Text entry for mobile devices using ad-hoc abbreviation

Stefano Pini
spini@mit.edu

Sangmok Han
sangmok@mit.edu

David R. Wallace
drwallac@mit.edu

Department of Mechanical Engineering
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA, 02139, USA

ABSTRACT
This paper presents a new method for improving the number of keystrokes and time required for text entry on mobile devices using ad-hoc abbreviations. The approach is easy-to-use because: users are not required to learn any pre-defined abbreviation rules; abbreviated input phrases are automatically detected and expanded; and it is possible to recover words that may be omitted from phrases either by accident or intention. The paper develops algorithms to detect abbreviated phrases using a Support Vector Machine trained on abbreviation examples and to expand abbreviations into complete phrases using a Hidden Markov Model learned from a text corpus. The abbreviation detector was evaluated on 3,000 word-abbreviation pairs and achieved 90% accuracy. The abbreviation expander was evaluated on 100,000 phrases and achieved 95% accuracy. A user study with 10 participants was performed to measure time and keystroke savings of the new approach compared to the existing iPhone text entry system. Keystroke savings were consistent amongst users, with an average decrease of 32%. Time for input varied considerably depending on familiarity with the approach, increasing for novice users. However, experienced users achieved an average time saving of 26%. Observations suggest that novice users were spending time thinking about how they wanted to abbreviate words.

Categories and Subject Descriptors
H.5.2 [User Interfaces]: Natural Language.

General Terms
Algorithms, Design.

Keywords

∗Permanent address:
Dipartimento di Ingegneria Industriale, Università degli Studi di Parma, Viale G. P. Usberti, 181/A, I-43124, Parma, Italy.
stefano.pini@nemo.unipr.it.

1. INTRODUCTION
Applications in modern mobile devices, such email clients, messaging tools and productivity software, require substantial text input and highlight the need for more efficient input systems. A corpus study of email messages suggests that many phrases are reused—the top 3% of the most frequently used 3-word phrases were found to account for 32% of all 3-word phrase occurrences—as will be shown in Section 6.2.2. This implies that systems which facilitate the reuse of previously written phrases may improve the efficiency of writing new messages.

We propose a multiple-word auto-completion approach that can infer full expressions from abbreviated input using statistical models learned from a user’s previously written phrases. For example, the approach can expand the following abbreviated input into full expressions:

\[ \text{a i wd lk t and I would like to} \]
\[ \Rightarrow \text{Let me know what you think} \]
\[ \text{lt m n wh u th} \equiv \text{Let’s meet at Starbucks} \]

The key contributions of this work are:

• An algorithm for the automatic detection of abbreviations: The algorithm detects abbreviated words in user’s input and returns the start and end of abbreviated input. It potentially frees users from manually specifying which part of text is abbreviated and needs to be expanded.

• An algorithm for expanding abbreviated input into a full expression: The algorithm allows users to use non-predefined abbreviation. For example, users can successfully use \( a \ i \ w d \ l k \ t o \) or \( n \ i \ w u \ l i \ t o \), or other abbreviations for the phrase \( \text{and I would like to} \). The algorithm can also recover omitted words in abbreviated input, which may occur by accident or intention.

• The evaluation of detection and expansion accuracy through a corpus study: The algorithm for detecting abbreviation was validated against 6,000 words and achieved 90% accuracy. The algorithm for expanding abbreviated input was evaluated on 100,000 phrases and achieved 95% accuracy.

• The evaluation of time and keystroke savings through a user study: Time and keystroke savings of the new approach were measured compared to the standard iPhone text entry system. Keystroke savings were 32% on average. Input time increased 25% for novice users, but decreased 26% for experienced users after one hour of practice using the system.

Subsequent to the discussion of related work, we describe models and algorithms for multiple-word auto-completion. We present estimation methods to learn the probability distributions and the pa-
rameters required by the models. Then we describe a user interface for auto-completion of multiple words. Then, method and results of two evaluations of the new text input method are presented. Finally, we discuss future work and present conclusions.

