PreText: A Predictive Text Entry System for Mobile Phones

Deepti Nalavade, Tushar Mahule, Harshvardhan Ketkar

Abstract: Majority of the currently used predictive text entry systems (like T9 for lower end mobile phones) do not provide word prediction. In these systems, the average number of key-taps per word is high resulting in higher typing efforts on the part of the user. At times,
T9 provides options (words) that may not fit into the context of the message, are wrong grammatically and are not valid English words. Also, T9 is slower to adapt to usage patterns.
PreText predicts the word that the user is typing with the help of grammar rules for the English language, making word prediction more precise, reducing the number of key taps required, saving the user's time and achieving an optimisation over the existing systems. It also adapts to the user's usage pattern with the help of a frequency model. The metric used here to evaluate the performance of text entry systems is *KSPC [1] (*keystrokes per character). The KSPC was found to be 0.7360 for PreText, providing an average improvement of 26.91% over T9 which has a KSPC of 1.023 [1].

Keywords: text entry, predictive, adaptive, KSPC

1. Introduction:

With the advent of smart phones, text entry is required for a host of applications other than text messaging like email, which require fast text entry.

With a text entry system like T9, more effort is required by the user as the system may not always throw up the desired word (for example, while typing "home", T9 frequently predicts "good" when what you want is "home", and the other way round). So the need for a word prediction utility, which predicts the right word in the right place, was felt.
Considering the behavior of T9, it can be said that it offers word completion and not word prediction. The average number of key-taps per word is high. In the best case, for T9, number of key taps for each word is equal to the number of letters in the word. As
T9 works primarily on permutations of letters; it also provides many non-English words as output.
PreText is developed to predict the word that the user is typing with the help of grammar rules for the English language. It adapts to the user's usage pattern with the help a frequency model, referring to the usage frequency for each word and its part of speech.
When the text is being entered, the expected part of speech is found out. The system will predict the next word taking into account already typed word or words to get probable words from the dictionary using an index search mechanism and display words taking into consideration the frequency count.
The user shall also be able to enter any word that is not present in the dictionary. Runtime conjugation of verbs, resolution of same words occurring in multiple parts of speech and handling of punctuation marks are some of the distinguishing features of PreText.

2. Related work:

A thesis by Afsaneh Fazly – "The Use of Syntax in Word Completion Utilities" [2] has proposed the statistical and syntactical prediction of words using part of speech tags and a bigram model. "A Swedish Grammar for Word Prediction" [3] has been developed by Ebba Gustavii and Eva Pettersson where a Swedish grammar for the FASTY word predictor has been defined and implemented in which the grammar functions as a grammar checking filter, re-ranking the suggestions proposed by a statistic n-gram model on the basis of both confirming and rejecting rules.

3. System model:

PreText is to be implemented on a Linux based mobile platform. However, the prototype is developed on Ubuntu Linux on a standard PC using the C language. The ambiguous, non-QWERTY mobile phone keypad is simulated on the computer keyboard using the number keypad. The Graphical User Interface (GUI)

used to simulate the text entry in a mobile phone makes use of GTK+ (Gimp Toolkit).
PreText assumes that the user must be conversant with the English language and that he enters approximately correct grammatical sentences in English. However, even if the user does not use grammatically correct English at times, he is not denied any word that he wants to type.
For words which are not present by default in the dictionary, the user must feed words into it. (For example short forms, names etc.
)
When the user enters a character, a drop-down list of options is generated. On entering another character, the list of options is refreshed and a new list of options is generated. Thus, the search in the dictionary gets narrowed to fewer words.

4. Problem Statement:

With the existing text entry systems in mobile phones, the average number of key taps per word is usually greater than the number of letters present in the word. This results in higher typing efforts on the part of the user. It is necessary to improve the user's typing speed by intelligent word prediction, eventually reducing the number of key-taps.

5. Proposed Solution:

PreText consists of

- A concise dictionary (segregated into part of speech files) of English words with their frequency counts.
- An N-level index search mechanism to search a word, given a portion of it.
- A bigram [2] and trigram [2] grammar structure.
- An adaptive mechanism making use of frequency counts associated with words as well as grammar rules.
- A verb sub-grammar in order to conjugate verbs.
- A facility for users to enter user-defined words and a GUI consisting of a text area to type text and a drop-down list of options which is scrollable.

