Report of English Input Predictor

Lihang Liu, 5120309708
backchord@sjtu.edu.cn

CS382 Natural Language Processing, Shanghai Jiao Tong University, China

***Abstract*—This paper addresses the problem of English Input Predictor on incomplete sentence with even last word incomplete. We first introduce method of traditional n-gram. Based on candidate set retrived by n-gram, we proposed an edit-distance based score function relevant to user behaviors. Secondly, a correction history map is learned. We found the behaviors of users are similar in some extent and the simple method got a better result. Finally, character based recurrent neural network is utilized to test potential of RNN in this task. Experimental results of these three method demonstrate the effectiveness of our approach.**

I. Introduction

English Input Predictor is a hard question due to different user behaviors. At the same time, in real world dataset, target words may be non-word as well as varies in upper and lower case from different users.

The most apparent method is firstly to find a possible candidate set, and then use edit distance to get the nearest word as the result. There are tricks in this method, including ways to find the candidate set, choice between several candidate sets, and the edit distance cost function.

Secondly, a tricky method to deal with real world correction is correction history learning. This is widely used in many input systems. Google Search gives the most frequent candidate phases based on statistics of other users when you type in the incomplete words. Experiment demonstrate the effectiveness of this method in real world dataset compared to the first method. However, this method has its limitations.

Finally, we utilized character based recurrent neural network to predict the target words. Recurrent neural network has the advantage of context-dependent prediction. Words based recurrent neural network ought to be more effective than character based. However, in this case, the incomplete words and even non-words make the words based rnn impractical.

The remainder of this report is organized as follows. Section II briefly introduce the dataset. Section III presents the details of our approach. Experimental results are shown in Section IV. Section V concludes the paper.

II. Dataset

Dataset on this report is from Cooltek, a company of input method applications. The first resource is a corpus with six million sentences or paragraphs. The second resource is a task set with incomplete sentence ends with incomplete words, the target words is also given. An example will be:

should I bring snacks or anythi anything

Given the incomplete sentence *should I bring snacks or anythi* and we can speculate the user may complete the last word with *ng* resulting the target word *anything*.

Also, this real world dataset contains many noises. Identical context may result in different target words; misuse in capitalization; full of cyberwords. Cases like these lead to obstacle in prediction. Some examples are shown below:

hi im im
And im I'm
hahahhaha OMG thanks soooooo soooooo
gOT sELECTED fROM tHEIR lottery but BUT

III. The Approach

A. N-Gram and Edit Distance

N-Gram. We utilized 1-3-grams. First of all, we discard all non character symbol except for ".", "?" and "!". For unigram, we especially introduce a external dictionary from git. Only words with frequency larger than 64 in our corpus or otherwise appeared in the external dictionary are included into the unigram set (or called the vocabulary). The policy results in both smaller vocabulary and less false negative. For bigram and trigram, we filtered the grams do not match the first step unigram set and those frequency less than 2. The use of 1-3-grams are not mixed. We looks for prediction words orderly from trigram, bigram to unigram. Every added step gains bigger accuracy.

Edit Distance. The choice of edit distance matters in our case. In computer science, edit distance is a way of quantifying how dissimilar two strings are to one another by counting the minimum number of operations required to transform one string into the other. Traditional allowed operations includes *Replace*, *Insert*, *Delete*. Based on our task, we introduce 3 more fine operations, including *Append*, *KBReplace*, *Switch*.

- **Append**. It's common for users to complete their spelling with several more characters. Interpret this kind of behaviors as *Insert* is misleading for cost function. From "importa" to "import**ant**", we need two *Append* operations.
- **KBReplace**. Keyboard distance is also a important factor in typing. The distance of two adjoin character in keyboard is loser than others, such as the distance of "a" to "s" is undoubtly smaller that "a" to "m" in our keyboard. From "import**am**t" to "import**an**t", we need one *KBReplace*.
- **Switch**. Mistype the order of two character is also common when the typing speed up, like the case of "import**an**t" versus "import**na**t".

The resulting operation set is Φ = {*Replace*, *Insert*, *Delete*, *Append*, *KBReplace*, *Switch*}. In implementation, the calculate of operations are executed by dynamic programming. Assume the nodes of search path is $P(S1, S2)$. Thus, the edit distance cost function of two strings $S1$ and $S2$ can be formulated as:

$$ED(S1, S2) = \Sigma_{\phi}^{\Phi}(w_{\phi} * \Sigma_{p}^{P(S1,S2)} 1 * \{p = \phi\})$$

where the weights w_{ϕ} of each operation ϕ need further evaluation.