2. RELATED WORK

The concept of increasing text input efficiency by auto-completion of the input is well established in literature. Early approaches were based on the prediction of characters based on prior keystrokes [1]. Later on, the focus shifted to word completion. Willis et al. presented a single-word completion system for persons with disabilities [2], a list of candidate words was created based on the abbreviated form introduced by the user. The approach employed a n-gram word model, and it put more weight on recently used words. Only a few systems explored the problem of auto-completion of multiple words. Shieber and Nelken [3] investigated using a compressed form of anticipated text, which was generated by applying strict abbreviation rules on each word, as a query for auto-completion of multiple words. One usability issue with the approach was that quite a few users unknowingly made small deviations from the strict rules, which resulted in de-compression failures. Another system, called Abbreviation Completion [4], had been developed for code completion of multiple keywords from abbreviated input. Because the technique did not support automatic detection of abbreviations, it required users to specify the beginning and end of abbreviated input, which can be a time-consuming task to do on a mobile device. While two previous systems focused on inferring full expressions from ambiguous input, Nandi and Jagadish [5] proposed an approach to infer the next words from the currently typed words.

3. MODEL AND ALGORITHM

This section first describes the abbreviation detection algorithm, which takes a string and classifies it according to whether it is an abbreviation or not. Then we describe the abbreviation expansion algorithm, which takes abbreviated words and turns them into a full expression of the abbreviated words.

3.1 Abbreviation Detection as a Supervised Machine Learning Problem

The approach for abbreviation detection presented in this paper is based on the assumption that expanded and abbreviated forms for words can be classified in two groups based on their similarity (or lack thereof) with the basic character pattern of the English language. Similar approaches are currently used successfully for the task of language identification (i.e. find the language of a given string).

Abbreviated input is processed by a Support Vector Machine (SVM) classifier. SVM is a well-established technique for case categorization: a set of cases, each described by a feature vector (i.e. a set of numerical attributes), is used to construct the optimal hyperplane that separates the original set into clusters of vectors of the same category, with the minimum error. An in-depth discussion of the SVM technique is presented by Vapnik [6].

In this work, the libSVM package [7] is used with the Radial basis function (RBF) kernel. The RBF function is a generalization of the standard linear kernel: It can handle nonlinear relations between class labels and is less susceptible to numerical difficulties. It is also particularly suited to the case when the number of cases is much larger than the set of features, requiring a mapping of the vectors to a higher dimensional space [8].

The string of characters representing the item to classify must first be transformed in a set of numerical quantities in order to be classified by the SVM machine according to whether it is an abbreviation or not. In our study, the attributes, which constitute the elements of the feature vector, are described in the following section.

3.1.1 N-gram similarity

An n-gram [9] is a subsequence of n items in a character sequence. In the literature, the term can describe any sequence formed with characters present in a string, but in the context of this paper it indicates only contiguous sequences. For instance, from the word abbreviation the following bigrams (n-grams of order 2) can be extracted:

\[ a \{ ab \mid bb \mid br \mid ev \mid vi \mid ia \mid at \mid ti \mid io \mid on \mid n \} \]

N-grams can be used to create a model of a language: the list of the most frequent n-grams in the language and their frequencies is compiled in the training phase. Such a ranking can be thought of as the encoding of the character pattern of that language. If the frequency profile is embedded in a vector space (Fig. 1), two models (i.e. the English language model and the encoding obtained from the word abbreviation) can be compared using a measure of distance, such as cosine similarity [10]. The cosine of the angle between the vector representations of the testing string and the language represents the similarity between them, and it is used as an attribute in the feature vector.

