	True if dialogue contains pattern: {wrote|typed} w_i
X=Y	True if dialogue contains pattern: c=w_i
X not Y	True if dialogue contains pattern: w_i not c

Table 5.3: Feature set for identifying the intended term for a given erroneous correction

earlier mistake, as in the asterisk example given in Figure 5.1a. *Punctuation features* attempt to capture the use of punctuation as a marker of a correction mistake. The presence of several punctuation marks were examined, as shown in the table. Each punctuation mark is represented by a separate feature.

Word form features capture variations in how a candidate word is written. One word form feature captures whether a word was typed in all capital letters, a technique used by text message writers to add emphasis. Two word form features were designed to capture words that were potentially unknown to the system, out-of-vocabulary words and words with letter repetition (e.g., "yaaay"). Because the system does not know these words, it will consider them misspellings and may attempt to change them to an in-vocabulary term.

Our *similarity feature* captures the character level distance between a word changed by the system and a candidate intended word. The normalized levenshtein edit distance between the two words is used as a measure of similarity. This gives us a rough idea of the chance that one could be confused for the other. Although more sophisticated distance measures have been proposed (McCallum et al., 2005), they require more training data than we have available in this domain. However, one aspect of the task does necessitate one modification to the typical edit distance metric: we only consider the edit distance up to the length of w_i, to account for the fact that the mistake could have been autocompletion, not autocorrection. Thus, the raw edit distance between *arthritic* and *wet* is 2, not 8.

Pattern features attempt to capture phrases that are used to explicitly state a correction. These include phrases such as "(I) meant to write", "(that was) supposed to say", "(that) should have read", "(I) wrote", etc.

Evaluation

To find the most likely intended term for a correction mistake, we rank every candidate word in W and predict that the top ranked word is the intended term. The ranking mode of SVMlight was used to train the ranker. By thresholding our results to only trust predictions

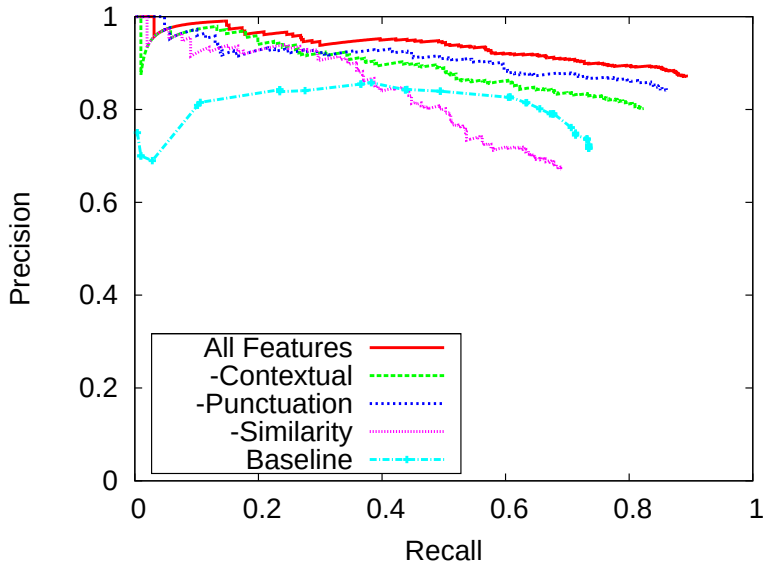


Figure 5.4: Precision-recall curve for intended term selection, including feature ablation results

in which the ranker reported a high ranking value for the top term, we are able to examine the precision at different recall levels. This thresholding process may also allow the ranker to exclude instances in which the intended term does not appear in the dialogue, which are hopefully ranked lower than other cases. As before, evaluation was done via leave-one-out cross validation.

Results are shown in Figure 5.4. As a method of comparison we report a baseline that selects the word with the smallest edit distance as the intended term. As shown, using the entire feature set results in consistently above baseline performance.

As before, we are more concerned with the precision of our predictions than the recall. It is difficult to assess the appropriate precision-recall tradeoff without an in-depth study of autocorrection usage by text messagers. However, a few observations can be made from the precision-recall curve. Most critically, we can observe that the model is able to predict the intended term for an erroneous correction with high precision. Additionally, the precision

stays relatively stable as recall increases, suffering a comparatively small drop in precision for an increase in recall. At its highest achieved recall values of 0.892, it maintains high precision at 0.869.

Feature ablation results are also reported in Figure 5.4. The most critical feature source was word similarity; without the similarity feature the performance is consistently worse than all other runs, even falling below baseline performance at high recall levels. This is not suprising, as the system's incorrect guess must be at least reasonably similar to the intended term, or the system would be unlikely to make this mistake. Although not as substantial as the similarity feature, the contextual and punctuation features were also shown to have a significant effect on overall performance. Conversely, removing word form or pattern features did not cause a significant change in performance (not shown in Figure 5.4 to enhance readability).

5.4.3 An End-To-End System

In order to see the actual effect of the full system, we ran it end-to-end, with the output of the initial erroneous correction identification phase used as input when identifying the intended term. Results are shown in Figure 5.5. The results of the intended term classification task on gold standard data from Figure 5.4 are shown as an upper bound.

The green line shown on the graph represents the end-to-end case. As expected, the full end-to-end system produced lower overall performance than running the tasks in isolation. The end-to-end system can reach a recall level of 0.674, significantly lower than the recall of the ground truth system. However, the system still peaks at precision of 1, and was able to produce precision values that were competitive with the ground truth system at lower recall levels, maintaining a precision of above 0.90 until recall reached 0.396.

The end-to-end system assumes that we must first identify problematic instances before we are able to identify the intended term. An alternative approach would be to rely solely on the ranking value to make this decision. That is, rather than performing an initial step of

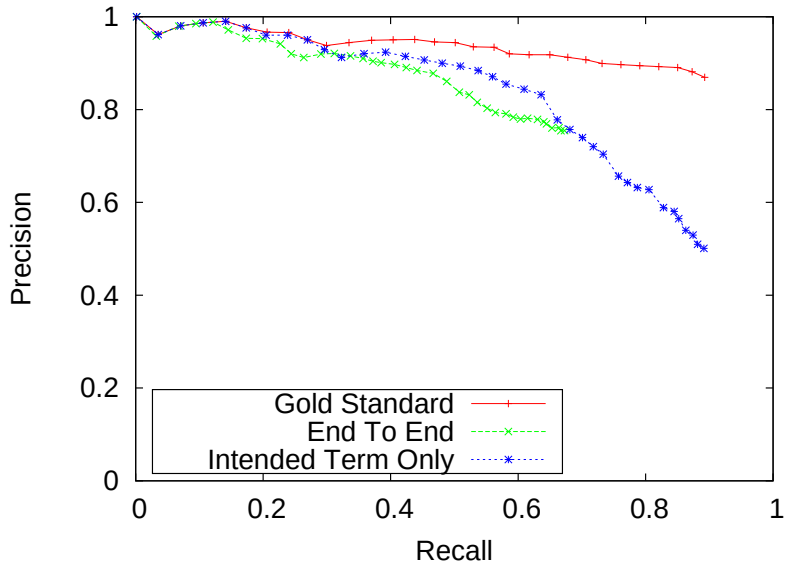


Figure 5.5: Precision-recall curves for an end-to-end system and a system that performs no initial problematic-unproblematic differentiation

removing unproblematic instances prior to finding intended terms, we simply attempt to find intended terms over all data instances, both problematic and unproblematic. If our feature set is sufficiently discriminative, the cases that we are most confident about the intended terms will be cases which are most likely to be problematic. Thus, we can implicitly perform the needed problematic-unproblematic differentiation without the initial step by thresholding the ranking value as before.

The blue line in Figure 5.5 shows the results of this type of intended term identification. As shown, the performance stays consistent with the gold standard performance until recall of around 0.3. It then slowly looses ground until recall of about 0.65, at which point overall performance declines sharply. Since both problematic and unproblematic cases are included, the performance is understandably poor at the highest recall levels where little to no thresholding is done.

The implicit method consistently outperforms the two stage end-to-end method. As ex-

pected, strong indications that some word in the discourse was the intended term translated into strong indications that the correction attempt was problematic. In contrast, the noise introduced by the problematic-unproblematic decision in the end-to-end case hurt performance by removing cases that the intended term identification step could likely have identified as problematic. This result is intuitive, as identifying problematic cases that the intended term identification step is unable to handle does not help the overall performance. This means that the only potential gain from problematic-unproblematic differentiation is pruning out unproblematic cases that the intended term predictor could erroneously suggest have an intended term. Since most unproblematic cases are unlikely to have a term that appears to be a strong intended term candidate, the original differentiation step must be able to perform with high accuracy in order to be more useful in intended term differentiation than implicit differentiation.

The relative success of the implicit differentiation method highlights an important point about the two self-assessment tasks: they are not independent. In particular, the presence of a good intended term candidate is likely to be a strong indication that a given correction attempt was erroneous. In the original problematic-unproblematic differentiation task, some features were designed to at least make a rudimentary attempt to identify whether a strong intended term candidate was present. However, the more focused intended term identification task gives a better sense of the overall likelyhood that an intended term is present. As such, rather than using the output of the problematic-unproblematic case to inform the intended term selection, we could potentially use the output of the latter to inform the former.

To do so, the intended term identification task must be run on all data, as in the implicit method discussed above. By taking the top ranked value for each case (again, as above), we get some indication of the likelyhood that some term in the discourse is the intended term for the user's autocorrection attempt. This can then be used as a feature in the problematic-unproblematic differentiation task.

Table 5.4 shows the results of including this feature, as compared to the results without

Features	Precision	Recall	F-Measure	Accuracy
-Intended	.861	.751	.803	.790
+Intended	.894	.835	.863	.850

Table 5.4: Results for identifying autocorrection mistakes with and without intended term ranking values as a feature

this feature from Table 5.2. Including the intended term ranking value as a feature has a significant effect on overall performance. Overall accuracy rises from 79% to 85% and absolute increase in performance of 6%. Including the feature also improves both precision and recall.

5.5 Discussion

When an autocorrection system attempts a correction, it has perfect knowledge of the behavior of both itself and the user. It knows the button presses the user used to enter the term. It knows the term it chose as a correction. It knows the surrounding context; it has access to both the messages sent and received by the user. It has a large amount of the information it could use to improve its own performance, if only it were able to know when it made a mistake. The techniques described here attempt to address this critical system assessment step. Users may vary in the speed and accuracy at which they type, and input on small or virtual keyboards may vary between users based on the size and shape of their fingers. The self-assessment task described here can potentially facilitate the development of autocorrection models that are tailored to specific user behaviors.