The dictionary does not implement a tree or linked list structure due to the overhead of handling pointers. Also, instead of using a CFG (Context Free Grammar) model, including noun phrases and verb phrases, a bigram and trigram model was preferred since the context of a text message in a mobile phone is small and the user may not adhere to grammar beyond three words in a sentence.
The following diagram shows a higher level flow of the system as a whole:

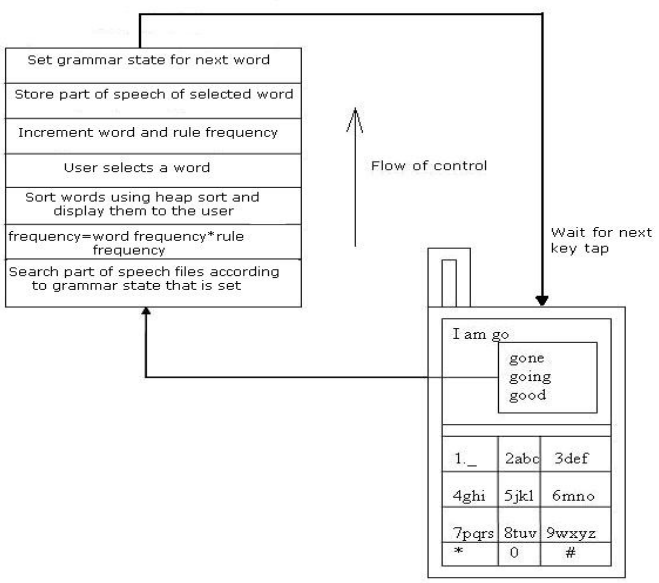

Fig 1: System model

Word searching and list retrieval:

files, each for a specific part of speech and one file for the user entered words which are not in the dictionary. The words in the files are previously sorted in ascending order of the digits. For example, the file for conjunctions is stored as:

2586844 **although** 234
263 **and** 4435
288 **but** 543
43 **if** 2354
74623 **since** 54
8436 **then** 2325
86844 **tough** 45
94453 **while** 65

Here, (the number on the left of the word (for example '2586844' for 'although') is the key code from the ambiguous mobile key pad for the word. The number on the right (for example '234' for 'although') is the frequency count of the word.

When the system is started for the first time, the offset bytes, from the byte at the start of the file, of words starting with distinct numbers from distinct files are recorded in a 2D array. This array is shown in the "Fig 2" below as the 'Offset Array'. The rows represent 9 out of the total 14 different parts of speech files in the "Fig 2" given below. The columns represent the first digit of the key code of a word.
Now, consider that the user is typing the word 'the' on the mobile. Using the grammar model, the possible parts of speech that the word might belong to is known. The system now

An n-level index search mechanism is used to search for words in the dictionary. The dictionary is primarily divided into separate scans the 'Offset Array' (in Fig 2) for the digit 8 in the parts of speech files that have been zeroed in on.

When PreText gets this offset byte, it jumps to that offset byte in the particular file, starts making a list of words that the user might intend to use and displays the top 5 words based on the adaptive mechanism involving frequency counts. While this list is being generated, this time is used to compute in parallel the offsets of words with number codes as 82, 83, 84 … 89. This can be done assuming that the user will not press backspace and keep on typing to get the word he intends.

The user presses key number 4 now. Here, the offsets of the words with number code 84…. from the all the probable parts of speech have already been obtained. So, jumping to those locations, a new list of probable words is generated. Now, the same process is repeated for generating offsets for words with number codes 841, 842, 843 … 849.

The above process gets executed repeatedly for all the words, thus giving a list of probable words and at the same time computing the offsets for probable words when the next key is pressed. This pre computing helps in saving the time of searching for a word as the user types. It gives better results in a faster way in combination with the grammar rules owing to the pre computing n-level index search mechanism.