![Figure 1: Representation of word and language bigram rankings in vector space.](image)

3.1.2 Markov model of text

A Markov Model is a stochastic process where the probability of the next state depends only on the current state. The probability for a sequence S of n states s can be expressed as [11]

\[ P(S) = P(s_1, \ldots, s_n) = P(s_1) \prod_{i=2}^{n} P(s_i|s_{i-1}) \]  

where the conditional probability of a transition from two states is denoted \( P(s_i|s_{i-1}) \) and called the transition probability. A simple model of the language can be built as a Markov Chain using characters as states and string of text as sequences. The Markov assumption is particularly restrictive, allowing only the consideration...
of the emission probability for the next character dependent on the current one. This limitation can be removed by extending the dependency to the last $k$ states. For instance, the transition probability matrix for a second order chain looks like

\[
\begin{array}{ccccccccc}
ab & .06 & .25 & .24 & .05 & .13 & .15 & .06 & .31 \\
ab & .04 & .05 & .07 & .06 & .9 & .3 & .35 & .12 \\
\end{array}
\]

The probability for a generic string to be generated by the language model can therefore be computed as (2).

\[
P(S) = P(s_1) \prod_{2}^{n} P(s_i|s_{i-1}, \ldots, s_{i-k})
\]

The value is then normalized to remove the dependency on the length of the string and used as an attribute in the feature vector.

### 3.1.3 Other features

Other features considered are the length of the string and the ratio between vowels and consonants in the text. Research on human abbreviation behavior [11] shows a general tendency to drop vowels over consonants and to remove the last characters in long words. If that is the case, some important information may be found in these quantities.

### 3.2 Auto-completion as a Decoding Problem of a Hidden Markov Model

We propose modeling auto-completion of multiple words from abbreviated input as a decoding problem of a particular Hidden Markov model (HMM). A HMM is a graphical model that involves two sequences that are associated based on probabilistic dependencies. One of these sequences, called an observation symbol sequence, is observable (known), while the other sequence, called a hidden state sequence, is not directly observable. The process of inferring the most likely hidden state sequence from an observation symbol sequence is called decoding. Because all possible values of a hidden state are known in most HMMs, including ours, decoding is a process of finding the most likely state value assignment to a hidden state sequence from an observation symbol sequence using probabilistic dependencies. The Viterbi algorithm [12] is an efficient dynamic programming algorithm for decoding a HMM. In the proposed HMM, the user’s abbreviated input is modeled as an observation symbol sequence, in which each observation symbol is an abbreviated word that needs to be expanded. Candidates for a full expression of the abbreviated input are modeled as hidden state sequences. The N-most likely sequences found by the decoding process are returned as auto-completion candidates. All possible values of a hidden state are collected from a corpus of text. Figure 2 shows an example of an HMM for decoding abbreviated input I would like to.

![Figure 2: Auto-completion of multiple words is solved as a decoding problem of a Hidden Markov Model.](image)

### 3.3 An Extended HMM

We have extended a standard HMM in two ways to take the following characteristics of our problem domain into account:

- The model should be able to recover omitted words in abbreviated input.

- The number of observation symbols is not finite because users are allowed to abbreviate keywords in their own, non-predefined way.

The first extension is needed to enhance usability of the approach by allowing users to omit some words in abbreviated input. Omission of words may occur by accident or intention. Users may forget to type some word in abbreviated input and may not notice the mistake. For instance, a user may type pl fd att as abbreviated input for Please find the attached missing the in the middle, possibly because it is more difficult to notice missing words when a phrase is presented in an abbreviated form. Some users may intentionally utilize this capability to accelerate typing. For example, such users can type hd mtg heidi to achieve had a meeting with Heidi, saving keystrokes for two words.

Although it is possible to design a system that recovers multiple omitted words of any kind, we decided to make it recover only one word that belongs to a subset of English words based on the following considerations. First, a particular subset of English words, notably articles (e.g. a, an, the) and prepositions (e.g. at, with, or), are likely to be a primary target of word omission because they are frequently used as a grammatical glue to combine keywords into a phrase. Second, allowing a maximum one word omission would not be too restrictive, since articles or prepositions do not usually occur consecutively. Note that the restriction will prevent users from omitting a preposition followed by an article (e.g. at the) together. Overall, the restriction is expected to be beneficial by improving accuracy and efficiency of omission recovery because it will effectively reduce the search space by removing unlikely recovery candidates.

Decoding of a HMM that may have an omission in an observation symbol sequence has been investigated as an extension of HMM, called Hidden Semi-Markov Model (HSMM) [13]. HSMM introduces additional hidden states between each hidden state, which have null observations. Figure 3 shows an HSMM for decoding abbreviated input pl fd att. Additional hidden states are added between each hidden state for the recovery of omitted words, such as the in this case.