Here is a brief outline of how our self-assessment module might potentially be used in building user-specific correction models. As a user types input, the system performs autocorrection by starting with a general model (e.g., for all text message users). Each time a correction is performed, the system examines the surrounding context to determine whether the correction it chose was actually what the user had intended to type. Over the course

of several dialogues, the system builds a corpus of erroneous and non-erroneous correction attempts. This corpus is then used to train a user-specific correction model that is targeted toward system mistakes that are most frequent with this user's input behavior. The user-specific model is then applied on future correction attempts to improve overall performance. This monitoring process can be continued for months or even longer. The results from self-assessment will allow the system to continuously and autonomously improve itself for a given user.

In order to learn a user-specific model that is capable of improving performance, it is important that the self-assessment system provides it with training data without a large amount of noise. This suggests that the self-assessment system must be able to identify erroneous instances with high precision. Conversely, because the system can monitor user behavior indefinitely to collect more data, the overall recall may not be as critical. It might then be reasonable for a self-assessment system to be built to focus on collecting high accuracy pairs, even if it misses many system mistakes. Although a full examination of this tradeoff is left for future work which may more closely examine user input behavior, the results presented here show promise for collecting accurate data in a timely manner.

One final point should be mentioned in relation to the data distribution. Although we tested our system on a balanced dataset with roughly even numbers of problematic and unproblematic instances, it is likely that an autocorrection system will get many more instances correct than wrong, leading to a data distribution skewed in favor of unproblematic instances. This suggests that the evaluation given here may overestimate the performance of a self-assessment system in a real scenario. Although the size of our dataset is insufficient to do a full analysis on skewed data, we can get a rough estimate of the performance by simply counting false positives and false negatives unevenly. For instance, if the cost of mispredicting a unproblematic case as problematic is nine times more severe than the cost of missing a problematic case, this can give us an estimate of the performance of the system on a dataset with a 90-10 skew.

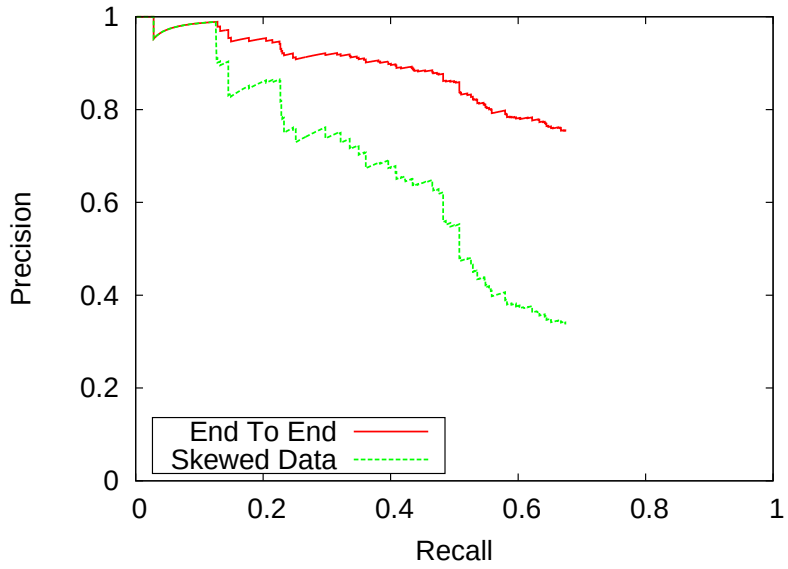


Figure 5.6: Precision-recall curve for the end-to-end system on data with a 90-10 unproblematic-problematic split

We examined the 90-10 skew case to see if the procedure outlined here was still viable. Results of an end-to-end system with this data skew are shown in Table 5.6. As expected, the results on the skewed data were consistently lower than in the balanced data case. The skewed data system can keep performance of 90% or better until it reaches 13% recall, and 85% or better until it reaches 22%. These results suggest that the system could still potentially be utilized. However, its performance drops off steadily, to the point where it would be unlikely to be useful at higher recall levels.

CHAPTER 6
Autocorrection Personalization

The work of the previous chapter established that there are a few viable methods available to do self-assessment of autocorrection performance. This chapter examines how this can be used to personalize and adapt autocorrection systems. Like Chapter 4, the discussion will focus primarily on a single means of personalization, in this case vocabulary acquisition. However, there are other possible ways to adapt and personalize autocorrection behavior, such as by adjusting the frequency of autocorrection output based on its performance or using the correction mistake-intended term pairs as training data to improve the overall model. A brief discussion of these other adaptation methods is given near the end of the chapter.

Before we begin, it should be noted that the dataset used in these experiments was necessarily built from messages aggregated from many anonymous users. As such, it does not directly allow us to pinpoint a single user's behavior as we would in a typical personalization setting. Thankfully, everything we talk about in this chapter with regards to personalization applies to adaptation in general. They apply to any set of data that we wish to examine for patterns and tailor our output to. It is not critical whether the data we examine comes from one user or many, as the methods we discuss will pertain to both.

6.1 Personalization Via Vocabulary Acquisition

Let's again consider Figure 5.1a, in which the system mistakenly corrects the user's intended term *gooooood* to *hooklike*. When the system takes the user's ambiguous input it attempts to map it to the correct term, but there is a fundamental problem here: the non-standard

word form *goooooood* is not in the system's internal dictionary. As the correction system can only predict words that it knows, the chance that the system will pick the correct term here is not only poor, it is exactly zero. In other words, in any case where the intended term is not in the dictionary the system can not get the answer correct. The best it could do is to simply not make a correction at all, meaning it is at least as bad as if no autocorrection system were available.

If we can automatically expand the size of the dictionary that the system uses online, we can avoid these no-win cases. A first solution to this problem might be to simply include a larger dictionary when the product is shipped. While this may help in some rudimentary cases, in most of the instances this solution is both problematic and inadequate. Many of the terms that cause autocorrection problems are slang, abbreviations, and non-standard word forms. These words tend to mean something to some users and not to others, so adding these to the vocabulary could cause problems for users who do not use these terms. Similarly, increasing the size of the dictionary increases the number of candidates for the autocorrection system to choose from, making its existing choices harder. It is not enough to just add words to the dictionary, it must be done intelligently.

Even if a universal expansion of the dictionary could prove fruitful, it would very quickly become obsolete as new words entered the lexicon. The ideal solution would be to dynamically tailor the vocabulary to the individual user. For a given user who is prone to heavy use of slang and non-standard word forms the system could continually learn and adapt to their lexicon, without corrupting the lexicon of another user who does not use or understand these terms. The system could also adapt to specialized vocabulary used by only a small subset of users, such as technical terms, names of local landmarks, and neologisms of their own creation. As time went on the vocabulary would grow, but it would only grow with words that were in the active lexicon of the current user. This would allow the autocorrection system to have a chance at correctly identifying the user's attempts to type the term, without interference from terms that do not exist in the user's lexicon.

Although it has garnered little attention from the research community, some simple methods of dictionary expansion have been implemented in current generation mobile devices. For instance, the Apple iPhone has employed a method that adds words typed as web browser search queries into the dictionary (Sadun, 2009). Although it does allow for some vocabulary expansion, this method is simplistic and likely to both include a fair amount of false positives and miss a large portion of legitimate words, as search query terms are often quite different from texting language. Additionally, the propensity of some users to employ web search for spell checking will further increase the false positive rate.

Another straightforward method to expand the dictionary would be to add words that the user has typed several times. While this method can be reasonable, it has two potential drawbacks. The first drawback is that it can be sabotaged by the existing autocorrection mechanism. For instance, let's examine a user whose own lexicon includes the slang term *pwn*[1]. When the user attempts to type this term into a message, it has a high chance to be automatically corrected to the common term *own*. If the user does not see and correct this mistake the system will not realize that they were attempting to type an out-of-vocabulary term, and therefore does not count it as a new word. If the user catches this mistake and corrects it, we have observed a single use of the term. This brings us to the second potential drawback of such a system: it learns new vocabulary fairly slowly. Consider a system that adds a word to the dictionary after it has seen the user type it 10 times. Before this system is able to learn the word, the system has at least 10 chances to erroneously correct the word. And if the user does not correct the system's mistake every time, they may end up being erroneously autocorrected even more than 10 times.

Thankfully, the system-assessment mechanisms outlined in this work gives us an alternative (or supplemental) method of vocabulary expansion. If we can accurately identify the intended term for a correction mistake, we have identified a case that we can confidently say

[1]This term originated as an unintentional misspelling of the word *own* in online video gaming communities, but eventually entered the lexicon as its own (roughly synonymous) entry.

the user wished to type. In other words, we have identified a term we can be confident is part of their lexicon. When we see an erroneous correction and identify the intended term, we can immediately add that term to the dictionary (if it is not there already); no need to get it wrong another few times before we catch on.

The rest of this section will examine the utility of using the system-assessment methods outlined previously to personalize and dynamically expand the autocorrection system's internal dictionary. However, before we are able to evaluate this method, we must take a new look at our dataset and examine it from the perspective of vocabulary expansion.

6.1.1 Classifying Out-Of-Vocabulary Terms

It is a reality of any unconstrained language input task that many tokens input by a user will fall outside of the system's dictionary. The general reasons for this are touched on above. The language is simply too dynamic, expansive, and user and domain specific to expect any dictionary to catch everything, and expanding the general dictionary to include obscure and uncommon tokens can cause as many problems as it solves. Of course, not every token typed by the user should be considered a legitimate term; some are produced by cognitive or typographic errors. This section will look at the different types of tokens that tend to be missed by autocorrection dictionaries and discuss which ones should and should not be added to an expanded dictionary.

In the broadest terms, we wish to differentiate between two classes of out-of-vocabulary terms: terms that we legitimately wish to enter into the dictionary and those we do not. If what the user typed results in an unintentionally produced typographical error, then clearly we should not include this entry in the dictionary. For the most part, any term in which the user truly intended to type should be included in the dictionary. The one exception to this is cognitive errors. Although these may correspond to true entries in the user's lexicon, they conflict with the standard form of the word. Since the difference in spelling is erroneously produced (as opposed to intentional use of non-standard word form for emphasis, emotion,

etc.), the correct response for the system would be to correct these cases to the standard form, not put them in the dictionary as new entries.

Although the two-class distinction will be sufficient for classification purposes, we can get a better understanding of the problem by further subdividing the classes. An examination of the out-of-vocabulary (OOV) terms in our dataset led to the following classification scheme:

- OOV terms that should be added to the dictionary:
  - Neologisms, proper nouns, technical terms, and other standard word forms missed by the dictionary. Examples: *staycation, Netflix, Yoda, pilates, sparticipation*
  - Texting slang and other intentionally produced non-standard word forms. Examples: *yeahhhh, goooooood, FTW, FML, stickin'*
  - Onomatopoeia. Examples: *Pshh, hmm, mmhmm, bwahaha*
  - Emoticons. Examples: O-.-)O, 0_0, XD
  - Number strings and other regularly formatted input. Examples: *2nd, 1:30, 50s, 5'11*
- OOV terms that should *not* be added to the dictionary:
  - Cognitive or typographic spelling errors. Examples: *mornim, accoubt, becaise, tbumbs, verticle*
  - Word concatenation. Examples: *mightbe, nonono, loveyoutoo*
  - Gibberish and keyboard mashing.

