A Feature Based Data Structure for Efficient Index

Yang LIU\textsuperscript{1,a\textsuperscript{*}}, Chao FENG\textsuperscript{1,b}, Carlos-cheuk-hang CHIU\textsuperscript{1,c}, Kang-heng WU\textsuperscript{1,d} and Zhi-bin LEI\textsuperscript{1,e}

\textsuperscript{1}Hong Kong Applied Science and Technology Research Institute (ASTRI), Hong Kong
yangliu@astri.org, charlesfeng@astri.org, carloschiu@astri.org, khwu@astri.org, lei@astri.org

*Corresponding author

Keywords: Feature Based, Data Structure, Index.

Abstract. Data collected from our daily lives by mobile electronic devices always contains errors. This paper proposed a feature based data structure to organize the dictionary for correcting the errors efficiently. The key points in the method includes explicit splitting the dictionary by high frequency used elements, implicit splitting further by clustering, and a hierarchical match procedure. The numerical results indicated the proposed methods can index the input effectively while reducing the consumption in both time and RAM.

Introduction

The rapid increase of mobile electronic devices allows the instantaneous collection of tremendous amount of digital data in our daily lives. Much of these digital data is meant to be processed and eventually be displayed in graphical and text formats, such as digital newsfeeds, instant image captures, and text messages. The processing that converts these raw digital data collected or captured in their binary and/or machine-readable formats into human-readable text may involve certain data decoding steps, other special conversion steps such as optical character recognition (OCR), and/or language translation. However, these data processing procedures are not error free, and often result in erroneous characters and words, or even illegible text. Thus, an additional step of language error detection and corrections, such as spell checking and auto-correction, is needed.

Traditionally, if the scenario was already known, a dictionary including all the possible results can help to correct the errors. However, in a mobile computing device, such as a smartphone, both CPU and memory space are much more limited in comparison to regular computers for the dictionary usage. Dictionary encoding and dictionary compress presented in [1][2][3][4] may reduce the memory space, but the calculation complexity prevents the techniques being used in mobile devices. On the other hand, the user experience of spell checking and auto-correction demands real-time performance and high level of accuracy. To speed up the indexing, a tree-like structure could be used to organize the dictionary as in [5][6], but the possible error in the inputs may sometimes make the listed tree-like structure fail to return a valid answer.

Therefore, there is a need for a better method for organizing and processing input text generated from raw data and dictionaries used to spell check and auto-correct the input text that has more efficient use of computing resources.

The following of this paper is divided into three parts. Section 2 illustrated the proposed structure method in details. Section 3 used numerical results to show the efficiency and accuracy of the proposal. Finally, section 4 summarized the work.

Data Structure Method

Overview. Referring to FIG. 1, which shows a flow diagram of the method for organizing and processing feature based data structures used in linguistic spell checking and auto-correction in accordance to one embodiment of the present invention. The method comprises following steps:
Step 1, splitting an original dictionary into sub-dictionaries using a feature based explicit split or implicit split method;

Step 2, receiving an input human-readable text that contains one or more errors;

Step 3, determining a sub-dictionary selection feature or selection criteria from the input text;

Step 4, selecting the sub-dictionary based on the determined sub-dictionary selection feature or selection criteria;

Step 5, executing a first matching of the one or more characters, words, or phrases in proximity of the errors against the characters, words, and phrases in the selected sub-dictionary; if a unique match is found, the resulting match is returned as an output; otherwise if multiple matching candidates are found;

Step 6, executing a second matching of the one or more characters, words, or phrases in proximity of the errors in the selected sub-dictionary with a raised threshold of degree of similarity; and repeat step 6 until a unique match is found.

**Key Points.** The key points need to be illustrated include dictionary split (explicit split, implicit split) and match procedure (first round match and second round match).

Explicit split. Referring to Fig. 2 as an example to illustrate the feature based explicit splitting of an original dictionary into two sub-dictionaries based on the common feature of high frequency words. In this example, the original dictionary contains the words and phrases {steak and kidney pudding, steak pie, steak and oyster pie, pork pie}. Various features can be used to split the dictionary. We prefer to use high frequency words for text samples. In the given examples, the high frequency words selected are “steak” and “pie”. After the explicit split, a first sub-dictionary contains “steak” as the index: {steak and kidney pudding, steak pie, steak and oyster pie} and a second sub-dictionary contains “pie” as the index: {steak pie, steak and oyster pie, pork pie}. It is noticeable that although there are overlapping between the two sub-dictionaries, each sub-dictionary is shorter than the original dictionary. Therefore, our method reduces the dictionary effectively while remain some space for the case when the input contains errors.
Implicit split. If the sub-dictionary after the explicit split is still too large to be practically used, a further optional implicit split can be used. The principle of implicit split is to use code difference to show the similarity between various strings. First, the vector dimension of the word space is determined by statistics to the dictionary. For example, if most of the recognized text in the dictionary is of length 4 in characters or words, the vector dimension will be 4. Secondly, the recognized text is transformed into vectors. For example, the Unicode value of each character/word in the recognized text words can be used to generate the vector. Finally, the dictionary from the explicit can be organized into $n$ clusters ($n > 1$). Following the example above, if K-means is used for clustering and $n = 2$, we can have the Euclidean distance of input string to the center vector of each cluster $D_1$ and $D_2$ in the vector space determined in the first step. If $D_1 < D_2$, the first cluster of implicit split results should be used.