![Figure 3: An extension of HMM, called HSMM, for decoding abbreviated input as well as supporting the recovery of omitted words.](image)
ity as a constant based on an assumption that users are expected to omit any article or preposition roughly with an equal probability. This assumption has not been validated yet because no relevant data were available at this time for a text entry system as proposed in this work. Validation may be possible in the future once a reasonable amount of usage data of our text entry systems has been collected. During the corpus study for evaluating accuracy of omission recovery (Section 5), the constant has been set to 0.1, indicating that an article or a preposition is expected to be omitted once in every ten occurrences.

The second is a probability distribution of a state remaining in the same state in the next time step, i.e., a word being followed by the next word without insertion of any omitted word, denoted as $D(s)$. Since we do not have access to relevant data to learn the probability distribution, we decided to use a transition probability distribution as its approximation. This may be justified based on a simple analogy: A word remaining in the same state in the next time step is similar, in effect, to a word continuing transition toward the next word in the next time step.

Equation \(1\) summarizes the approximation of two additional probability distributions of a standard HMM given the limited access to relevant training data. \(E(\tilde{\theta}|s_t)\) denotes the probability of state \(s_t\) emitting null observation; \(D(s_t)\) the probability of a state remaining in the current state in the next time step; and \(T(s_{t+1}|s_t)\) the probability of state \(s_t\) making transition to state \(s_{t+1}\) in the next time step.

\[
E(\tilde{\theta}|s_t) \equiv \text{const} \quad \text{if } s_t \text{ is a recovered state } (s_t \neq \text{null}) \quad (3)
\]

\[
E(\tilde{\theta}|s_t) = 1 \quad \text{if } s_t \text{ remains the same as previous state } (s_t = s_{t-1}) \quad (4)
\]

\[
D(s_t) = T(s_{t+1}|s_t) \quad (5)
\]

A second extension to a standard HMM is necessary to support non-predefined abbreviations. Because there are infinitely many correct or incorrect abbreviations for a certain word, the emission probability cannot be directly computed by counting. This necessitates the use of an estimation model that can predict the emission probability for any given pair of original and abbreviated words. However, it is very difficult to develop a valid estimation model satisfying the basic property of an observation emission distribution—that its sum over all possible observation symbols (all possible abbreviations) is one.

We took an approach introduced by Han et al. in [4] to address this problem by modifying the structure of an HMM. As shown in Figure 4, the modified HMM incorporates match indicator nodes, denoted as $y_i$, whose values become one if an observation symbol (an abbreviated word) is a correct match of a hidden state (an original word) and zero otherwise. The probability of a match indicator becoming one is called a match probability.

![Figure 4: A modified HSMM with match indicator nodes.](image)

It is straightforward to develop a valid estimation model for the match probability because it is a probability of finite, binary events. Given the setup with two extensions to support omission recovery and non-predefined abbreviations, the Viterbi algorithm can be used to find the most likely state sequence from an observation symbol sequence. Finally, an $N$-best backtracking algorithm has been implemented based on [15], because it is desirable for an auto-completion system to provide $N$-most likely candidates instead of only one.

4. PARAMETER ESTIMATION

4.1 SVM Training

SVM is a supervised learning method: to build the model a number of training examples need to be prepared and labeled with the appropriate category identifier. The training procedure [8] involves a grid-search for the best value of the penalty factor, denoted as $C$, and slack parameter, denoted as $\gamma$, based on an $n$-fold cross validation. The training dataset is partitioned into a training group and several independent test subsets: each of the $n$ subsets is then used to test the model trained on the remaining $n-1$ subsets. The folding mimizes the impact of data dependency, thus improving the reliability of the results.

The training procedure consists of the following steps:

1. Compose the feature vector for every case using the attributes discussed in Section 3.2.

2. Scale linearly the values for each feature in the set of vectors to the range $[-1, +1]$. The same scale factors are used for the scaling of the testing data.

