International Conference on Advances in Human Machine Interaction (HMI - 2016),
March 03-05, 2016, R. L. Jalappa Institute of Technology, Doddaballapur, Bangalore, India

Predictive and Corrective Text Input for desktop editor using n-grams and suffix trees

Akshay Bhatia , Amit Bharadia, Kunal Sawant, Swapnali Kurhade
Information Technology Department,
Sardar Patel Institute of Technology,
Mumbai, India.

***Abstract*—Even though desktop computers have existed since a long time, the method of typing and feeding input has not changed much. Versions after versions of popular text editors have come, and yet no editor yet has addressed the difficulty that smaller user groups such as senior citizens and handicapped people have while entering input. Also, predictive editors cease to exist in desktop computers even today. This paper explores the use of a new software for input on desktops, which relies on a dynamic predictive algorithm using n-grams and suffix trees to significantly reduce the effort of typing. This paper also compares the newly developed Smart Text Editor to traditional popular editors.**

Keywords: editor, utility, desktop, autocomplete, autocorrect.

I. INTRODUCTION

Text editors are in abundance in present times. However, most text editors fail to help in a very important aspect i.e. typing. In the era of touch-screens and voice commands, most text editors require typing out entire words on our old keyboards and laptops. While most people are used to this mechanical approach, novice users and senior citizens often have a hard time trying to locate alphabets on their keyboards. People with physical disabilities are unable to use such conventional systems. Also, often people in non-English speaking countries are unfamiliar with exact spellings of English words.

A suggestion mechanism will notably improve the accuracy of documents and will encourage the use of computers by user groups like these. Text prediction and correction mechanisms in an editor can address multiple issues along with the one mentioned earlier. The total number of keystrokes required to type a single word can be significantly reduced thereby improving overall typing speed. Such mechanisms are currently found in integrated development environments such as Eclipse and mobile keyboards such as SwiftKey, but are unavailable in popular desktop applications such as Word and Notepad. This paper proposes a smarter text editing mechanism for mass usage on desktop machines.

II. RELATED WORK

This paper discusses usage of existing algorithms for implementing a software which is not available for desktop machines. Work in the text prediction domain has been going on since a long time. One of the oldest algorithms is HWYE or Hear What You Expect algorithm which works by building a large language model based on frequencies of usage of words and other predictive cues [Lois Boggess, 1987]. Prediction by Partial Matching and its multiple variants are very useful in this application. [Byron Knoll, 2005]. "Text prediction systems: a survey" elaborates on the concepts of Text block size: One character suggestions, One word suggestions and Multiple words suggestion. It introduces a Dictionary structure. It also highlights various prediction methods such as prediction using frequencies, probability tables, Syntactic prediction using grammars, Semantic prediction, etc. [Nestor Garay-Vitoria, Julio Abascal,2005]

B. Issac in his paper titled First Level Text Prediction using Data Mining and Letter Matching in IEEE 802.11 Mobile Devices [3] investigates into a first level text prediction scheme through data mining and letter matching. It uses statistical model to provide suggestions on mobile devices. Adam Pauls and Dan Klein in 2011 introduced several language model implementations that are both highly compact and fast to query: Trie-based Language Models, Implicit Trie, Encoding n-grams. It also demonstrates a Compressed Implementation. Kavita Asnani et al. discuss factors and parameters that define effectiveness of system in their paper *Sentence Completion using Text Prediction System*: Syntactic, Semantic and statistical measures. It introduces generation of n-gram model adaptive to users by applying instance based learning. All these concepts have been referred to or applied in the Smart Text Editor which is discussed in this paper.

