

PREDICTIVE TEXTING FOR SMS IN TAMIL.

Corresponding author: R.Shriram, Asst.Professor, TIFAC-CORE, Velammal Engineering College.

Co-authors: Ganesan R¹, Logesh, Vikranth, Vasanth²- Velammal Engineering College.

¹: Lecturer, TIFAC-CORE

²: B.E. CSE Students

ABSTRACT

For Tamil SMS to become a reality, predictive texting is absolutely necessary. Traditionally Predictive Text has been associated with Mobile Phones alone. At present Predictive Text is being developed by Mobile Phone manufacturers like NOKIA, SAMSUNG, Software vendors like Tegic Communications and Service providers like Oli. The availability of Predictive Text entry drives increased SMS usage. For this reason, In short, Predictive Text entry on reduced keypads will become part of the expected framework in which people will operate their phones. For a linguistic minority like Tamil in India, there is an urgent need for developing predictive texting solutions.

The contributions of thjs paper are that it

- Outlines our predictive texting approach
- Presents the early experimental results based on Tamil SMS solution.

INTRODUCTION AND LITERATURE REVIEW

Predictive Text input techniques strive to reduce the input burden by predicting what the user is entering. This is accomplished by analyzing a large collection of documents – a corpus – to establish the relative frequency of characters, digrams (pairs of characters), trigrams, words, or phrases in the language of interest. These statistical properties are used to suggest or predict letters or words as text is entered.

The seminal publication in the area of text prediction is by Shannon (1951), and, although there are many ways to implement text prediction, most are based upon this paper.

Predictive input technologies have the capacity to significantly reduce the effort required to enter text – if the prediction is good. There are a few caveats to consider in basing a language model on a standard corpus, however. These include: (a) the corpus may not be representative of the user language, (b) the corpus does not reflect the editing process, and, (c) the corpus does not reflect input modalities.

The characteristics of the text users enter are dependent on the application used to create the text. For example, we expect more formal prose using a word processor than an e-mail application. Additionally, the type of application depends upon the input device available – few people have the patience to enter volumes of text into a hand-held PDA device. The kind of text most likely to be entered in this context is short notes, phone numbers, URLs, acronyms, slang, etc., the

statistical properties of which differ from formal English texts. Highly cryptic messages are common for text entry on cell phones

A corpus contains no information about the editing process, and we feel this is an unfortunate omission. Users are fallible and the creation of a text message – or interaction with a system on a larger scale – involves much more than the perfect linear input of alphanumeric symbols. The input process is really the editing process.

T9, the dictionary-based Predictive Text entry software available from AOL, uses approximately 60K of memory for each language dictionary, and over 100K for some languages. To provide even five languages in a phone, manufacturers have been forced to dedicate over 300K of memory per phone. As memory is both limited and expensive, including T9 and one or more languages has a very significant cost, both in terms of money paid for memory and in terms of lost opportunity. Opportunities are lost because the existing memory could have been used to provide other features (additional games, larger address book, storage of more SMS messages, longer voice memos, screen savers, etc.).

Memory intensive dictionary-based Predictive Text solutions limit the ability of manufacturers to appeal to a wider market. Manufacturers must resort to dividing their markets geographically, offering different small subsets of languages in each. Manufacturers deliver embarrassing and awkward solutions that offer phone menus in over thirty languages, but Predictive Text software in just a subset of these. As a result, customers have to puzzle over setting their "phone language" versus their "Tegic language". The situation is made worse by the need for users to resort to Multi-Tap to enter names into their address book because the dictionary-based methods are useless for generalized entry of names.

In these ways, the use of memory-intensive dictionary-based Predictive Text solutions tightens the manufacturer's hand. With a product such as T9 or eZiText, even if the needed language databases were available, there is simply no way to make phones appeal to a wider linguistic audience without dedicating megabytes of memory to store these databases. Manufacturers who use these products are forced to draw gross linguistic and geographical divisions that necessarily allow the needs of hundreds of millions of people to fall through the cracks.

OUR APPROACH

Our studies here indicate that a typical user works over an average vocabulary of around 500 words. Hence an algorithm that learns by itself and quickly populates the data in the cache will be designed.

Also a link between the allied applications and the type of words being used is also noticed. For example if the user has a meeting at 3.00 pm, his messages during the time will be directly related to the subject of discourse viz. the meeting. Another trivial but significant aspect that comes out of messaging studies is the relation between incoming and outgoing messages. For example a message 'what time today' will inevitably generate a reply of say '7.45' etc. So a clue could be that messages can be parsed to categorize in contextual content or clues be extracted from the incoming messages to act as guides that dictate the word to be typed.

Such data guides can be extracted from the traffic and used to improve the overall efficiency of the algorithm. For that reason we envisage a post-release support period in which the software traffic can be monitored and linguistic guides be extracted for further study.

The system should also be robust enough to update itself and learn from others. The software developed will be an interesting case of how one system can learn from the experiences of others. A scenario wherein each users wordlist is fed into the main system and the word list being updated conjures up an image of Distributed Computation which is continuously being adapted and the results permeated all along the network.

This data from the hundreds of machines can be pondered over by the language researchers.

Thus the system will not only involve linguistic studies, but will test the limits of distributed data sharing and adaptive algorithms with scope of work in that order.

COMPARISON WITH MULTITAP MODEL

Multitap Model

This follows the normal English Keypad Layout of 3 and 4 characters per key.