It should be noted that some of the categories listed do not nicely fit the "add" vs. "do not add" distinction. In particular, texting slang and regularly formatted number strings are often generatively produced. That is, rather than being a single separate entry in the dictionary, there are particular rules for how to produce these terms. This makes simply adding the string to the dictionary when we see it a poor approximation of the underlying

mechanism, and one that is not able to catch the smallest amount of variation. For instance, placing *2nd* in the dictionary does not help us with small but legal variations such as *52nd*. Similarly, including *yeahhhh* in the dictionary does not help us get *yeahhh* or *yeahhhhh*.

There are two ways to think about this problem. In the one case, we are simply stuck with the dictionary representation, as the dictionary is a critical aspect of current autocorrection systems. However, if we resign ourselves to this fact then the best solution is to place these terms in the dictionary. After all, while adding *goooooood* to the dictionary may not help us recognize that small variations in the number of *o*s also result in an acceptable term, it at least can handle the single case where the user attempted to type *gooooood*. And since the intended meaning of *goooooood* or *goooood* is likely the same, placing one in the dictionary may help an instance of the other, since correcting *goooood* to *goooooood* is more desirable than correcting it to *hooklike*.

The other way to think about this problem is that we should be able to understand the generative rule that produced these cases, and then somehow incorporate this information into the autocorrection system. While incorporating them effectively requires modifications to the autocorrection system that are as of yet unclear (and not within the scope of this thesis), a fair amount of work has been done in trying to identify these rules. The field of normalization takes non-standard word forms and maps them onto their standard forms (Sproat et al., 2001) and in recent years a substantial amount of research has focused on the normalization of text messages specifically (Choudhury et al., 2007; Kobus et al., 2008; Cook and Stevenson, 2009). Of course, mapping the non-standard forms to the standard form is not the ideal solution, because the non-standard form often carries additional information that the user wishes to express (Baldwin and Chai, 2011). Nonetheless, the normalization process may help us to understand that the non-standard form was intentional, and therefore should be a viable candidate for the autocorrection system to correct to. In order for this to happen, we still need to think of these non-standard forms as "good" cases, cases that the autocorrection system will consider as potentially correct terms. As such, the inclusion of

these terms in the "should be added to the dictionary" category is the correct one.

Now that we have a sense of the distinction between types, we can examine the distribution of them in our data. To determine if a term is out of the vocabulary, it is necessary to have a vocabulary to begin with. The dictionary from the CMU pronouncing dictionary[2] was used for this purpose. The CMU pronouncing dictionary is a dictionary of over 125,000 words in North American English pulled from a variety of sources. It was chosen because it has reasonably good coverage of proper names and other terms that would appear in a carefully collected dictionary, and it thus gives us a competent starting point for an unpersonalized dictionary.

All out-of-vocabulary tokens in our dataset were annotated with the appropriate token type to help us understand their distribution. Note that we are concerned with the number of unique tokens not the number of total tokens of each type. Since we want to observe the number of dictionary entries missed and not the total number of words, we are most concerned with the number of unique tokens. A small number of tokens whose intent could not be confidently predicted were marked as unknown and removed from consideration. This left a total of 1087 unique tokens that were not in the vocabulary.

The type distribution is shown in Table 6.1. As shown, OOV terms that should be in the dictionary made up about 65% of all tokens, suggesting that the majority of out-of-vocabulary words are actually not mistakes at all. These words were split relatively evenly between non-standard word forms and technical terms/other OOV standard forms. Onomatopoeia also made up a sizable portion of the "good" OOVs. The other types of good input, emoticons and number strings, appeared infrequently.

Spelling mistakes made up the majority of the "bad" OOVs, accounting for over three fourths of all OOV terms that should not be added to the dictionary. Most of the rest of the bad OOVs were concatenations. Instances of pure gibberish or keyboard mashing were understandably rare.

[2]http://www.speech.cs.cmu.edu/cgi-bin/cmudict

Type	Unique Tokens	Percentage of Category	Percentage of All
Technical Terms/Neologisms	269	36.7%	24.7%
Non-standard word forms	281	38.4%	25.9%
Onomatopoeia	158	21.6%	14.5%
Emoticons	9	1.2%	0.8%
Other	15	2.0%	1.4%
Spelling Errors	265	74.6%	24.4%
Concatenation	80	22.5%	7.4%
Gibberish	10	2.8%	0.9%

Table 6.1: Distribution of out-of-vocabulary words in the dataset

At a high level, the data in Table 6.1 gives us some relevent information about the difficulty of identifying words that should be included in a personalized dictionary. Firstly, it shows that identifying the two classes is likely to be non-trivial. The coarse "include or do not include" distinction hides many differences that could make differentiating between the two classes harder. Additionally, the relative balance between the classes makes simple baselines (such as assuming all OOV words should be in the dictionary) unappealing as they would be too noisy to be practically useful.

6.1.2 OOV Differentiation

As we have seen, erroneous autocorrection attempts are often followed by predictable dialogue patterns. In particular, after a correction mistake has occurred, the dialogue participants engage in grounding dialogue that establishes what term the autocorrected user had intended to type. The autocorrection system often makes these mistakes because it does not have the intended term in the vocabulary. Thankfully, we can use the knowledge we gained from the self-assessment mechanism described in this chapter to ensure that these mistakes do not happen again. Since we know that the intended term is what the user really intended to write, we know it is in their lexicon. If we encounter a correction mistake and identify that the user's intended term is out of the dictionary, we know that we should add it.

System	Precision	Recall
Majority Class Baseline	0.65	1.0
Frequency Baseline	1.0	0.02
Self-Assessment	0.87	0.16

Table 6.2: Results of self-assessment based vocabulary acquisition

As such, vocabulary acquisition via self-assessment requires very little additional overhead. First, we identify correction mistakes and their corresponding intended term as normal. At this point, if the intended term is in the vocabulary we gain no further insight. If the term is out of the vocabulary, we assume that it is in the user's lexicon and add it immediately to our dictionary, even if this is our first encounter with the term.

This procedure was run on the dataset to assess its performance. The end-to-end system from Section 5.4.3 was used for self-assessment. No threshold was used in predicting self assessment; it was always assumed that the top term predicted by the system was the intended term of the user. Results are shown in Table 6.2. Two baselines are given for comparison. The first baseline is a majority class predictor that predicts that all OOV words should be added to the dictionary. The other baseline is a frequency-based baseline that adds all OOV words that appear more than ten times in the dataset to the dictionary.

As before, the balance between precision and recall is not equal. We would certainly like to have high recall so that we may adapt more quickly and substantially to the user's lexicon. However, in order for our adaptation to be of any use, we must be reasonably confident that the word we add to the dictionary is actually in the user's lexicon. This means that precision should again be considered more critical than recall, but the tradeoff is not as stark as before.

As expected, the table shows that the majority class baseline does not have competitive performance; the precision is too low to be practically useful. On the other hand, the frequency baseline has perfect precision, making it plausibly useful. Unfortunately, the frequency baseline suffers from poor recall, capturing only 2% of all OOV terms that were in the user's lexicon. What's more, since the frequency baseline captured only those words

that appeared quite often, most of the words it captured are cases that would likely be in a dictionary that was built to be run on text messaging, such as the common texting language terms *lol* and *lmao*. This suggests that while a frequency-based method may help to find out of vocabulary words that should be in the dictionary, it is unlikely to help for all but the most common terms.

In contrast, the self-assessment system is able to capture a larger and more varied swath of the "good" OOV words. It does this at a cost to precision. However, the precision is still high enough that the self-assessment mechanism can be useful for dictionary expansion. Because the system will occasionally place an incorrect term in the dictionary, the system should have some sort of mechanism to remove personalized terms from the dictionary as well. One method employed by current generation systems is to simply let terms expire if the user hasn't used them in a while. Including a system such as this would help to mitigate the damage caused by adding an incorrect term to the dictionary. And of course further dictionary expansion can be achieved by using the self-assessment method in tandem with other methods.

We can get a better understanding of where the self-assessment system faltered by looking at its incorrect predictions. The self-assessment system incorrectly predicted that the following terms should have been placed in the dictionary:

- spelling mistakes: hanfung, viber, xray, vomitting, scardey, verticle, gregio, walhberg
- concatenation: hungover, yesyes, lightyear, redbull, poptarts, qtips

Since the user's intended term is by definition what they intended to type, it must be in their lexicon. This leave two cases where the self-assessment system can fail: incorrectly identifying the intended terms and cognitive errors on the user's part. Most of the system's mistakes can be understood to fall into one of these categories even without checking to see if the system truly did pick the incorrect term. Cases such as *viber* and *hanfung* are incorrect predictions by the system, while the majority of the other cases result from cognitive

mistakes. For the cognitive errors, the self-assessment mechanism is working as intended; it is the user who is mistaken, not the system. Of course, these are the exact cases where an autocorrectly system is supposed to help the user. Many of the cognitive mistakes, particularly those involving concatenation, may debatably be seen as not a mistake at all, depending on how liberal we are willing to be with our understanding of what should be placed in the dictionary. For instance, *redbull*, *poptarts*, and *qtips* are all mistaken attempts at spelling certain proper nouns (Red Bull, Pop-Tarts, and Q-tips, respectively). In some cases these alternative spellings may be used as frequently as the correct versions. Since the correct spelling of these terms is also likely to be out of the dictionary, placing the alternative spelling in the dictionary would at least stop the system from changing the term entirely next time it encountered it.

Now that we have examined the performance of the self-assessment system at vocabulary acquisition we can examine how this can potentially translate into improved autocorrection performance. We have established that an autocorrect system cannot possibly correct to the intended term if the intended term is outside of the dictionary. Given this, we can compute an upper bound on the performance of an autocorrection system on our dataset by observing what percentage of the intended terms are actually in the vocabulary. Of the 728 autocorrection attempts in the DYAC data set, 541 of the intended terms are in the vocabulary. This means that with the current, non-personalized dictionary the system can reach an accuracy of 74.3% on this dataset, *at best*. The self-assessment mechanism is able to identify and include 125 of the 187 out-of-vocabulary terms. Adding these terms to the dictionary changes this upper bound to 91.5%, an increase of an absolute 17.2% over the system with no vocabulary expansion.