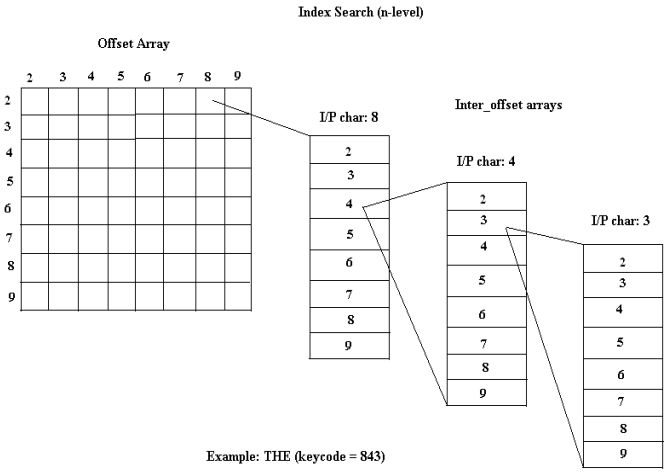

Fig 2: N-level Index Search

Grammar:

The grammar used contains a mixture of productions belonging to bigram and trigram models.

Bigram takes into consideration the context of the previous word and predicts the current possible word. For example:

Article → Noun

Trigram considers the past two words and predicts the current possible word. For example:

Noun → Verb → Adjective

Consider the following grammar rules:

Article → Noun

Article → Adjective

On entering a character, to search for a word, only the part of speech files that the grammar rules specify are searched. For example, in accordance with the above grammar rules, only the noun and adjective files will be searched to display options if the previously typed word is an article.
However, if the user attempts to enter a word, which is neither in the noun file, nor in the adjective then, in such a case, PreText searches the remaining part of speech files. PreText searches the remaining files only if no matching pattern is found in the grammatically valid files. Thus, PreText ensures that the whole dictionary is not searched, and hence the searching overhead is reduced.
If the desired pattern is not found even after searching the remaining files, then PreText assumes that the word is not present in the dictionary, and it prompts the user to add the word to the dictionary.
If the grammar is violated frequently, and the user consistently uses grammatically incorrect English, the degradation in performance is graceful, since the bigram and trigram model is used.

Frequency and Word Ranking Model:

The frequency and word-ranking model operates at two levels: word usage level and rule-based level.
Just as each word has a frequency, each grammar rule has usage frequency level associated with it.

For example, the following bigrams each have a frequency associated with them,

Article → Noun
(Rule frequency: 0.4)

Article → Adjective
(Rule frequency: 0.2)

This frequency is incremented if the rule is used that is, if the user selects a noun, the frequency of the corresponding rule **(Article → Noun)** will be incremented. Before sorting the options, the frequency of each word from a part of speech file is multiplied with the rule frequency of the part of speech. Thus the cumulative frequency of the two is found out. The list is then sorted in descending order of this cumulative frequency. This improves adaptability to the user's usage patterns. If a particular word is selected, the word frequency as well as the frequency of the grammar rule that has been used is incremented.

Verbs and User-defined words:

PreText also ensures that there is agreement in person and number for the verb predicted, with respect to the pronoun previously typed. For example, if the user wishes to type "he runs", then the verb 'to run' is conjugated at runtime. After the user has entered 'he', the form 'runs' is shown in the list of options, and 'run' is discarded. This modification in the infinitive is done at runtime. Only the infinitive 'run' is stored in the dictionary, and various rules are used to generate the correct conjugation of the verb. The system also conjugates the verb according to tense used. This results in efficient use of the limited memory resources of a mobile phone.
For user-defined words, a separate file is maintained and a relatively higher frequency is assigned to these words because the usage probability of these words is more since the user has entered them.
If the user does not encounter the desired word in the suggested list of words, then he continues typing till the word gets over. Thus, PreText gives a worst-case performance equivalent to the best case performance of T9.

6. Analysis:

In order to evaluate the performance of the system, *KSPC* is used. *KSPC* is an acronym for *keystrokes per character*. It is the number of keystrokes required, on average, to generate a character of text for a given text entry technique in a given language [1]. The lower the value for the *KSPC*, the lesser the typing efforts by the user and lesser the time required for entering text.
KSPC is computed as follows:

$$KSPC = \Sigma(Kw \times Fw) / \Sigma(Cw \times Fw)$$

Where,
Kw is the number of keystrokes required to enter a word,
Cw is the number of characters in the word, and
Fw is the frequency of the word in the corpus. [1]
Kw and *Cw* takes into consideration the terminating SPACE after each word. For the QWERTY keyboard, *KSPC* is equal to 1 since more than one letter does not have to share one single key. For word completion KSPC is greater than 1. However, with word prediction, there is potential for *KSPC* to be less than 1 because words can be entered without explicitly entering every letter, as is done in PreText. The complete word can be extracted and supplied from a portion of a word. A sorted list of words starting with the part of word typed by the user until that moment is displayed every time a letter of the word is entered. On observing the desired word in the list, the user selects the word. The number of keystrokes to enter the word is determined and the *KSPC* is computed.
For PreText, the number of keystrokes to enter a word includes the number of characters in the word stem at the point where the intended word appears in the candidate list, and the keystroke overhead to select the intended word in the option list wherein the user selects the word and adds a SPACE.