First round match. This is the procedure to find a candidate group from the selected dictionary(cluster) with the highest match score after comparison to the input. The match score adds one when a character/word in the input is the same as that in the dictionary line in corresponding specified locations. The specified locations mean the location of the input word/character in the input and the adjacent ones. If the matched result selected based on a threshold is not unique, start the second round check. The given threshold can be determined by statistics from the dictionary and the scenario. In general, a reasonable low threshold will be preferred as a trade-off between accuracy and efficiency in the second round check. For example, if the selected dictionary is {steak pie, steak and oyster pie, pork pie} and we assume the score is based on word, the score of first round match for input “steak pie &” will be {2,2,1}. Setting the threshold at 2, the output candidates after first round match are {steak pie, steak and oyster pie}.

Second round match. Determine the final output by comparing the distance of the original input to each candidate selected in the first round. The distance is defined the sum of character/word Unicode difference absolute value (ABS) on the corresponding locations. In the example given above, the distance of input “steak pie &” to the candidates {steak pie, steak and oyster pie} can be the word Unicode difference ABS on the corresponding locations. That is, the distance calculation will be {ABS(Unicode value of symbol ‘&’), ABS (Unicode value sum of string ‘and’ - Unicode value sum of string ‘&’ + Unicode value sum of string ‘oyster pie’)}. Easy to see, {steak pie} with the smaller distance (ABS(Unicode value of symbol ‘&’)) will be the final match result for original input “steak pie &”. In another sentence, the input “steak pie &” with error “&” is corrected to “steak pie".
Numerical Results

Four groups of text samples after OCR procedure are used as inputs in our experiments. Each group has 100 samples with the same OCR accuracy level. The dictionary before splitting is over 3500 samples. The sample length varies from 3 to 10. For the explicit split, we use two high frequency words. For the implicit split, we use two clusters with K-means. The experiment is done on android 6.0.1 platform. We monitor the time and RAM consumption of the whole procedure when the algorithm runs.

Table 1 shows the average resource consumption results. Due to the split of original dictionary, the benefit on calculation time and RAM consumption is evident. The average calculation reduces 80% and the average RAM consumption reduces 50%.

Table 1. Resource Consumption Results.

<table>
<thead>
<tr>
<th>Metric</th>
<th>with proposal</th>
<th>w/o proposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average calculation time [ms]</td>
<td>18</td>
<td>102</td>
</tr>
<tr>
<td>Average RAM consumption [Mbytes]</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2 compares the correction rate results of our method to a simple match procedure throughout the whole dictionary. Our method has a higher (generally around 10%) correction rate than that of simple match procedure in three test groups. The reason of the improvement is that we use two round match to determine the final results. The Unicode difference comparison in the second round is more accurate than a simple match. In the high OCR accuracy group, our algorithm error correction rate is similar as that in simple match procedure with a small loss (3%).

Table 2. Correction Rate Results.

<table>
<thead>
<tr>
<th>Test Group</th>
<th>OCR accuracy</th>
<th>with proposal</th>
<th>w/o proposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&gt;80%</td>
<td>93%</td>
<td>96%</td>
</tr>
<tr>
<td>2</td>
<td>60%-80%</td>
<td>70%</td>
<td>61%</td>
</tr>
<tr>
<td>3</td>
<td>40%-60%</td>
<td>59%</td>
<td>49%</td>
</tr>
<tr>
<td>4</td>
<td>&lt;40%</td>
<td>22%</td>
<td>10%</td>
</tr>
</tbody>
</table>

Summary

This paper concludes our work in a feature based data structure for efficient index in text correction scenario. To the help of dictionary split and two rounds match, our method reduces the time and RAM consumption evidently on mobile devices while maintaining good error correction accuracy. The future work includes how to adapt the method in other scenarios such as classifications and how to deal with other type of data such as images.

References