Of course, since the DYAC dataset includes only instances in which autocorrection failed, this dataset is likely to represent cases that are more difficult overall than the average autocorrection case. Nonetheless, the gains on this dataset are stark enough to suggest that self-assessment based vocabulary expansion is a relable means of personalization-based

performance improvement.

6.2 Other Personalization Methods

Vocabulary acquisition is be no means the only method by which self-assessment mechanisms can be used to adapt and personalize autocorrection behavior. This section discusses two other potential personalization methods. Due to a lack of concrete data to study these cases in depth, we can only get a rough estimate of the ability of the current method to enable these forms of adaptation and thus only a brief analysis is given.

6.2.1 Adjusting Autocorrection Frequency Based on Performance

Users exhibit differences in vocabulary, typing style, finger size, and mindfulness of their input. As all of these things can have an effect on autocorrection performance, the autocorrection system may produce very different performance for different users. Without online self-assessment, the system is unable to notice and account for these differences. If the system is able to assess its own performance, it can understand for which users it performs poorly and for which it performs well, and adjust accordingly. Section 5.2 suggested that one such adaptation would be to simply adjust how often autocorrection is performed. If the system detects that it is performing poorly, it can decide to perform autocorrection less, only correcting in cases where it is most confident. Similarly, if it detects that it performs quite well for a given user, it may attempt to correct more often to increase its coverage.

Ideally, to show the utility of such a method we would need a large-scale comparison of this adaptation and a suitable control case where no adaptation is done. As established previously, such a dataset is currently unavailable, and extremely difficult to collect. However, there is one aspect of the process that we can get some rudimentary understanding of, and that is whether or not the current self-assessment methods are even good enough to allow the system to get a reasonable estimate of its own performance. Afterall, if the autocorrec-

tion system cannot estimate how well it is performing, it cannot make an informed decision about whether it should reduce or increase its autocorrection frequency. The autocorrection system must be able to use self-assessment to approximate the true performance. Since the first step in our self-assessment process is to decide whether or not each correction attempt is erroneous, we should be able to get an understanding of overall performance by simply looking at how frequently we predict that an autocorrection attempt failed. Of course, since the self-assessment mechanism does not have perfect performance, it is unlikely to predict entirely accurately. While we may be able to adjust for some amount of systematic error, random error in the predictions will have to be seen as noise. If there is too much random error in the self-assessment predictions, we will not be able to get an accurate prediction of true system performance.

To understand whether this style of personalization is plausible with the current system, we must investigate whether or not the current self-assessment mechanism can accurately predict autocorrection performance. To do so, we need autocorrection systems that produce varying levels of performance. This can be simulated by resampling the data from our dataset to create datasets of certain accuracies. This can be done simply by sampling a random instance from the unproblematic dataset with a probability p and from the problematic dataset with a probability of $1-p$, where p cooresponds to the accuracy we wish to simulate. Sampling is done with replacement, to create a dataset of equal size to the original dataset used in the evaluations in the previous chapter. We can then use the self-assessment mechanism to approximate the performance of the system on the resampled dataset. Because we would not know ahead of time what performance a user would exhibit, we must have the same model for each run regardless of data skew. To account for this, a model trained on the balanced data used in the previous chapter was used for all runs. This procedure can be repeated multiple times for each level of accuracy to get an understanding of the variance.

This resampling procedure was performed at accuracy intervals of 5 percentage points. To ensure that we had a good understanding of between-run variance, the resampling procedure

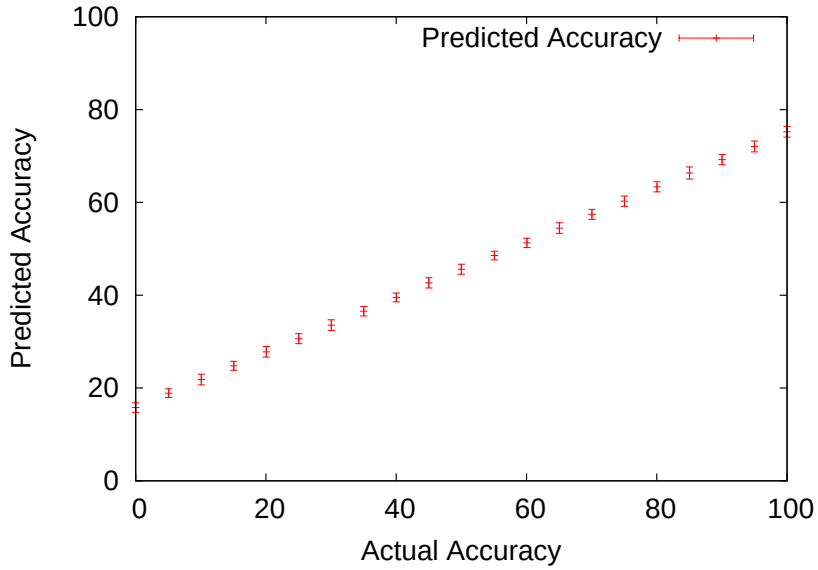


Figure 6.1: Accuracy predictions made by the self-assessment system at different levels of true accuracy.

was replicated 100 times per interval. The results are given in Figure 6.1. There are several points to observe from the figure. First, the performance of the classifier increases linearly as the true accuracy increases. This is good, as it means that the classifier is at least good enough to not be making arbitrary predictions.

Secondly, while the performance of the classifier coorelates positively with the true performance, it does not give particularly good predictions. In particular, the classifier tends to under predict true accuracy at higher accuracy levels and over predict it at lower levels. This should not be suprising, as the classifier was trained on roughly balanced data we should expect it to regress somewhat towards predicting about 50% of the data falls into each class. This is not a serious setback, as this type of error is systematic and well understood, and can thus be accounted for when making predictions, as long as some preanalysis is done of a self-assessment system before it is deployed. However, this regression does have the effect of compressing the scale over which the predictions vary, which does have implications for

our predictive system.

Because of the compressed scale, the difference between two points on the graph is smaller than the true difference. For instance, the true absolute performance difference between systems that perform at 95% and 100% accuracy is 5%, but the difference between the classifier points is 3.2%. This effects the system's ability to differentiate between smaller differences in performance. Essentially, we can be confident in a system's prediction only between intervals in which the variances do not overlap. For instance, since the error bars in the graph are generally separated from one another, we can confidently differentiate between two systems that differ by 5% accuracy or higher. However, the current classification system would not be able to confidently differentiate between systems whose true accuracies differed by less than 5%. Thankfully, since we will only adjust performance when confronted with fairly sharp differences in system performance, this level of differentiation is likely to be sufficient.

A few caveats about this analysis are in order. First and foremost, this is only a simulation experiment. True performance is likely to vary more substantially across data sets, so in a true scenario the self-assessment system may only be able to differentiate between systems with an even larger difference in performance between them. Getting a better analysis would require a significant amount of naturally collected data from several users and several systems over a long time period, an experiment that is out of the scope of this work. Another point of note is that this analysis may seem straightforward and perhaps obvious and unnecessary to those with an understanding of classification systems. It is presented here both for the benefit of those who do not immediately see this insight and to get a further understanding of the specifics, such as between trial variance.

In this section we have established the plausibility of one potential self-assessment based personalization method. The next section will take a brief look at yet another way to personalize autocorrection systems.

6.2.2 Collecting Adaptation Training Data

The last method of personalization that we will discuss is using the data collected from the self-assessment procedure to improve the overall spell-checking/autocorrection model. The idea behind this method is that by performing the self-assessment procedure we are collecting a new corpus of problematic instances, which tell us information about the instances that the system handles incorrectly. We can use the self-assessment mechanism to collect the intended term for each correction mistake, and couple this with all of the other information we would need (keystrokes, dialogue context, etc.) to understand why the system reacted the way it did and what went wrong. Given this data we can either rebuild or modify the existing autocorrection system to better handle these particular problematic cases. This case is the word-level analogue of the character-level personalization mechanism discussed in Chapter 4.

Unfortunately, there is little that can be done with this type of personalization given the current self-assessment methods. To personalize in this manner, we need a self-assessment system that has better overall performance than the methods outlined in the previous chapter. As previously stated, precision is critical for this style of personalization, as we do not wish to add too much noise to the collected training corpus. By sacrificing recall, the current system can achieve high precision in certain cases. However, because training instances are less frequent at the word-level than the character-level, we can not sacrifice recall quite as liberally as we could for key-target resizing. For instance, given a dialogue with a problematic autocorrection attempt, an autocorrection self-assessment system can get, at most, one single training instance to add to our dataset. At the same time, a character-level system could grab a few hundred training instances over the course of that short dialogue. This means that even at perfect recall, it will take much longer for the word-level system to adapt than the character-level system. Because the adaptation is of little utility if we must wait years to employ it, autocorrection self-assessment systems must have both high precision

and reasonably high recall for this style of adaptation. Since the current system is unable to produce this, we cannot consider adapting in this manner. If in the future the self-assessment mechanisms outlined here are expanded upon, this type of personalization may become a reality. Until then, we must content ourselves with other personalization methods such as the two described earlier in this section.

6.3 Conclusion

The ultimate goals of the last two chapters were to introduce the notion of autocorrection self-assessment and motivate the need for it as an online data collection mechanism; to establish that these methods could be applied intelligently to personalize and adapt autocorrection systems; to give an analysis of the results of problematic and unproblematic autocorrection behavior. This chapter attempted to empirically establish that the methods it introduces are both plausibly implemented in state-of-the-art systems and practically useful.

The previous chapter described a novel problem of an autocorrection system assessing its own correction performance based on dialogue between two text messaging users. The identification of automatic correction mistakes is a potentially important task for personalization that has not been previously studied. This work explored two interrelated tasks necessary for a system to evaluate its own performance at automatically correcting words in text messages. The system must first evaluate each correction it has performed to assess whether the correction was erroneous. When erroneous corrections are detected, the system must then identify what term the user intended to type in order to understand the nature of its error. Because users use grounding utterances to explicitly correct errors after they have happened, cues from the dialogue following a system error can help inform these decisions.

The evaluation results indicated that given a problematic situation caused by an autocorrection system, the dialogue between users could provide important cues for the system to automatically assess its own correction performance. By exploring a rich set of features from the discourse, the approaches outlined in this work are able to both differentiate between

problematic and unproblematic instances and identify the term the user intended to type with high precision, achieving significantly above baseline performance.

Once it was established that self-assessment is practically possible, this chapter outlined how these self-assessment mechanisms could be used to personalize and dynamically adapt autocorrection systems. This chapter focused primarily on personalization through vocabulary expansion, in which self-assessment mechanisms were used to identify terms that a user intended to type, which were then added to the system's dictionary. It was shown that personalization in this manner had the potential to add vocabulary terms faster than a frequency-based baseline, while still maintaining high precision.