978-1-4673-8810-8/16/$31.00 ©2016 IEEE

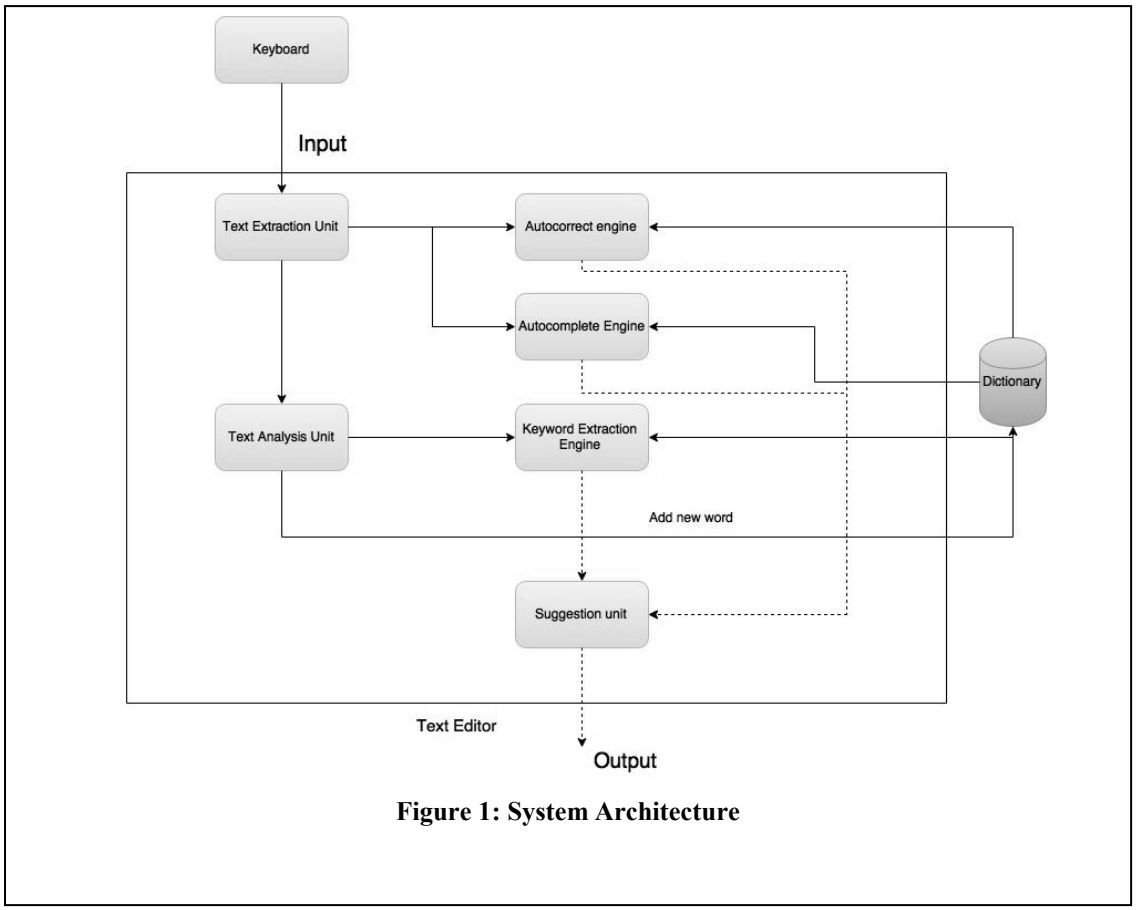

Figure 1: System Architecture

III. Proposed System:

The proposed editor has multiple modules as shown in Figure 1. The editor accepts input from the user, passes it on to the text extraction unit which further branches out in the autocorrect and autocomplete engines which are responsible for correction and prediction respectively. Extracted text is also sent to an analysis unit which extracts keywords from the text. The suggestion unit is responsible for providing real time suggestions ordered by ranks in the user interface. Each module is explained in detail below.

1) Text Extraction Module:
The text extraction module primarily serves as an interface between the user and the system and extracts each individual character typed by the user on his keyboard and passes it on to the engines. In the event of insertion of a complete word (known or unknown) by the user, this unit passes on the word for further analysis to the analysis unit.

2) Autocorrect Engine:
The Autocorrect Engine is responsible for verifying the correctness of a typed word. It builds a set of close words to the mistaken word and searches these words in the dictionary for an exact match.

