Noisy FAQ Retrieval Using Longest Common Subsequence Problem

Pooja Porwal porwal11293@gmail.com

Aman Jain aman8sanky@gmail.com

INDIAN SCHOOL OF MINES

DHANBAD,JHARKHAND-826004

ABSTRACT

The aim of this task is to find the best 5 matches for a SMS based Query from a corpora of Frequently asked Questions (FAQs). Both the SMS queries and the FAQs are in the same language i.e English. Our system uses a single retrieval engine named INDRI tool .In our approach, the SMS queries are first transformed into normalized or corrected form with the help of Longest Common Subsequence method and Levenshtein Distance algorithm .The normalized list of SMS queries are submitted to a retrieval engine to obtain a ranked list of FAQ queries.

Introduction

The growing use of information technologies such as mobile devices has had a major social and technological impact such as the growing use of Short Message Services (SMS), a communication system broadly used by cellular phone users.
SMS is a very popular short message-based communication service among mobile phone users. However, SMS is also synonym of the short message itself which can contain up to 160 characters. The length limitation of an SMS has led to create a sort of “sub-language” which includes a vocabulary of words, similar to that of the original natural language, but that regularly omit grammatical forms, punctuation marks and vowels.

This leads to various lexicographical errors which create problem in analysis of query and providing corresponding correct results.

The aim of this task is to find the best 5 matches for a SMS based Query from corpora of Frequently asked Questions (FAQs). We participated in the monolingual retrieval task where both the SMS queries and the FAQs are in English. We use a single Indri retrieval engine into our system. The SMS queries are first transformed into normalized or corrected form with the help of Longest Common Subsequence method and Levenshtein Distance algorithm.The normalized list of SMS queries are submitted to a retrieval engine to obtain a ranked list of FAQ queries. As a first-timer, our performance is moderate but promising.

SMS based FAQ Retrieval Scheme

The SMS based FAQ Retrieval scheme works in two sections. First section includes correction of incorrect SMS query into corrected query by using Longest Common Subsequence algorithm and Levenshtein Distance algorithm.The combination of maximum lcs value and minimum levenshtein distance value will give us better matches for a given incorrect word.

The second section involves passing the corrected form of query into INDRI tool and getting corresponding 5 best matches for a given SMS query.

Experiment Description

The first task is to correct the SMS query using Longest common subsequence method and Levenshtein distance algorithm.

The **longest common subsequence** (**LCS**) **problem** is to find the longest common to all sequences in a set of sequences (often just two). It is different from substring because they need not be consecutive terms of the original sequence. It works on the principle of Dynamic programming. The LCS function is stated as follows:
Let two sequences be defined as follows: $X = (x_1, x_2...x_m)$ and $Y = (y_1, y_2...y_n)$. The prefixes of X are $X_{1, 2,...m}$; the prefixes of Y are $Y_{1, 2,...n}$. Let $LCS(X_i, Y_j)$ represent the set of longest common subsequence of prefixes X_i and Y_j. This set of sequences is given by the following.

$$LCS\left(X_i, Y_j\right) = \begin{cases} \emptyset & \text{if } \ i = 0 \text{ or } j = 0 \\ LCS\left(X_{i-1}, Y_{j-1}\right) + 1 & \text{if } x_i = y_j \\ \text{longest}\left(LCS\left(X_i, Y_{j-1}\right), LCS\left(X_{i-1}, Y_j\right)\right) & \text{if } x_i \neq y_j \end{cases}$$

The Longest Common Subsequence algorithm is as follows:

The function below takes as input sequences `X[1..m]` and `Y[1..n]` computes the LCS between `X[1..i]` and `Y[1..j]` for all $1 \leq i \leq m$ and $1 \leq j \leq n$, and stores it in `C[i,j]`.`C[m,n]` will contain the length of the LCS of `X` and `Y`.

LCS-LENGTH*(X, Y)*
1 *m* ← *length*[*X*]

2 *n* ← *length*[*Y*]
3 **for** *i* ← 1 **to** *m*
4 **do** *c*[*i,* 0] ← 0
5 **for** *j* ← 0 **to** *n*
6 **do** *c*[0*, j*] ← 0
7 **for** *i* ← 1 **to** *m*
8 **do for** *j* ← 1 **to** *n*
9 **do if** *xi* = *yj*
10 **then** *c*[*i, j*] ← *c*[*i* − 1*, j* − 1] + 1
11 *b*[*i, j*] ← "&"
12 **else if** *c*[*i* − 1*, j*] ≥ *c*[*i, j* − 1]
13 **then** *c*[*i, j*] ← *c*[*i* − 1*, j*]
14 *b*[*i, j*] ← "↑"
15 **else** *c*[*i, j*] ← *c*[*i, j* − 1]
16 *b*[*i, j*] ← "←"
17 **return** *c* and *b*

The value of c[m][n] gives subsequence of maximum length.This gives us no. of words having maximum matches with the given incorrect word.