3. Consider a grid space of $(C, \gamma)$ with $\log_2 C \in [-5, 12]$ and $\log_2 \gamma \in [-5, 12]$.

4. For each pair $(C, \gamma)$ in the grid space, perform a 5-fold cross validation.

5. Choose the parameter pair $(C', \gamma')$ that results in the best classification rate.

6. Create the classifier model using the chosen parameters.

4.2 Estimation of HMM Parameters

A standard HMM has been extended to an HSMM with match indicator nodes to support the recovery of omitted words and the use of non-predefined abbreviation. The HSMM requires five model parameters to be estimated: start probabilities, transition probabilities, match probabilities with non-null observation, match probabilities with null observation, and state duration probabilities, denoted as $T(s_t)$, $T(s_{t+1}|s_t)$, $M(y_t = 1|x_t \neq \text{null}, s_t)$, $M(y_t = 1|x_t = \text{null}, s_t)$, and $D(s_t)$, respectively. Approximation techniques for estimating the last two parameters have already been described in the previous section. Note that we can equate $M(y_t = 1|x_t = \text{null}, s_t)$ and $E(\tilde{\theta}|s_t)$ because an observation symbol for omitted words is always null. This section describes the estimation method for the three remaining parameters. Start and transition probabilities are estimated by counting state transitions (word transitions) in a text corpus. Among many possible choices of text corpuses, e-mail messages in a user’s sent message folder were expected to be a useful data source to learn relevant state transition patterns. For users with few sent email messages, public text corpus, such as the British National Corpus (BNC) [16], may be used as an additional data source for training. Assuming a relevant text corpus is available for training, the following procedure is used to calculate the maximum likelihood estimates for start and transition probabilities: First, split each body of text in the corpus at punctuation marks and white-spaces, converting the corpus into a stream of words (a
state sequence). Second, count the number of transitions from one state to another as well as the number of occurrences of each state. The first counter is denoted as $Count(i, s)$, and the second one is denoted as $Count(i, s \rightarrow s')$. Third, calculate the maximum likelihood estimates for transition probabilities and start probabilities denoted as $\hat{T}(s)$ and $\hat{T}(s'|s)$.

\[
\hat{T}(s) = \frac{\sum_i Count(i, s)}{\sum_i \sum_j Count(i, s)}
\]

\[
\hat{T}(s'|s) = \frac{\sum_i Count(i, s \rightarrow s')}{\sum_i \sum_j Count(i, s \rightarrow s')}
\]

Match probabilities are estimated from examples of correct and incorrect abbreviation pairs. An abbreviation pair refers to a pair of an original word and an abbreviated word. A total of 4349 correct abbreviation pairs were generated by human volunteers as described later in Section 6.1.1. A total of 4349 incorrect abbreviation pairs were also generated by pairing an original word with a randomly chosen, wrong abbreviation. Then, we trained a logistic regression model that can predict match probabilities for any given word-abbreviation pair. The logistic regression model is expected to give some value close to one if an abbreviation pair has a good match, and some value close to zero otherwise. To prepare training data for the logistic regression model, the correct and incorrect examples were converted into a feature vector representation, with each feature element representing a different text-based similarity between a word and an abbreviation. The list of similarity features can be found in Table 1.

### Table 1: Similarity features used in the logistic regression model for estimating match probabilities.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Sim_1(s, x)$</td>
<td>number of consonant letter matches</td>
</tr>
<tr>
<td>$Sim_2(s, x)$</td>
<td>length of matched prefix</td>
</tr>
<tr>
<td>$Sim_3(s, x)$</td>
<td>length of matched postfix</td>
</tr>
<tr>
<td>$Sim_4(s, x)$</td>
<td>1 if $s$ and $x$ are the same; 0 otherwise</td>
</tr>
<tr>
<td>$Sim_5(s, x)$</td>
<td>number of letter matches with ordering ignored</td>
</tr>
<tr>
<td>$Sim_6(s, x)$</td>
<td>number of letter matches with ordering enforced</td>
</tr>
<tr>
<td>$Sim_7(s, x)$</td>
<td>percentage of matched prefix or postfix letters in $s$</td>
</tr>
<tr>
<td>$Sim_8(s, x)$</td>
<td>percentage of matched consonant letters in $s$</td>
</tr>
<tr>
<td>$Sim_9(s, x)$</td>
<td>percentage of matched letters in $s$</td>
</tr>
<tr>
<td>$Sim_{10}(s, x)$</td>
<td>percentage of matched letters in $x$</td>
</tr>
</tbody>
</table>