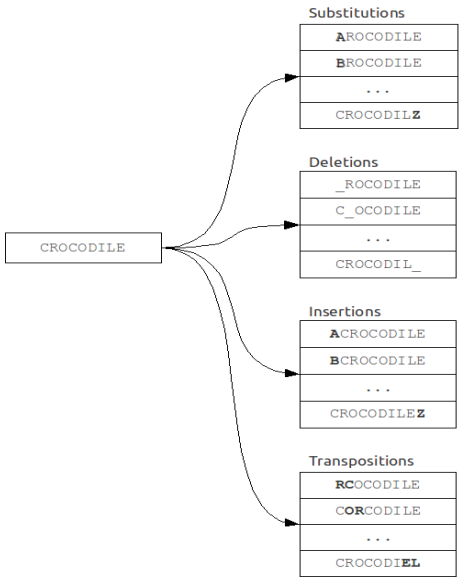

Figure 2: Proximate Word Formation

978-1-4673-8810-8/16/$31.00 ©2016 IEEE

The set of proximate words is determined as shown in Figure 2. All proximate words to the typed word are formed using operations such as substitution, deletion, insertion and transposition. This algorithm doesn't require any dictionary pre-processing or additional memory. Time complexity is dependent on number of errors k and alphabet size |A|.

3) Autocomplete Engine:
This engine is responsible for predicting the remaining part of an incompletely typed word and for guessing which word is likely to continue a given initial text fragment. This module is further divided into two submodules explained as follows:

a) Current word prediction:
Initially, all the words in the dictionary are scanned to build a suffix tree a.k.a. trie.[1] This tree is traversed with respect to frequency to suggest the top words. The structure and traversal of the suffix tree is shown in Figure 3.

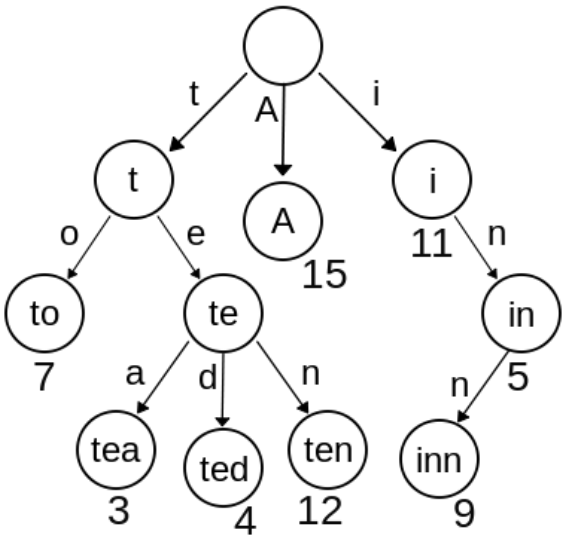

Figure 3: Trie Structure

b) Next word prediction:
Next word predictions are carried out by implementing n-gram language models described by Adam Pauls and Dan Klein in 2011. N-grams are implemented by adding additional nodes to the existing suffix tree and sorting them with respect to their frequencies. N-grams can be implemented on multiple levels such as unigram, bigram, trigram and so on.
N-grams are developed in two phases. In the first phase, the predictions are made on the basis of the statistical model. This statistical model consists of default frequencies of words in the dictionary. In the second phase, these frequencies are modified with each iteration according to the user's style of typing.

4) Dictionary:
Dictionary is the main database used in the system. Initially, a linear list of words is taken. The words are stored in an ascending order alphabetically and corresponding frequencies of the words are also maintained. The dictionary is then developed by using this list to form a tree structure. The tree structure is used in order to make searches faster. Multiple dictionaries are designed in order to make the system faster. Once created, the dictionary can be adapted to the user's vocabulary. The user is given the choice of adding a new word whenever the system encounters an unknown word.

5) Text Analysis and Keyword Extraction Unit:
This unit accepts the entire text entered by a user and extracts keywords from it. These keywords are analyzed and words semantically related to these extracted keywords are then suggested to the user.

6) Suggestion Unit:
This unit compiles suggestions from all individual engines and displays top suggestions by ranking them.