The second method is Edit distance, also known as **Levenshtein distance** or evolutionary distance is a
concept from information retrieval and it describes the number of edits (insertions, deletions and substitutions) that have to be made in order to change one string to another. It is the most common measure to expose the dissimilarity between two strings .
The edit distance ed(x, y) between strings x=x1 ... xm and y=y1 ... yn, where x, y

$\in \Sigma*$ is the minimum cost of a sequence of editing steps required to convert x into y.

Mathematically, the Levenshtein distance between two strings is $lev_{a,b}(|a|,|b|)$ where

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and 1 otherwise.

```
// len_s and len_t are the characters in string s and t respectively

Int levenshteinDistance(string s,int len_s,string t,int len_t)

{

/*test for degenerate cases of empty strings*/

If(len_s==0) return len_t;

If(len_t==0) return len_s;

/*test if last characters of strings match*/

If(s[len_s-1]==t[len_t-1]) cost=0;

Else                cost=1;

/*return minimum of delete char from s,delete char from t,delete char from both */

Return minimum(levenshteinDistance(s,len_s-1,t,len_t) +1,

                levenshteinDistance(s,len_s,t,len_t-1) +!,

                levenshteinDistance(s,len_s-1,t,len_t-1) +cost);

}
```

Thus,the combination of both the algorithm gives matches for a given word with maximum lcs value(showing maximum matching) and minimum distance value(showing minimum no. of changes to be made to convert incorrect word to correct word).

The corrected words are joined to form a sentence.And finally we employed these sentences into INDRI tool which provides us 5 best matches for a given SMS query.

Observation and Results

Results for our run submission are as follows:

Text result

No. of In-domain queries	200
No. of out of domain queries	99
In domain correct	77/200 (0.385)
Out of domain correct	0/99 (0.00)
Total Score	0.257
Mean Reciprocal rank(MRR)	0.468

Speech result

No. of In-domain queries	192
No. of out of domain queries	49
In domain correct	1/192 (0.005)
Out of domain correct	0/49 (0.00)
Total Score	0.0041
Mean Reciprocal rank(MRR)	0.0071

Overall result

No. of In-domain queries	392
No. of out of domain queries	148
In domain correct	78/392 (0.198)
Out of domain correct	0/148 (0.00)
Total Score	0.144
Mean Reciprocal rank(MRR)	0.242

Conclusion and Future Work

An SMS query contains certain noisy terms, most of them are due to the limitation of characters allowed in an SMS and typographical errors by the

sender. Therefore, retrieving the relevant queries from FAQ corresponding to the SMS query is a very challenging task.

The proposed selection of the relevant queries is based on the matching by LCS and Levenshtein distance method of the FAQ queries with an SMS query. The method is implemented in monolingual English task in FIRE 2013 SMS-based FAQ Retrieval and it has performed significantly to retrieve the relevant queries from FAQ corresponding to an SMS query.

The results obtained for out of domain queries are unexpected and needs to be worked upon more. Distinguishing in-domain and out-of-domain queries is yet to be done. However currently the system ends up predicting certain matches for these queries instead of identifying them as out of domain.

In future,we would work on increasing the efficiency of matching which would give better MRR.

Acknowledgement

We acknowledge our project guide, Prof. Sukomal Pal for his support and advices for the proper direction in completion of the project and help in solving the problems we came through this project.

References

- Working notes of FIRE 2012
- http://en.wikipedia.org/wiki/UTF_8
- http://www.lemurproject.org/indri.php
- http://www.google.com/#hl=en&safe=off&tbo=d&site=&source=hp&q=SMS+query+retreival&oq=SMS+query+retreival&gs_l=hp
- http://sourceforge.net/projects/lemur/
- www.isical.ac.in/~fire
- Dynamic programming,Cormen Introduction to algorithms-2nd edition
- en.wikipedia.org/wiki/levenshtein_distance