Two other personalization methods were proposed, but not fully evaluated due to lack of data. In one, self-assessment is used to to monitor the overall performance of the system and adjust the frequency with which autocorrection is performed. Although it could not be established that this method would be effective, it was established that the self-assessment performance given in the previous chapter should be accurate enough to enable this style of personalization to be attempted. Another proposed method of personalization was to use the data collected by self-assessment to retrain the autocorrection model, but it was concluded that such a method would likely only be practical with a system that could assess its performance with both very high precision and recall.

CHAPTER 7
Conclusion

The goal of this chapter is to briefly summarize the methods and contributions presented in previous chapters and to make one last attempt to explain what was done and why one should care. This chapter also outlines some limitations of the current methods and proposes avenues of future work.

7.1 Summary of Contributions

In brief, the main contributions of this work were:

- Examining the task of online data collection for key-target resizing algorithms, proposing several online data collection strategies and establishing that they could effectively be used to enable online adaptation.
- Establishing that the proposed online data collection strategies could be used to build continuously updated personalized key-target resizing touch models that outperformed non-adaptive general models.
- Proposing several hybrid approaches and establishing that they could be used to further improve upon online adaptation and personalization methods of key-target resizing.
- Examining the task of online data collection for autocorrection and autocompletion systems, proposing two self-assessment based data collection strategies, and establishing that they could effectively extract data for adaptation.

- Establishing that the proposed self-assessment strategies could be used to effectively personalize and adapt autocorrection and autocompletion systems.

The rest of this section gives a more in-depth overview at these and other more minor contributions.

The work presented here addresses the heretofore underexamined problem of online adaptation for mobile device text entry aids. To this end, it attempts to assess whether data collected without supervision is reliable enough to be trusted for use in building models and other forms of personalization and adaptation, a problem which has not been previously studied in this domain. The aim of this task is to collect data automatically and accurately in an online manner. This data can then be used to train or modify personalized models that display more nuance and produce better performance than the equivalent general model.

This work differentiates between text entry aids that work at the character level and those that work at the word level, and outlines the problems each wishes to solve. It is then argued that both of these text entry aid approaches can benefit from personalization methods. With this motivation, methods for online adaptation for both word and character level text entry aids are explored. To restrict our examination to relevant areas, we examine the assessment problem for a few text entry aids that are already in wide use.

In Chapters 3 and 4, the character-level text entry aid of key-target resizing was discussed. Key-target resizing attempts to help users of soft keyboards input each character by dynamically expanding the target area of keys they are likely to wish to select and decreasing the area of those they are not likely to select. This is done by calculating the probability of the user wishing to hit each key using a combination of two models: a model of the language and a model of touch behavior.

Because tailoring the language model is difficult and data intensive, and because previous studies by Himberg et al. (2003) and Rudchenko et al. (2011) have hinted at the power of personalized touch models, the work in Chapter 3 focuses on online data collection for building personalized touch models. Four models are examined: 1) a conservative model

that only includes keystrokes involved in error correction operations, 2) a word-level method that includes error correction keystrokes and data from words familiar to the system, 3) a discourse-level method that uses word sequence probabilities to attempt to remove real word spelling errors, and 4) a character-level method that trusts every character written by the user. Each of these methods examines different amounts of information from the discourse to determine which data should be considered in building a personalized touch model. These methods build on each other to examine and manage the tradeoff between the overall accuracy of the data collected and the time it takes to collect enough data to build a model.

Chapter 3 also addresses the logistical problem of collecting a dataset to examine the online adaptation of key-target resizing models. It is argued that the typical text copying methodology used to examine previous key-target resizing is insufficiently realistic for examining data collection in an online manner. To address this, a new data collection methodology, dubbed the semi-guided chat input methodology, is utilized. This methodology more closely simulates real use cases than the text copying methodology, while still allowing for high internal validity.

To examine the relative utility of each online data collection strategy in collecting data, this work examined the precision and recall of each method during data collection. As expected, each method handled the precision-recall tradeoff differently. The conservative method was able to produce high precision of over 0.99 (less than 1% noise added to the data), but it did so at the expense of recall, which only reached 0.123 (12.3% of all characters typed by the user were captured by the method). The word-level and character-level model were both able to produce recall of over 0.8, while keeping precision above 0.98. This suggested that the proposed methods were robust enough to handle the data collection aspect of online adaptation.

Chapter 4 looked at how the online adaptation data could be used for personalization. Data collected via each of the methods described in Chapter 3 was used to train personalized

models in order to understand the effect of each method on the final key-target resizing system. The conservative and discourse-level models proved to be ineffective (likely due to a lack of data), performing worse overall than general, non-personalized models trained on aggregate data. Thankfully, both the character-level and word-level models were able to outperform the general model. The word-level model performed best, with an overall keystroke error rate of 4.3%, a relative reduction in error rate of 10.4% over the general model. Notably, the performance of the word-level model was on par with a personalized model trained on gold standard data, suggesting that carefully chosen online data collection strategies are capable of retaining the gains of personalized models trained offline. Using data from previous studies, it was shown that this personalization process could be done relatively quickly; enough data to build a word-level personalized model can be obtained in about 5 days of average usage. This established that the methods proposed in this work effectively handled the adaptation and personalization problems for key-target resizing.

To identify any shortcomings of the proposed methods, an in-depth analysis of the models was performed. An analysis of keystroke error rate for each character revealed that, due to their limited amount of data, personalized models perform poorly on characters that appear infrequently. To overcome this deficiency, four hybrid models were proposed that combined data from both the personalized and general models. The four models were: 1) a seeding method that initializes the data for each key in the personalized model with a small number of data points from the general model, 2) a backoff method that trusts the personalized model's judgment on frequent keys and the general models judgment on infrequent keys, 3) a full backoff model that trusts the general model if an infrequent key could conceivably be the user's intended key and trusts the personalized model in all other cases, and 4) a linear combination model that takes a weighted average of the touch probabilities produced by the personalized and general models.

To understand the utility of each hybrid model, overall keystroke error rate was calculated. The backoff model performed poorly and was removed from further consideration. All

other hybrid models performed on par with the personalized model alone. An examination of the keystroke error rate at each key revealed that all three remaining models were able to outperform the personalized model on infrequent keys, but only the seeding method produced performance that was comparable to the general model. Another potential advantage of a hybrid model was that it has the potential to reduce the data collection time. An examination of how much data was needed by each model to show gains revealed that, while all hybrid methods could reduce the collection time, the linear combination method could cut the amount of data needed in half. Given that both the seeding and linear combination methods show potentially helpful contributions, it was concluded that which method was seen as superior would depend on a designer's understanding of the tradeoff between the need for swift collection and the need for good coverage of all keys.

Not all attempts to improve key-target resizing performance were successful. In particular, experiments with outlier removal and touch model bigrams were unable to improve overall performance. Nonetheless, the experiments in Chapter 4 established the potential for online data collection methods to help improve key-target resizing performance by personalizing and continuously adapting models to individual users. Overall, the methods in Chapters 3 and 4 were able to establish that adaptation and personalization of key-target resizing is both possible and desirable via the methods outlined in this thesis.

Chapter 5 examined the online data collection problem for two frequently deployed word-level text entry aids, automatic completion and automatic correction. Autocompletion systems attempt to speed up text entry by predicting what word a user intends to type before they have finished typing it, and autocorrection systems attempt to improve input accuracy by automatically correcting erroneous input. As before, autocorrection and autocompletion systems work by examining the user input and a model of the language to determine the best completion or correction.

Because the cost of an erroneous correction or completion attempt is higher than the gains associated with non-erroneous corrections and completions, it is critical that these

technologies have high accuracy. This fact, coupled with the fact that differences in user behavior are likely to lead to large variations in erroneous instances between different users, motivated the need for personalized models of autocompletion and autocorrection.

As it is difficult to obtain a large corpus of erroneous autocorrection behavior, a significant data collection problem needed to be addressed. Because of the differences in users and in autocorrection systems, building a large enough dataset to address the online adaptation problem was non-trivial. To build a corpus of problematic instances, screenshots of autocorrection mistakes made available on the internet by real mobile device users were transcribed and annotated. Additional steps were taken to remove potentially falsified instances, to ensure that this dataset was as representative as possible of real usage behavior. To build a comparable dataset of unproblematic instances, snippets of uncorrected dialogue were extracted from the data collected for the key-target resizing experiments in Chapter 3.

An autocorrection system must be able to assess its own performance if it wants to be able to dynamically adapt to user behavior, and as such the online data collection task was focused on system self-assessment. Chapter 5 divided this self-assessment task into two parts: 1) differentiating between problematic and unproblematic dialogues and 2) identifying the intended term of a problematic instance. To address these problems, novel methods were proposed that drew inspiration from previous self-assessment work.

Differentiating between problematic and unproblematic instances was modeled as a binary classification problem. As many of the data sources used in spoken dialogue system self-assessment were unavailable, the features used to differentiate between classes focused primarily on detecting certain dialogue behaviors that were indicative of problematic instances. In particular, the features used could be broadly divided into features that attempted to detect confusion, features that attempted to detect correction attempts, and features that attempted to detect when the overall flow of the dialogue suggested that an autocorrection mistake had occurred.

A support vector machine classifier was used for classification. Results suggested that by utilizing the entire feature set the classifier could detect problematic instances with a precision of 0.86 and a recall of 0.75, achieving performance above a majority class baseline. The classifier achieved accuracy of 0.79, suggesting that it could make the correct self-assessment judgment in about 4 out of every 5 cases. To see the impact of each feature source, a feature ablation study was conducted. It was shown that removing any of the feature sources had little effect on the overall precision of the classifier, but had a negative impact on the recall. Removing the dialogue flow features had the largest effect on the recall, an absolute drop of about 0.2.

Identifying the intended term for a problematic instance was modeled as a ranking problem. The feature set for intended term identification focused primarily on the relationship between an erroneous correction attempt and a candidate intended term. There were five feature sources: 1) contextual features captured discourse-level phenomena, 2) punctuation features captured information given by the usage of certain punctuation marks, 3) word form features captured variations in how a candidate word was written, 4) similarity features captured the character-level similarity between an erroneous correction and a candidate word, and 5) pattern features captured certain phrases that highlighted a candidate term.

To identify the intended terms, each candidate word in the dialogue was ranked and the highest ranked term was deemed to be the intended term. The precision of the method at different recall levels was examined by thresholding at different ranking values. Results suggested that intended term identification could be performed at over 0.9 precision with recall of about 0.4, and could maintain high precision while recall increased. Feature ablation was again conducted to assess the contribution of each feature source. It was shown that removing similarity, contextual, or punctuation features caused a notable decrease in performance, while pattern and word form features had relatively little effect overall.