IV. RESULTS

Features	Notepad	Notepad++	MS Word	Smart Text Editor
Autocorrect	No	No	Yes	Yes
Autocomplete	No	No	Partial	Yes
Prediction	No	No	No	Yes

Table 1: Comparative Feature Analysis

The editor was compared with various popular text editing applications and the results are tabulated in Table 1.
The speed of typing was also compared. A 200-word essay took an average of 5 minutes to type on MS Word by a set of 10 users. The same essay took just above 3 minutes to type on the Smart Text Editor by the same set of users with at least 50% reduction in the total number of keystrokes required. These results are demonstrated in Figures 4 and 5.

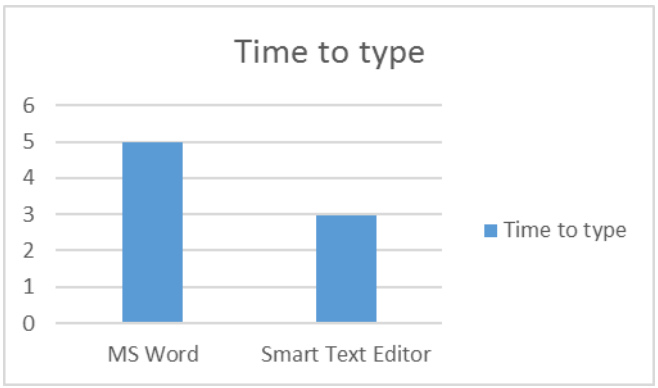

Figure 4: Typing Time Comparison

978-1-4673-8810-8/16/$31.00 ©2016 IEEE

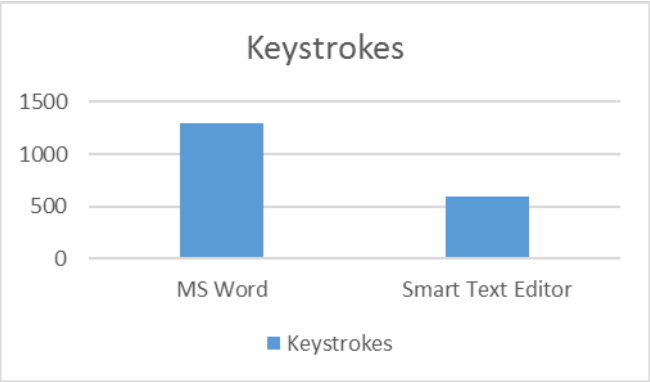

Figure 5: Keystroke Comparison

V. CONCLUSION AND FUTURE WORK

The Smart Text Editor can become a very handy tool for text input and as shown earlier, can perform significantly better than most widespread editors in some aspects. Adding facilities like better context based suggestions will improve the usability of the editor.

References

[1] A. Pauls and D. Klein, *Faster and smaller n-gram language models*. 2011.

[2] N. Garay-Vitoria and J. Abascal, "Text prediction systems: a survey", Universal Access in the Information Society, vol. 4, no. 3, pp. 188-203, 2005.

[3] T. Sobh and B. Issac, Innovations and advances in computer sciences and engineering. Dordrecht: Springer, 2010, pp. 319-324

[4] H. Komatsu, S. Takabayashi and T. Masui, "Corpus-based predictive text input", Proceedings of the 2005 International Conference on Active Media Technology, 2005. (AMT 2005)., pp. 75-80, 2005.

[5] S. Satapathy, K. Asnani, D. Vaz, T. PrabhuDesai, S. Borgikar, M. Bisht, S. Bhosale and N. Balaji, Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA). Cham [u.a.]: Springer, 2015, pp. 397-404.

[6] L. Boggess, Two simple predition algorithms to facilitate text production. Austin, TX: MSU Dept. of Computer Science, 1988, pp. 33-40.

[7] B. Knoll, Text Prediction and Classification Using String Matching, 2009.

[8] N. Smetanin, "Fuzzy string search", Nikita's Blog, 2011

978-1-4673-8810-8/16/$31.00 ©2016 IEEE