5. USER INTERFACE

5.1 User Interface Design

This section describes a user interface for text input using ad-hoc abbreviations. The design requirements were identified as:

1. Acceptance of both abbreviated and regular text input;
2. Automatic identification of abbreviated expressions and efficient display of completion candidates;
3. Possibility for the user to override the response of the system, either selecting unidentified abbreviated input for completion or ignoring system suggestions.

The design process consisted of three design-and-test iterations, performed using a paper mockup, a web application, and a native iPhone application. Initially, an additional requirement was considered: allowing the user to edit the single completion candidates when none of them was a perfect match. The first round of testing suggested that this functionality was not needed: the users preferred a more lightweight interface and were satisfied with manually editing the text when needed. The requirement was therefore dropped.

A screenshot from the resulting interface is shown in Figure 5. The upper half of the screen hosts the text input area, while a button bar on top of the keyboard accommodates the commands for the management of the completion system.

![Figure 5: iComplete application screenshot.](image)

5.2 User Interaction

When the user types, inserted text is parsed and checked for abbreviation. If at least $N - 1$ tokens out of the last $N$ are recognized as abbreviated forms, all the sequence is highlighted and the best four candidates sorted based on their likelihood are shown in a box next to the caret position. If the user keeps typing, the text in the suggestion box is updated. If one of the candidates matches what the user intends to write, the respective line can be selected with a touch and the abbreviated expression is replaced with the correspondent full form. If the auto detection fails to identify the correct abbreviation sequence, the user can override the selection using the two buttons on the left of the bar on top of the keyboard. The buttons to cancel the selection and clear the abbreviated text are also located on the bar, together with a button to show a larger number of candidates in a full screen table view. From the navigation bar on the top of the screen the user can access the configuration page.

5.3 Prototype Implementation

The interface was implemented in a test application called *iComplete* using the Apple iPhone SDK 3.12. The application is based on a client-server architecture: the server side provides the abbreviation detection and decoding functionality and the client side manages the user interface. The client-server scheme was chosen for the following reasons: First, it is well suited for rapid prototyping of a test application. Second, it allows more freedom on the training phase because of the impossibility of accessing user data in a sandbox-based environment such the particular platform used. Third, it not only avoids time-intensive optimization needed to run the system on a particular device but it also makes testing the approach on different devices more economic.

6. ARTIFICIAL CORPUS STUDY

6.1 Abbreviation Expander Testing

6.1.1 Preparation of training and testing example

To validate the abbreviation detection module a number of human-generated abbreviations were collected. A set of 1000 words were...
randomly extracted from the 5000 most frequent items in the British National Corpus and were divided in 50 subsets of 20 words each. The subsets were used to create a number of tasks to be submitted on Amazon Mechanical Turk (AMT) [17]. Workers were instructed to abbreviate the words under the following rules: (1) only letters in the original word should be used to compose the abbreviated form, (2) abbreviation should be of at least two characters. Every task was assigned 10 times. A total of 98 workers completed the resulting 500 tasks in about 48 hours. The resulting set of abbreviations was checked automatically, and the items non-compliant to the conditions discarded. A total of 4349 unique valid abbreviations were saved for the training and testing phases.

The training and testing datasets were composed using abbreviations from the AMT experiment and regular words from the BNC set in an equal number. Items were not repeated in the two sets. Every string in the two sets was transformed in the respective feature vector and labeled according to its category.

6.1.2 Test execution

A number of tests were conducted to populate the feature vector with different combinations of the following attributes:

- N-gram similarity \((n = 1\) and \(n = 2\)).
- Markov sequence probability \((k = 1\) and \(k = 2\)).
- Vowel/consonant ratio.
- Number of characters.