In order to get an assessment of the overall performance in a more realistic setting, two additional experiments were conducted. In one, the output of the problematic vs. unprob-

lematic classification was given as input to the intended term predictor. In the other, the performance of this end-to-end system was estimated on an unbalanced dataset that contained 90% unproblematic instances. In each of these cases the performance was weaker than on the intended term classification alone, but they were still able to achieve high precision at lower recall levels. Nonetheless, the work in Chapter 5 established the potential of a few different methods that could be used to enable online adaptation.

After Chapter 5 established that self-assessment of autocorrection systems was feasible, Chapter 6 examined several personalization methods that were made possible by these methods. In one, it was established by simulation that the current level of problematic vs. unproblematic classification was likely to be sufficient to get a rough estimate of overall autocorrection performance for a given user. Using this, the system can adopt a more conservative or more aggressive autocorrection policy as appropriate based on its current performance.

As autocorrection mistakes are often caused by the autocorrection system not having the intended term in its dictionary, vocabulary acquisition was another avenue of autocorrection personalization. Self-assessment based vocabulary acquisition reasons that if we identify an intended term that is out of the dictionary, we can safely add it to our dictionary as we have identified the fact that it is what the user intended to write. This method of vocabulary acquisition is able to capture a larger percentage of out-of-vocabulary terms than standard methods based on word frequency, while still maintaining high precision. Because an autocorrection system has no chance of correctly identifying intended terms that are not in its dictionary, the coverage of its dictionary serves as a limiting factor to possible autocorrection performance. Using this, it was established that by using the self-assessment based vocabulary acquisition method an autocorrection system would be able to achieve an absolute increase in upper-bound performance of over 17% over a system that did not perform any vocabulary expansion.

As was the case with personalization of character-level aids, not every attempt at autocorrection personalization proved to be successful. Notably, utilizing the self-assessment data as training data to improve the overall model performance was deemed to be impossible with the performance of the current assessment mechanisms. Despite this, the conclusion to be drawn from the work in Chapter 6 follows that drawn in Chapter 4; self-assessment is a valuable tool that is able to enable adaptation and personalization mechanisms that help text entry aids improve their overall performance.

7.2 Limitations and Future Work

This final section gives a brief outline of work that still must be done in relation to mobile text entry self assessment, adaptation, and personalization. Overall, the methods outlined in this thesis address problems that have not been adequately addressed previously, and as such there are plenty of further experiments that could be conducted to move the field forward. Although this thesis attempts to give an overview of the task and some attempts at implementation in a few key areas, it covers only a small subset of the overall field of mobile device text entry, and only discusses a small number of techniques relevent to personalization and user modeling.

We can divide the future work into two general categories: future work on online data collection and adaptation of the text entry aids discussed in the previous chapters and future work applying these methods to other areas. For the text entry aids that have been discussed we can give concrete examples of open questions and needed improvements. The latter category is overly broad, so only a general sketch of the potential for improvement in these areas will be provided.

As one might imagine, understanding what work has yet to be done is intrinsically related to understanding the limitations of the current approaches. As such, this chapter will also highlight some of the limitations of the methods proposed, and some of the limitations of online data collection and personalization techniques in general. Some of these limitations

have been touched on previously and some have not.

7.2.1 Key-Target Resizing

In general, the techniques outlined for online data collection, adaptation, and personalization of character-level text entry aids are more mature and closer to widespread implementation than those outlined for word-level aids. One of the primary reasons for this is the quantity of data and the collection time; with the potential to capture a data point for training with every keystroke, character-level aids can quickly gain enough data to adapt models, overcoming one of the main hurdles to personalization (Webb et al., 2001). In addition, adapting touch models addresses a difference between users that manifests itself both cognitively (in differences between mental models) and physically (in differences between hand sizes), potentially leading to stark divides in behavior. Finally, the frequency of text input means that even small gains from personalization are meaningful.

This is not to say that there are not reasons to be cautious. One point for consideration is that it is possible to overadapt. In order to personalize, the system must be able to associate a given behavior with the profile of a single individual. In many user modeling applications the system handles the profiles of many different users and employs some sort of sign-in process to differentiate between them. In the work presented here we only store a single profile, associated with the primary user of the device. While we can safely assume that mobile devices are most frequently associated with a single individual, this is likely not universally true. Work by Liu et al. (2009) indicates that over 80% of mobile device users share their devices with others on occasion. If the personalization process has adapted to the primary user, the borrowing user may find it difficult to type on the device. If the device is only lent occasionally, then this should be seen as a minor drawback. However, if the device is habitually shared between several users, the one-user assumption used in personalization breaks down. Handling this problem means addressing the severity and speed of adaptation, two points that have not been addressed in this thesis.

Similarly, the personalization process can run into problems if there is too much intra-user variation, particularly in regards to typing style. In the experiments conducted in Chapter 3, users were asked to adopt a consistent typing style throughout the data collection process. While this is a reasonable control that follows previous work, it may have an adverse effect on the external validity of the results. Although a user's mental model would likely stay constant between typing styles, the physical characteristics of their typing behavior would change (e.g., differences in angle between one and two thumb input, differences in finger size between thumb and other finger input). If the user changes typing style frequently, the personalization process would attempt to assign a single typing model when several different models would be most appropriate. One possible solution to this would be to attempt to dynamically identify the typing style of the user at input time and learn and apply the appropriate model. This would likely require some understanding of the characteristics of each typing style. It should be noted that this problem is not limited to key-target resizing personalization; even with aggregate data, different typing styles may be best captured by distinct models. This suggests that the problem of dynamically identifying and applying typing style may be a very promising avenue of future work.

Now that we have discussed some of the more general limitations of key-target resizing personalization, we can examine the limitations of the approaches advanced here. For instance, in regards to the online data collection strategies outlined in Chapter 3, one potential criticism is that they are, to be blunt, rather simplistic approaches. This is of course by design, as there is a time and computation premium placed on the data collection process, as discussed previously. Another point in the favor of the current approaches is that they work rather well, producing the high precision needed to train accurate models while still managing to keep recall high enough to collect data swiftly. Nonetheless, one might reason that the simplistic approaches outlined here would be outclassed by clever implementations of more sophisticated approaches that utilized more complex algorithms or took into account more rich feature sources, which may indeed be the case.

In particular, there are a few feature sources that may prove useful in future online data collection methods and personalization attempts. Information about the size of the blob registered by each keypress may help to differentiate between typing styles or identify keypresses that are not registered correctly by the system. Examining the user's typing speed may also help to inform judgments in both of these areas. Finally, the application that the user is typing into or the topic of conversation may have a noticeable effect on their typing behavior, and as such capturing this information could be useful in advancing personalization.

7.2.2 Autocorrection

Unfortunately, despite the work outlined here, there is still much work to do before true personalization of autocorrection and autocompletion mechanisms is practically viable. One of the most fundamental problems is that the personalization mechanisms are likely to be too slow. Using the currently proposed mechanisms, if the autocorrection system is working poorly for a given user, the system must see several failed autocorrection attempts before any adaptation takes place. However, many users will simply turn the autocorrection system off when faced with poor autocorrection performance. This is problematic because those users that give up on autocorrection are likely to be the users that would benefit the most from the personalization process.

As with key-target resizing, the autocorrection personalization process runs the risk of overadapting. One example would be a system that detects that it is performing poorly and reduces the frequency of autocorrection more than in the ideal case. This system runs the risk of cutting its autocorrection policy too severely, to the point where almost no autocorrection takes place. Because autocorrection is undergone infrequently (relative to character-level aids), this system may take a considerable amount of time to realize that it is being too conservative, and during this time the user will not be getting the full benefits of autocorrection.

As another example, assume the system learns a new word via a self-assessment based vocabulary acquisition process. If this word was actually a spelling mistake, this will effect the system performance until this mistake is purged from the autocorrection dictionary. Although simple methods for removing these terms from the dictionary exist, there is certainly room for further work in this area. Furthermore, even if the system made the correct judgment in adding it to the dictionary, the word may be topic or application specific, a distinction that current methods of dictionary expansion would not pick up on. This could lead to the autocorrection system suggesting the word in cases in which it is not appropriate, unless a more constrained dictionary expansion method is developed.

There are still many improvements that could be made to the self-assessment approaches outlined in this thesis. Most notably, there is a need to utilize more feature sources. The difficultly in collecting a sufficiently large and realistic corpus of autocorrection mistakes led to the use of transcribed screenshots, which removed several feature sources that could have proved useful. Information about the character stream is likely to be the most influential feature source that was unavailable. One possible use of character-level information would be in building a more robust similarity metric. If the self-assessment mechanism had access to the sequence of touch points the user typed for the autocorrected word it could evaluate the similarity between a candidate intended term and the touch sequence, a more relevant measure than the similarity between the candidate term and the autocorrected word. Another possible use of character-level information is in understanding the correction behavior of the user. For instance, a user that frequently corrects typing errors is more likely to be monitoring the text closely than a user that corrects errors infrequently, and thus the former user is less likely to let an erroneous autocorrection slip by.

Similarly, having access to the user's low level actions would enhance the system's ability to self-assess, as it would be able to identify cases in which it acted incorrectly even if the user caught and corrected the mistake before committing their message. For instance, if the system performs autocorrection and the user changes the corrected word, the system can be

fairly confident that it acted incorrectly and that the term the user chose instead was their intended term. Using this information the system could still collect self-assessment data even for a careful user that never allowed an autocorrection mistake to pass by uncorrected.

Related to the above points is the fact that one of the hardest and most critical problems that future work must address is collecting a dataset that allows access to these low-level user actions while still providing realistic examples of autocorrection mistakes. As addressed earlier, it is difficult to collect such a dataset in the lab without doing severe harm to the external validity of the experiment. Likewise, collection in a more open setting would likely take a considerable amount of time to collect a sufficiently large dataset and raise troubling privacy issues. Collecting a more realistic dataset will also address points that were unknown during the current evaluation, such as the correct data balance between problematic and unproblematic instances. Solving the data collection problem is likely to be fundamental to advancing autocorrection self-assessment and personalization.

Beyond expanding feature sources, autocorrection self-assessment may also benefit from improved methods. The problematic and unproblematic classification task and intended term identification task were presented separately here, but they are of course interrelated. For instance, correction dialogue that points out an intended term also indicates that a mistake has taken place. While the current feature set did attempt to capture some of this overlapping information, further methods may chose to combine these two tasks into a single task in order better handle their inherent coupling.

7.2.3 Adaptation and Personalization for Other Aids

One primary question we may wish to ask about the methods outlined here is, "will these methods be of any use in other areas?". The methods presented here are tailored to the specific problem that they are designed to address, and as such one might believe that they are of limited utility in other areas. While it is unlikely that the methods outlined here could be applied as is to other text entry aids, they do give an outline as to what online

data collection and personalization might entail in these areas. The methods outlined here address a fundamental user modeling problem of attempting to collect and utilize client-side data under time, complexity, and privacy limitations, and personalization of most other mobile device text entry aids will share these limitations.