- First SIX (1-6) keys are occupied by MEI EZHUTHUKKAL.
- The THREE keys (7, 8, and 9) are occupied by UYIR EZHUTHUKKAL.

A transition time between characters can be set according to our convenience. It is very important that we need to press the key 1 thrice within that transition time in order to get the desired letter. The same procedure needs to be followed for the construction of a word and thereby completing a sentence.

Let us consider the word மங்கி. The steps followed are listed below.

Key 4 is pressed once to get the letter 'ம'

|

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	ளறன6
அஆஇஈ7	உஊஎஏ8	ஐஒஔௌ9
* 00	0	#

Key 3 is pressed twice to get the letter 'ந'

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	ளறன6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key * is pressed once to get the dot

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	ளறன6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key 3 is pressed once to get the letter 'த'

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	ளறன6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key 7 is pressed thrice to get the letter 'இ'

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	ளறன6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

த+இ=தி

Key 4 is pressed thrice to get the letter 'ர'

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	ளறன6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key 7 is pressed thrice to get the letter 'இ'

கங்ச1	சூடண2	தநப3
மயர4	லவழ5	ளறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஒள9
* .ஃ	0	#

$$ர + இ = ற$$

So, the total number of keypresses needed here is **14** times.

Predictive Text (Agaram - Tamil Dictionary)

In this method, to get the required word, we have to press the keys in which the letter is present only once. Here, in order to type the word மந்திரி the following steps needs to be followed:

Key 4 is pressed once

கங்ச1	சூடண2	தநப3
மயர4	லவழ5	ளறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஒள9
* .ஃ	0	#

Key 3 is pressed once

கங்ச1	சூடண2	தநப3
மயர4	லவழ5	ளறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஒள9
* .ஃ	0	#

Key * is pressed once

கங்ச1	சூடண2	தநப3
மயர4	லவழ5	ளறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஒள9
* .ஃ	0	#

Key 3 is pressed once

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	எறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key 7 is pressed once

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	எறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key 4 is pressed once

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	எறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

Key 7 is pressed once

கங்ச1	ஞடண2	தநப3
மயர4	லவழ5	எறண6
அஆஇஈ7	உணஎஏ8	ஐஒஓஔ9
* .ஃ	0	#

So, the total number of keypresses needed here is only 7 times.

Thus the number of keypresses is drastically reduced from 14 times to 7 times which is a 50% reduction in keypresses as well as time.

Method of Implementation	Number of Keypresses
Multitap	14
Predictive Text(Agaram)	7

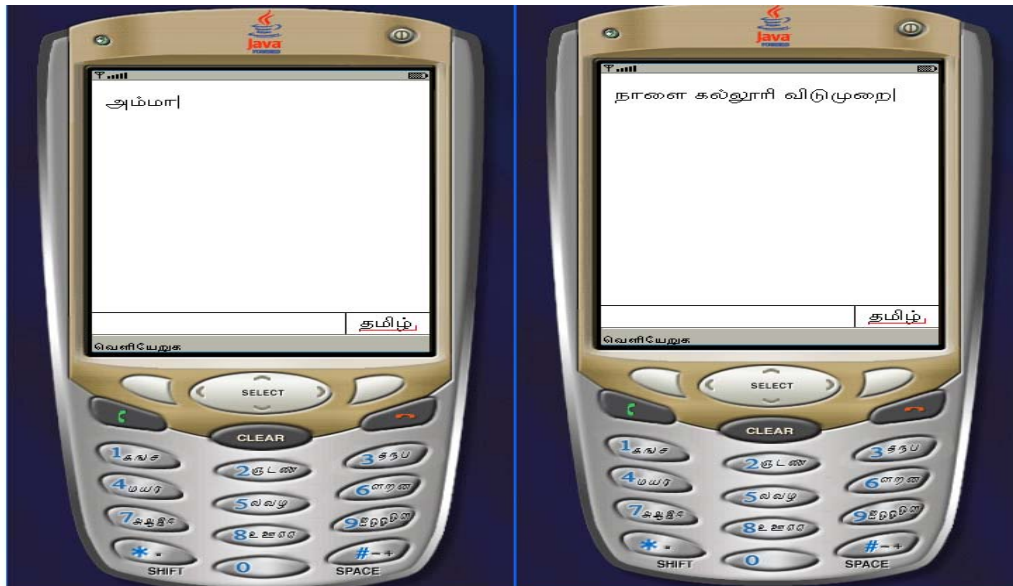
The comparisons of our algorithm with the T9 and eziText will be shown at the time of the conference.

Our Implementation of Predictive Text (Agaram)

'Agaram' has been developed using J2ME. Its execution is viewed through the Toolkit's Emulator Phone (A Simulation of the real mobile phone in computer). As words are embedded in the program's memory, it makes the search for words quicker. Since there is a drastic reduction in number of key presses it saves most of time and also the key presses.

We had implemented this application development software and thus achieved an efficient algorithm in developing the Tamil dictionary.

SNAPSHOTS:



The software is being designed taking into account the ongoing trials of the TAMIL SMS software from which we are collecting valuable data as to the preferences of the users, words frequently used from a broad user community.

CONCLUSION AND FUTURE WORK:

In this paper, we have traced the need for predictive texting and examined the problems involved in developing predictive text solutions. Agaram our predictive texting solution is discussed. Agaram uses internal database of Tamil words and contains approximately of about 750 words. In future the number of words can be increased to make it suitable for messaging. The software will also be integrated with the Tamil SMS software being developed in our research center to take it to the masses.