The SVM predictor was built on training sets of different size according to the procedure described at Section 4.1 and then used to classify the 6000 instances of the testing dataset. The predictions were compared with the class labels and the accuracy calculated as the ratio between the number of correct matches and the total size of the testing set.

6.1.3 Results discussion

The test was repeated employing different feature vectors and varying the training set sizes (250, 500, 750, 1000 and 1500 cases respectively). The results show that the accuracy initially increases with the size, but the variation becomes very small going from 1000 to 1500 instances in the training set. Classifications performed using a single attribute show accuracies between 74% and 85%, while using a complete feature vector the classification is correct in the 89.65% of the cases. The error rate was not equally split, with a predominance of false negative (i.e. abbreviation mistaken for a regular word). This is due to the difficulty in identifying abbreviated forms very close to the language pattern (i.e. abbreviations from the AMT experiment and regular words from the BNC).

The artificial corpus study aims to evaluate the accuracy of the iComplete system. A total of 100,000 frequent phrases were extracted and every word automatically abbreviated following 3 different procedures:

1. Keep the first character unchanged. Drop vowels and replace double consonants with single ones in 80% of the median value.
2. Keep the first two or three letters. If a word is equal to or shorter than 8 letters, first two letters are kept. If a word is longer than 8 letters, first three letters are kept. One-letter words are left unchanged.
3. Abbreviate the words like in (1), omitting one word if it appears in a list of common English prepositions.

By adopting several different abbreviation approaches the effect of a possible bias in the choice should be reduced, thus providing a reasonable estimate of the system performance in real working conditions. For the first two approaches 2000 testing phrases were generated for each user, ranging uniformly from 2 to 5 words. In the third case 200 abbreviated phrases of 4 and 5 words each were composed for the testing.
### 6.2.4 Results

For every user, the following test procedure was followed:

1. Train the HMM model on the full set of messages in the user sent box.
2. Expand the abbreviated test lines using the HMM and record the number of successful expansions and the rank of the successful candidate.
3. Calculated top-\(N\) accuracy by dividing the occurrences of successful expansion within top-\(N\) candidates by the number of lines of abbreviated text tested.

The first test was conducted without considering word omission. The Top-10 accuracy against 100,000 phrases written by 50 different individuals was 97% for the first abbreviation style and 93% for the second encoding. The same set was tested again, this time considering the possibility of omitted words. The Top-10 accuracy decreased to 92.5% and 88.3% respectively. Such decrease can be explained by the larger search space of the decoder. The processing time was also significantly larger. Analyzing the detailed results in Table 3, one can see how the accuracy values are consistent between the different phrase lengths. This result is significant since it allows the user to enter a larger number of abbreviated words before replacing with a suggestion without any penalty in accuracy. A test run with third style (omitting a common preposition in 4- and 5-word phrases) resulted in an average Top-10 accuracy of 76%.

### 7. USER STUDY

The user study was intended to measure the possible time and keystroke savings of the new approach compared to the iPhone text entry system. The study reported keystroke savings with average 32% decrease amongst users, while time for input varied consideraby depending on the user familiarity with the approach, increasing 25% for novice users. Observations suggest that users were spending time thinking about how to abbreviate words, while typing in a more natural way when using the standard system to which they were accustomed. More extended tests suggested that a time saving of 26% can be achieved when the user is allowed to gain experience with the completion system.

#### 7.1 Scenario

The evaluation took place in a particular test scenario: a user is writing an email message and needs to enter a particular phrase. The user already has a concrete idea of what needs to be written, so the text introduction can be performed without any interruption. To simulate such scenario with the best approximation, the testers were provided with a reference of the text line to insert on a paper sheet placed in front of them. Participants were encouraged to memorize the phrase before starting typing, in order to enable them to focus on the system when typing instead of spending time switching between the instruction and the application.