This is not to say that these methods would need great modifications to be useful for other text entry aids. Many of these aids are based on or make some utilization of touch models and word-level and character-level language models. For instance, the relevant information for key highlighting (Paek et al., 2010) is similar to that of key-target resizing. In the former case the system wishes to identify the keys that the user is likely to hit next given the user's previous keypresses, while in the latter case the system uses the user's previous keypresses to identify the current key. Given the similarity, it is likely that the collection and adaptation techniques outlined here could be applied to key highlighting with little to no modification.

In the word-level case, we have already seen an instance in which there is parity between two aids, autocorrection and autocompletion. Moreover, the self-assessment techniques proposed address the case in which the system is a dialogue helper but not a participant, a situation that appears frequently for text entry aids but had not been previously addressed in the literature. Additionally, the classification and ranking approaches used are reasonably straightforward and likely to be applicable to many similar situations.

7.3 Final Thoughts

Above all, the point that this thesis wished to stress is that dynamically obtaining the data to build a model to represent human behavior is a non-trivial task. Just as we believe it necessary to consider different word choices in different contexts, we should also believe that it is necessary to take into account individual user differences. We can see a significant amount of evidence for this fact by examining the ways in which user differences effect the relatively narrow task of building text input aids, from the shape and size of their fingers to the differences is internal mental models. We must realize that lumping the plethora of

human differences together into a single user model is a crass and ultimately futile way to build a model that supposes to capture the diversity of human behavior. By learning to assess user data as it is given to us, we can built a system that is wiser and more accurate.

The work presented here attempts to advance the state-of-the-art by allowing the injection of more intelligence into mobile device text entry systems. It has done so by proposing and evaluating several methods that allow for online adaptation, and evaluating several personalization methods that can be built on top of them. It is hoped that by examining the online adaptation problem we can not only personalize and improve existing methods, but help to develop further methods that are able to take advantage of the expanded awareness to individual user differences that these methods provide to us.

APPENDIX

Pair	User	Gender	Age	Dominant Hand	Previous Soft Keyboard Experience	Experience with device
1	1	F	25	Right	Yes	No
	2	F	25	Right	Yes	No
2	3	F	21	Right	Yes	No
	4	M	22	Left	No	No
3	5	M	22	Right	No	No
	6	M	22	Right	No	No
4	7	M	21	Right	No	No
	8	M	21	Right	No	No

Table A.1: Study independent statistics about each user in the key-target resizing dataset

Pair Pair	User User	Device Device	Input Style Input Style	Avg. Message Length	Total Keystrokes	Text Keystrokes	Backspace Keystrokes
1	1	EVO	2 Thumb	20.9 ± 15.8	6088	4826	843
	2	Nexus	1 Thumb	15.3 ± 10.9	5805	4954	440
2	3	EVO	2 Thumb	34.9 ± 26.2	6912	5540	1112
	4	Nexus	2 Thumb	32.6 ± 23.9	4597	3987	404
3	5	EVO	2 Thumb	35.3 ± 30.0	5267	4598	453
	6	Nexus	2 Thumb	33.5 ± 26.5	3061	2561	382
4	7	EVO	2 Thumb	26.6 ± 24.6	4591	3492	543
	8	Nexus	2 Thumb	18.8 ± 19.5	4455	3053	962

Table A.2: Study dependent statistics about each user in the key-target resizing dataset

REFERENCES

REFERENCES

Ahmad, F. and Kondrak, G. (2005). Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 955–962, Stroudsburg, PA, USA. Association for Computational Linguistics.

Al Faraj, K., Mojahid, M., and Vigouroux, N. (2009). Bigkey: A virtual keyboard for mobile devices. In *Proceedings of the 13th International Conference on Human-Computer Interaction. Part III: Ubiquitous and Intelligent Interaction*, pages 3–10, Berlin, Heidelberg. Springer-Verlag.

Allen, J. and Perrault, C. R. (1986). Readings in natural language processing. chapter Analyzing intention in utterances, pages 441–458. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Ardissono, L. and Sestero, D. (1996). Using dynamic user models in the recognition of the plans of the user. *User Modeling and User Adapted Interaction*, pages 157–190.

Balakrishnan, R. and MacKenzie, I. S. (1997). Performance differences in the fingers, wrist, and forearm in computer input control. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '97, pages 303–310, New York, NY, USA. ACM.

Baldwin, T. and Chai, J. (2011). Beyond normalization: Pragmatics of word form in text messages. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1437–1441, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Baldwin, T. and Chai, J. (2012a). Autonomous self-assessment of autocorrections: Exploring text message dialogues. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 710–719, Montréal, Canada. Association for Computational Linguistics.

Baldwin, T. and Chai, J. (2012b). Towards online adaptation and personalization of key-target resizing for mobile devices. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, IUI '12, pages 11–20, New York, NY, USA. ACM.

Bast, H., Mortensen, C. W., and Weber, I. (2008). Output-sensitive autocompletion search. *Inf. Retr.*, 11:269–286.

Battestini, A., Setlur, V., and Sohn, T. (2010). A large scale study of text-messaging use. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 229–238, New York, NY, USA. ACM.

Benko, H., Wilson, A. D., and Baudisch, P. (2006). Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1263–1272, New York, NY, USA. ACM.

Brajnik, G., Guida, G., and Tasso, C. (1987). User modeling in intelligent information retrieval. *Inf. Process. Manage.*, 23(4):305–320.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, Morristown, NJ, USA. Association for Computational Linguistics.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19:263–311.

Carberry, S. (2000). Plan Recognition: Achievements, Problems, and Prospects. *User Modeling and User-adapted Interaction.*

Card, S. K., English, W. K., and Burr, B. J. (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys, for text selection on a crt. *Ergonomics*, 21:601–613.

Carenini, G., V. M. and Moore, J. D. (1994). Generating patient specific interactive natural language explanations. In *Proceedings of the Eighteenth Symposium on Computer Applications in Medical Care*, Banff, Canada.

Carlberger, A., Carlberger, J., Magnuson, T., Hunnicutt, M. S., Palazuelos-cagigas, S. E., and Navarro, S. A. (1997). Profet, a new generation of word prediction: An evaluation study. In *Proceedings of the 2nd Workshop on NLP for Communication Aids.*

Chai, J. Y., Baldwin, T., and Zhang, C. (2006a). Automated performance assessment in interactive qa. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 631–632, New York, NY, USA. ACM.

Chai, J. Y., Zhang, C., and Baldwin, T. (2006b). Towards conversational qa: automatic identification of problematic situations and user intent. In *Proceedings of the COLING/ACL on Main conference poster sessions*, COLING-ACL '06, pages 57–64, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Technical report TR-10-98, Harvard University.

Choudhury, M., Saraf, R., Jain, V., Mukherjee, A., Sarkar, S., and Basu, A. (2007). Investigation and modeling of the structure of texting language. *Int. J. Doc. Anal. Recognit.*, 10(3):157–174.

Clark, H. H. (1996). *Using Language.* Cambridge University Press.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46.

Cohen, R., Song, F., Spencer, B., and van Beek, P. (1991). Exploiting temporal and novel information from the user in plan recognition. *User Modeling and User-adapted Interaction*, 1(3-4):125–148.

Cook, P. and Stevenson, S. (2009). An unsupervised model for text message normalization. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 71–78, Boulder, Colorado. Association for Computational Linguistics.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7:171–176.

Dvorak, A. and Dealey, W. L. (1936). Typewriter keyboard. United States Patent 2040248.

Fazly, A. and Hirst, G. (2003). Testing the efficacy of part-of-speech information in word completion. In *Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods*, TextEntry '03, pages 9–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391.

Frey, L., White, K.P., J., and Hutchison, T. (1990). Eye-gaze word processing. *Systems, Man and Cybernetics, IEEE Transactions on*, 20(4):944 –950.

Garay-Vitoria, N. and Abascal, J. (1997). Intelligent word-prediction to enhance text input rate (a syntactic analysis-based word-prediction aid for people with severe motor and speech disability). In *IUI'97*, pages 241–244.

Gauch, S., Speretta, M., Chandramouli, A., and Micarelli, A. (2007). The adaptive web. chapter User profiles for personalized information access, pages 54–89. Springer-Verlag, Berlin, Heidelberg.

Ghorab, M., Zhou, D., OConnor, A., and Wade, V. (2012). Personalised information retrieval: survey and classification. *User Modeling and User-Adapted Interaction*, pages 1–63. 10.1007/s11257-012-9124-1.

Godfrey, J., Holliman, E., and McDaniel, J. (1992). Switchboard: telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 517 –520 vol.1.

Goodman, J. (2001). A bit of progress in language modeling. *Computer Speech and Language*, 15:403–434.

Goodman, J., Venolia, G., Steury, K., and Parker, C. (2002). Language modeling for soft keyboards. In *Eighteenth national conference on Artificial intelligence*, pages 419–424, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Gorin, A. L., Riccardi, G., and Wright, J. H. (1997). How may i help you? *Speech Commun.*, 23(1-2):113–127.

Greenshpan, O., Milo, T., and Polyzotis, N. (2009). Autocompletion for mashups. *Proc. VLDB Endow.*, 2:538–549.

Grover, D., King, M., and Kushler, C. (1998). Reduced keyboard disambiguating computer. US Patent no. 5818437.

Gunawardana, A., Paek, T., and Meek, C. (2010). Usability guided key-target resizing for soft keyboards. In *Proceedings of the 15th international conference on Intelligent user interfaces*, IUI '10, pages 111–118, New York, NY, USA. ACM.

Hadi, A. S. (1992). Identifying multiple outliers in multivariate data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 54(3):pp. 761–771.

Hara, S., Kitaoka, N., and Takeda, K. (2010). Estimation method of user satisfaction using n-gram-based dialog history model for spoken dialog system. In Chair), N. C. C., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Himberg, J., Häkkilä, J., Kangas, P., and Mäntyjärvi, J. (2003). On-line personalization of a touch screen based keyboard. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 77–84, New York, NY, USA. ACM.

Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Nat. Lang. Eng.*, 11:87–111.

Hoggan, E., Brewster, S. A., and Johnston, J. (2008). Investigating the effectiveness of tactile feedback for mobile touchscreens. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1573–1582, New York, NY, USA. ACM.

Holz, C. and Baudisch, P. (2010). The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 581–590, New York, NY, USA. ACM.

Holz, C. and Baudisch, P. (2011). Understanding touch. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2501–2510, New York, NY, USA. ACM.