Prediction Steps. After n-grams and edit distance cost function are defined, we can do prediction now.

1) Set N = 3 and threshold θ.
2) Retrieve candidate set based on n-gram and find the nearest candidate word S_i to incomplete word S_{input} based on edit distance function.
3) If the distance between S_i and S_{input} is smaller than the threshold θ, output S_i. Else, set $n = n - 1$ and go to step 2.
4) If n is equal to 1, output S_i.

B. Correction History Learning

The assumption in this method is different users may share similar even the same behavior in words completion, thus one correction behavior is not sole in the dataset. Thus, we can learn a user correction history mapping, from S_{input} to S_i. In test phase, we look up S_{input} among keys of the user correction history mapping and return the most frequent mapped word as the prediction word. If not found, we can use the method of III-A, "N-Gram and Edit Distance" to find prediction word.

An enhanced version of this method is to extend the mapping of $last\ word \Rightarrow target\ word$, to $last\ n\ words \Rightarrow target\ word$.

However, this method has its limitations. It largely depends on the size of training set and the similarity between training set and testing set. As the size of training set decreases, the prediction accuracy drops largely. The application of this method is also confined to specific tasks.

C. Character based Recurrent Neural Network

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. In natural language learning, a word based RNN is more effective than character based RNN. However, dealing with incomplete words, cyber words and non-words, it's hard to embed all these words. Thus we choose the character based RNN.

Concretely, we will encode each character into a vector using 1-of-k encoding (i.e. all zero except for a single one at the index of the character in the vocabulary), and feed them into the RNN one at a time. After all the characters are fed, we output the predicted characters and utilize cross entropy loss as the training loss function. At test time, we feed all the input characters into the RNN and when the first predicted character distribution is given, we retrieve the most likely character as the output. Repeat the process until the halt symbol is got. An illustration of the process is given in the graph 1. The length of the RNN Model is dynamic and the maximum length in our experiment is 100.

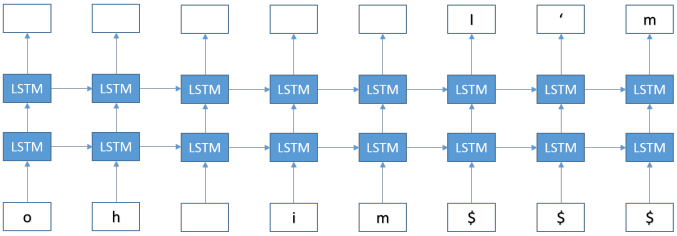

Figure 1. RNN Model. Given the input *oh im*, the target word will be *I'M*.

IV. Experimental Results

In this section, we show experiments on our approach. The former two method is implemented by python and Java. While the last method of RNN is implemented by Caffe.

In method one (NG+ED) (section III-A N-Gram and Edit Distance), we learn the n-gram by corpus file. And we set $\omega_{Replace} = \omega_{Insert} = \omega_{Delete} = 1$, $\omega_{Append} = 0.1$, $\omega_{KBReplace} = 0.5$ and $\omega_{Switch} = 0.5$. In Prediction Steps of section III-A, we set the threshold $\theta = 1$.

In method two (section III-B Correction History Learning), we separate the train.txt into train set and test set by ratio 9:1. Correction history mapping only (CHM), as well as combining correction history mapping and method one (CHM+NG+ED), are tested on the dataset.

In method three (CRNN) (section III-C Character based Recurrent Neural Network), we also separate the train.txt into train set and test set by ratio 9:1.

The results are shown in the table. Note that these experiment are executed disregarding the difference of upper case or lower case. Considering the difference of upper case and lower case brings about 7% drop of accuracy.

Method	Accuracy(%)
NG+ED	77.7
CHM	79.1
CHM+NG+ED	**83.2**
CRNN	82.8

V. Conclusion

In this paper, we propose an three approaches to address the problem of English input prediction. We combined the method of traditional n-gram and edit distance to give prediction. Also, a tricky and effective user correction history method was proposed. Finally, we utilized character based RNN on our dataset to do prediction.