#### 7.2 Experiment Setup

A total of 10 participants were recruited among graduate students on a college campus. The testers were informed that the user study would take about 30 minutes and one of the participants would be awarded a $20 gift certificate. The group consisted of 8 males and 2 females, with an average age of 28.2 years. All of the users were familiar with the iPhone text entry system, with different levels of proficiency. The user study was conducted in a laboratory space, using the test application running on an Apple iPod Touch© (totally equivalent to the iPhone for the purpose of this study). Each volunteer was assisted by an experiment facilitator and performed two series of text entering tasks, using both iComplete and the standard input system of the device. The test phrases were extracted from 3 different user sets obtained from the Enron dataset. The sets were chosen randomly within the range (-20%, +20%) from the median size of the Enron corpus. For each of the users 4 phrases of 2, 3, 4, and 5 words respectively were randomly selected, resulting in a series of 12 phrases. The experiment consisted of 4 tasks:

1. Practice iComplete under the guidance of the facilitator, asking any question about the use of the system.
2. Practice the standard text input system. This part was conducted to avoid any bias, even though all of the participants were already familiar with the device.
3. Input the 12 test phrases using iComplete.
4. Type the 12 test phrases using the iPhone standard text input system.

To balance the experiment, half of the subjects performed task 3.
and 4 in reverse order. After the insertion of every phrase the user handed the device back to the facilitator to set up the system for the next line. Time and keystroke usages were recorded automatically by the test application. In addition, the participants were asked to fill out a short survey about the test experience.

7.3 Results

The user study showed a keystroke saving of 32.4% but an increase in time 28.3% for this group of novice users. The reduction in keystroke usage was consistent amongst the participants, ranging from 11.8% to 49.8% depending on the style of abbreviations they used. It was noticed a tendency for some users to choose long abbreviation, even when showed in the training phase that this was not necessary. This leads to believe that a further decrease in keystrokes usage is possible once the user has gained some experience in the use of the system. The time variation was not uniform amongst the set of phrases. In general the system was less effective for the input of very short phrases, while savings could be achieved for longer texts. The saving of keystrokes was expected to determine a decrease in the amount of time needed as well. Observation suggests that did not occur for a number of reasons: (1) the user were not abbreviating in a natural way; instead they were constantly trying to think of “good” abbreviations, slowing down the process; (2) similarly, they were spending time checking more than once the phrase to enter on the instruction sheet, (3) as mentioned, some of the testers were using overcomplicated abbreviations, even though being instructed to try to use the system in all its potentiality; finally (4), the approach introduces a time requirement to validate the system’s suggestion.

Additional training and experience in the use of the system is expected to reduce the inefficiencies in at least the first four areas. To evaluate this assumption, four participants were asked to practice the use of the system for an hour before repeating the test (in comparison to the 10 minutes of practice in the novice test). The results show a sensible improvement in the performance, with average time saving of 26%.

At the conclusion of the test, the participants were asked to fill out a questionnaire about their experience using the system. It is interesting to observe how the participants correctly believed to have saved keystrokes, but testers had the perception of saving time as well. Overall the tester expressed their interest in using abbreviation auto-completion for text entry on their device.

8. CONCLUSION

A new method for improving efficiency of text entry on mobile devices using ad-hoc abbreviations was presented. An algorithm to detect abbreviations in text input using SVM was presented and validates against 6,000 abbreviated or unabbreviated words and achieved 90% accuracy. A HMM learned from a text corpus was used to expand abbreviations into full expressions. The abbreviation expander was evaluated on 100,000 phrases and achieved 95% accuracy. A user interface was designed and implemented in a test application for Apple iPhone. The application was used to conduct a user study: 10 participants were asked to use iComplete to enter a predefined set of phrases extracted from an email corpus. Time and keystroke savings of the new approach were measured compared to the standard iPhone text entry system. Keystroke savings were consistent amongst users, with a 32% decrease on average. Input time varied considerably depending on familiarity with the approach, resulting in an average increase of 25% for novice users. A more extended study revealed that time saving of 26% can be achieved by users with an hour of practice using the system.

Future work will be directed to improve the efficiency of the system in order to allow for a direct implementation on the device. Another promising line of work is in text prediction: the system may be enhanced to complete following words from abbreviated input as well as original words. Such a capability should further improve the efficiency of typing.

9. ACKNOWLEDGMENTS

We thank MIT CADlab and UID group members who provided a creative environment and helpful discussions, and study participants for their time and comments.

10. REFERENCES