Hu, J., Brown, M. K., and Turin, W. (1996). Hmm based on-line handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18:1039–1045.

Informa (2011). Global SMS traffic to reach 8.7 trillion in 2015. "`http://www.informatm.com/itmgcontent/icoms/whats-new/20017843617.html`". [Online; accessed April 27, 2011].

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556.

Kernighan, M. D., Church, K. W., and Gale, W. A. (1990). A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics*, pages 205–210, Morristown, NJ, USA. Association for Computational Linguistics.

Khoussainova, N., Kwon, Y., Balazinska, M., and Suciu, D. (2010). Snipsuggest: context-aware autocompletion for sql. *Proc. VLDB Endow.*, 4:22–33.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181 –184 vol.1.

Kobus, C., Yvon, F., and Damnati, G. (2008). Normalizing SMS: are two metaphors better than one ? In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 441–448, Manchester, UK. Coling 2008 Organizing Committee.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Konstan, J. and Riedl, J. (2012). Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 22:101–123. 10.1007/s11257-011-9112-x.

Kristensson, P.-O. and Zhai, S. (2004). Shark2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, pages 43–52, New York, NY, USA. ACM.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24:377–439.

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1):pp. 159–174.

Lee, S. and Zhai, S. (2009). The performance of touch screen soft buttons. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 309–318, New York, NY, USA. ACM.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Li, J. and Hirst, G. (2005). Semantic knowledge in word completion. In *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, Assets '05, pages 121–128, New York, NY, USA. ACM.

Ling, R. and Baron, N. S. (2007). Text messaging and im: Linguistic comparison of american college data. *Journal of Language and Social Psychology*, 26(3):291–298.

Litman, D., Hirschberg, J., and Swerts, M. (2006). Characterizing and predicting corrections in spoken dialogue systems. *Comput. Linguist.*, 32:417–438.

Litman, D. J. and Pan, S. (2002). Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, 12:111–137. 10.1023/A:1015036910358.

Liu, Y., Rahmati, A., Huang, Y., Jang, H., Zhong, L., Zhang, Y., and Zhang, S. (2009). xshare: supporting impromptu sharing of mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 15–28, New York, NY, USA. ACM.

MacKenzie, I. S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Hum.-Comput. Interact.*, 7:91–139.

MacKenzie, I. S. (2002). Kspc (keystrokes per character) as a characteristic of text entry techniques. In *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, Mobile HCI '02, pages 195–210, London, UK. Springer-Verlag.

Mackenzie, I. S., Seller, A., and Buxton, W. (1991). A comparison of input devices in elemental pointing and dragging tasks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 91*, pages 161–166. ACM.

MacKenzie, I. S. and Soukoreff, R. W. (2002). A character-level error analysis technique for evaluating text entry methods. In *Proceedings of the second Nordic conference on Human-computer interaction*, NordiCHI '02, pages 243–246, New York, NY, USA. ACM.

MacKenzie, I. S. and Soukoreff, R. W. (2003). Phrase sets for evaluating text entry techniques. In *CHI '03 extended abstracts on Human factors in computing systems*, CHI EA '03, pages 754–755, New York, NY, USA. ACM.

MacKenzie, I. S. and Tanaka-Ishii, K. (2007). *Text Entry Systems: Mobility, Accessibility, Universality (Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

MacKenzie, I. S. and Zhang, S. X. (1999). The design and evaluation of a high-performance soft keyboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 25–31, New York, NY, USA. ACM.

Mahalanobis, P. C. (1936). On the generalised distance in statistics. In *National Institute of Sciences of India*, volume 2, pages 49–55.

Matias, E., MacKenzie, I. S., and Buxton, W. (1996). One-handed touch typing on a qwerty keyboard. *Human-Computer Interaction*, 11:1–27.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Inf. Process. Manage.*, 27:517–522.

Mayzner, M. S. and Tresselt, M. E. (1965). Tables of single-letter and digram frequency counts for various word-length and letter-position combinations. *Psychonomic Monograph Supplements*, 1:13–32.

McCallum, A., Bellare, K., and Pereira, F. (2005). A conditional random field for discriminatively-trained finite-state string edit distance. In *Conference on Uncertainty in AI (UAI)*.

Michaud, L. N. and McCoy, K. F. (1999). Modeling user language proficiency in a writing tutor for deaf learners of english. In *Proceedings of a Symposium on Computer Mediated Language Assessment and Evaluation in Natural Language Processing*, ASSESSEVALNLP '99, pages 47–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mulligan, D. and Schwartz, A. (2000). Your place or mine?: privacy concerns and solutions for server and client-side storage of personal information. In *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*, CFP '00, pages 81–84, New York, NY, USA. ACM.

Paek, T., Chang, K., Almog, I., Badger, E., and Sengupta, T. (2010). A practical examination of multimodal feedback and guidance signals for mobile touchscreen keyboards. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 365–368, New York, NY, USA. ACM.

Paris, C. L. (1989). The use of explicit user models in a generation system for tailoring answers to the user's level of expertise. In Kobsa, A. and Wahlster, W., editors, *User Models in Dialog Systems*, pages 200–232. Springer, Berlin, Heidelberg.

Prasad, R. and Walker, M. (2002). Training a dialogue act tagger for human-human and human-computer travel dialogues. In *Proceedings of the 3rd SIGdial workshop on Discourse and dialogue - Volume 2*, SIGDIAL '02, pages 162–173, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rabin, E. and Gordon, A. (2004). Tactile feedback contributes to consistency of finger movements during typing. *Experimental Brain Research*, 155:362–369. 10.1007/s00221-003-1736-6.

Renaud, A., Shein, F., and Tsang, V. (2010). Grammaticality judgement in a word completion task. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids*, CL&W '10, pages 15–23, Stroudsburg, PA, USA. Association for Computational Linguistics.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 175–186, New York, NY, USA. ACM.

Ritter, A., Cherry, C., and Dolan, B. (2010). Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American*

Chapter of the Association for Computational Linguistics, pages 172–180, Los Angeles, California. Association for Computational Linguistics.

Rudchenko, D., Paek, T., and Badger, E. (2011). Text text revolution: A game that improves text entry on mobile touchscreen keyboards. In *Proceedings of PERVASIVE*.

Sadun, E. (2009). What the duck? Train your iPhone to (truly) learn new words. "`http://arstechnica.com/apple/news/2009/01/what-the-duck-train-your-iphone-to-truly-learn-new-words.ars`". [Online; accessed April 20, 2012].

Schomaker, L. (1998). From handwriting analysis to pen-computer applications. *Electronics Communication Engineering Journal*, 10(3):93 –102.

Sholes, L. C., Glidden, C., and Soule, S. W. (1868). Improvement in type-writing machines. United States Patent 79868.

Soukoreff, R. W. and MacKenzie, I. S. (2001). Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. In *CHI '01 extended abstracts on Human factors in computing systems*, CHI EA '01, pages 319–320, New York, NY, USA. ACM.

Soukoreff, R. W. and MacKenzie, I. S. (2003). Metrics for text entry research: an evaluation of msd and kspc, and a new unified error metric. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 113–120, New York, NY, USA. ACM.

Sproat, R., Black, A. W., Chen, S. F., Kumar, S., Ostendorf, M., and Richards, C. (2001). Normalization of non-standard words. *Computer Speech and Language*, pages 287–333.

Swerts, M., Litman, D., and Hirschberg, J. (2000). Corrections in spoken dialogue systems. In *In Proceedings of the Sixth International Conference on Spoken Language Processing*, pages 615–618.

Tappert, C., Suen, C., and Wakahara, T. (1990). The state of the art in online handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(8):787 –808.

TextwareSolutions (1998). The fitaly one-finger keyboard.

The Associated Press (2011). Number of Cell Phones Worldwide Hits 4.6B. "`http://www.cbsnews.com/stories/2010/02/15/business/main6209772.shtml`". [Online; accessed April 26, 2011].

Toch, E., Wang, Y., and Cranor, L. (2012). Personalization and privacy: a survey of privacy risks and remedies in personalization-based systems. *User Modeling and User-Adapted Interaction*, 22:203–220. 10.1007/s11257-011-9110-z.

Toutanova, K. and Moore, R. (2002). Pronunciation modeling for improved spelling correction. In *40th Annual Meeting of the Association for Computational Linguistics(ACL 2002)*.

Trnka, K., Yarrington, D., McCoy, K., and Pennington, C. (2006). Topic modeling in fringe word prediction for aac. In *Proceedings of the 11th international conference on Intelligent user interfaces*, IUI '06, pages 276–278, New York, NY, USA. ACM.

Vogel, D. and Baudisch, P. (2007). Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 657–666, New York, NY, USA. ACM.

Walker, M., Langkilde, I., Wright, J., Gorin, A., and Litman, D. (2000). Problematic situations in a spoken dialogue system: Experiments with how may i help you? In *Proceedings of the North American Meeting of the Association for Computational Linguistics*, pages 210–217.

Webb, G. I., Pazzani, M. J., and Billsus, D. (2001). Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):19–29.

Whitelaw, C., Hutchinson, B., Chung, G. Y., and Ellis, G. (2009). Using the Web for language independent spellchecking and autocorrection. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 890–899, Singapore. Association for Computational Linguistics.

Wilcox-O'Hearn, A., Hirst, G., and Budanitsky, A. (2008). Real-word spelling correction with trigrams: a reconsideration of the mays, damerau, and mercer model. In *Proceedings of the 9th international conference on Computational linguistics and intelligent text processing*, CICLing'08, pages 605–616, Berlin, Heidelberg. Springer-Verlag.

Wobbrock, J. O. (2007). *Measures of Text Entry Performance*, chapter 3. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Wobbrock, J. O. and Myers, B. A. (2006). Analyzing the input stream for character- level errors in unconstrained text entry evaluations. *ACM Trans. Comput.-Hum. Interact.*, 13:458–489.

Wu, D. (1991). Active acquisition of user models: Implications for decision-theoretic dialog planning and plan recognition. *User Modeling and User-adapted Interaction*, 1(3-4):149–172.

Yamada, H. (1980). A historical study of typewriters and typing methods: from the position of planing japanese parallels. *Journal of Information Processing*, 2:175–202.

Zhai, S., Hunter, M., and Smith, B. A. (2000). The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, pages 119–128, New York, NY, USA. ACM.

Zhai, S. and Kristensson, P.-O. (2003). Shorthand writing on stylus keyboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 97–104, New York, NY, USA. ACM.

Zukerman, I. and Litman, D. (2001). Natural language processing and user modeling: Synergies and limitations. *User Modeling and User-Adapted Interaction*, 11(1-2):129–158.

Zukerman, I. and Mcconachy, R. (1993). Consulting a user model to address a user's inferences during content planning. *User Modeling and User-Adapted Interaction*, 3(2):155–185.