7. Simulations and Experimentation:

The system was tested on a number of English sentences. The words not present in the dictionary (proper nouns and short forms) were entered by the user as user defined words and stored separately.
It is found that for the Multi Tap mode of text entry, *KSPC* is 2.0342 while that for the dictionary mode (T9), it is 1.0072. Among 100 test sentences, the following “Table I” shows the 10 sentences that were tested on PreText in which results range from an improvement of 9% to as good as 36%: an average of the performance of PreText in comparison to T9, for each sentence gave the figure of 26.91% and the *KSPC* was calculated to be 0.7360.

Table I: Test Results

Sr. no	***Sentence***	***KSPC (Key Strokes Per Character)***	***Performance (as compared to T9 KSPC of 1.0072)***
1.	come tomorrow at his place	0.629862	+37.46%
2.	the boys run slowly	0.829815	+17.61%
3.	come to the mystery spot	0.638690	+36.58%
4.	gnsd sweetu	0.421453	+ 58.15%
5.	have you had enough food?	0.701082	+30.39%
6.	both are same	0.909692	+9.68%
7.	you are made for each other	0.784206	+22.13%
8.	your project is good	0.752143	+25.32%
9.	the competition is tough	0.458880	+ 54.43%
10.	btw i forgot to tell that henry went home	0.824244	+18.16%

8. Conclusion:

Thus, PreText, on an average, provides an improvement of 26.91% over T9 in terms of KSPC.
This means that the user can type

more in lesser number of key taps. Therefore, at least by initial perceptions, PreText provides better performance in terms of number of key taps taken to type a word. However, better testing techniques might be required in the form of user trials, so that user acceptability and adaptability can be tested.

9. Future Work:

Provision for short forms using textual compression techniques:
PreText predicts the whole word to the user, unless the short form has already been entered as a user-defined word. The user may run out of space to type further and also, may not be inclined to enter the short form into the dictionary every time. A solution to this can be to compress the contents of the message textually: i.e. 'coming' becomes 'cmng', and so on.

Prediction of options after using backspace:
When the user makes a mistake while typing, he backtracks by pressing backspace single or multiple times. This disturbs the grammar. It is imperative that the user gets optimal performance even after he backtracks. Therefore, the grammar must backtrack accordingly. Currently, in the prototype developed, the grammar is reset after the user presses backspace.

ACKNOWLEDGMENT

Thanks for the guidance from Ms. Dimple Kuriakose (Lecturer, Vishwakarma Institute of Technology, Pune, India).

REFERENCES

[1] I. Scott MacKenzie. *KSPC (Keystrokes per Character) as a Characteristic of Text Entry Technique.* Dept. of Computer Science, York University, Toronto, Ontario, Canada M3J 1P3. smackenzie@acm.org

[2] Afsaneh Fazly. *The Use of Syntax in Word Completion Utilities*. A thesis submitted in conformity with the requirements for the degree of Master of Science, Graduate department of Computer Science, University of Toronto. (2002).

[3] Ebba Gustavii and Eva Pettersson {ebbag, evapet}@stp.ling.uu.se. *A Swedish Grammar for Word Prediction.* Master's thesis in Computational Linguistics Språkteknologiprogrammet (Language Engineering Programme) Uppsala University, Department of Linguistics.

[4] Hedy Kober, Eugene Skepner, Terry Jones, Howard Gutowitz and Scott MacKenzie. *Linguistically Optimized Text Entry on a Mobile Phone*. Submitted to CHI 2001.

[5] Mark D Dunlop (Centre for Human Machine Interaction, Risø National Laboratory, Denmark) and Andrew Crossan (Department of Computing Science, University of Glasgow, Scotland.) *Predictive Text Entry Methods for Mobile Phone.*