**CHAPTER 1. INTRODUCTION**

The number of mobile users is growing at an amazing rate. In India alone a few million subscribers are added each month with the total subscriber base now crossing 370 million. The anytime anywhere access provided by mobile networks and portability of handsets coupled with the strong human urge to quickly find answers has fueled the growth of information based services on mobile devices. This FAQ retrieval system is designed to find a match from the given set of Frequently Asked Questions for a query written in SMS language. The problem with questions asked in SMS language is that the SMS text has a lot of noise present in it which might be due to lack of a proper keypad in low cost mobile phones, less screen space and also cause of convenience to the users. Understanding user questions in natural languages requires Natural Language Processing (NLP). This QA system can provide a convenient and effective way of giving answers to such questions.

The nature of texting language, which often as a rule rather than exception, has misspellings, non-standard abbreviations, transliterations, phonetic substitu­tions and omissions, makes it difficult to build automated question answering sys­tems around SMS technology. This is true even for questions whose answers are well documented like a FAQ database. Unlike other automatic question answer­ing systems that focus on generating or searching answers, in a FAQ database the question and answers are already provided by an expert. The task is then to iden­tify the best matching quest ion-answer pair for a given query. The approach we have adopted in this project is an automated FAQ (Frequently Asked Question) answering system that gives the best matching questions, from a pre-stored set of questions and answers that have been provided, to questions asked in ordinary SMS text. This is achieved using sequence matching techniques, disremvoweling, etc.

* 1. **Problem Statement:**

In this task, we have a corpus of frequently asked questions and answers from var­ious domains that have been provided. The corpora of questions in the database are represented by Q. The query is in SMS language which may or may not contain noise. The goal of the task is to find a question Q\* from the corpora of FAQ's Q, that is the best possible match for the SMS query S. In order to achieve this, we have made use of techniques like disrvoweling, removal of stop words, Longest Common Subsequence (LCS), etc.. We remove all the stop words from the SMS query. In disvoweling, we remove all the vowels from the user's query and from the corpus of questions and search for keywords of the disemvoweled query in the disemvoweled set of questions. The question that has maximum number of matching keywords gets the highest score (keyword score) Also, we find the best possible match for each word in the query from all the words occur­ring in all the questions in Q. For this we use techniques like Longest Common Subsequence. The question which has words that are best possible matches for the words in the query get the highest score (similarity score).

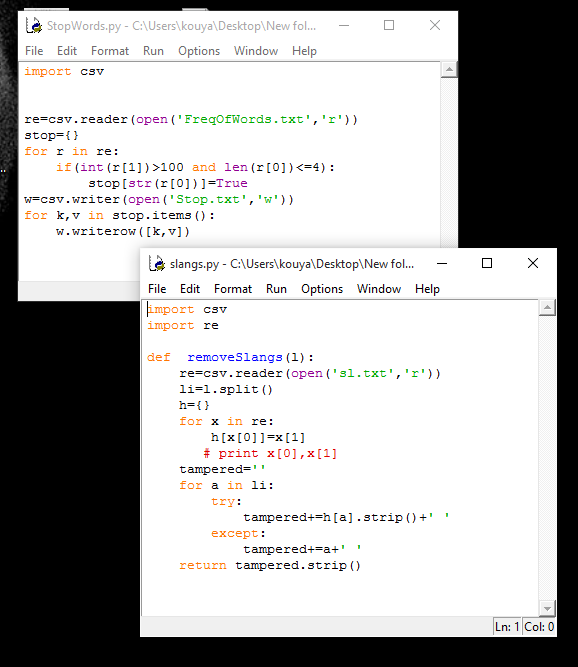
Therefore, we have two parameters for calculating the score of a question, keyword score and similarity score. The methods for calculating the keyword score, like disvoweling, are based on the general observations made about the language and slangs used by people while typing SMS text. On the other hand, the similarity score is calculated using dynamic programming techniques for string comparison and pattern matching algorithms, like Longest Common Subsequence and Gestalt Pattern Matching.

We combine the two scores obtained to get the total score for the questions and the question having the maximum total score is returned as the best possible match Q\* for the SMS query S.

**CHAPTER 2. SYSTEM IMPLEMENTATION**

* 1. **Preprocessing:**

In this section we describe the prior work required before we start finding the match for the SMS query. We obtained set of words W hat contains all the words occurring in all the questions in Q. These words have been stored in a hash table in which the keys are characters from a-z and numbers 0-9. Thus the words get stored in alphabetical order. For Example, a key ‘I’ contains all the words in the set Q that start with the letter ‘I’, like ‘insurance’, ‘improve’, and so on. This is done so that we can find the lists of matching words for each of the words in the SMS query S more efficiently. Thos is described in detail later. A list of stop words is also prepared and disemvoweled. Digits occurring in SMS token (e.g. ‘w8’ , ‘4get’ ) are replaced by string based on a manually designed digit-to-string mapping (‘8’ ‘eight’).

 **Fig 2.1-Code of Slangs And Stop words.**

**2.2 Calculation of Weight (W)**

For each token of the SMS query (not disemvoweled), we calculate its similarity with every word ‘w’ in the corpus W. The weight of a word is given by the equation:

Weight (w,s)= LCSR(w. s) \* SM Ratio(w,s) \* I DF(w)

LevDistance (w,s)

Where,

LCSR (w, s) - Longest Common Subsequence Ratio of the SMS query token s and the word w in W.

SMRatio (w, s) - Similarity ratio using Ratcliff Obershelp algorithm.

LevDistance (w, s) - Levenshtein Distance between disemvoweled w and s.

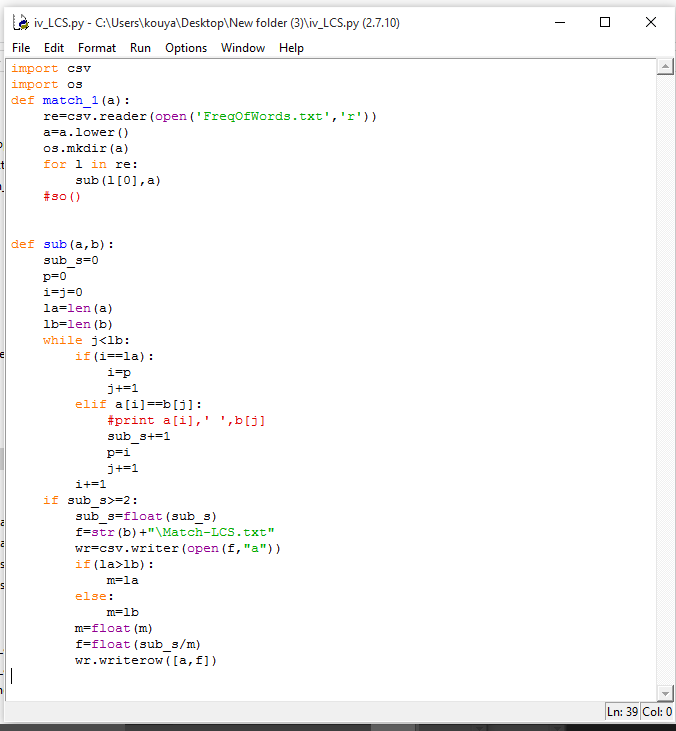
IDF (w) - Inverse Document Frequency of w.

Longest Common Subsequence Ratio (LCSR). The longest common sub­sequence (LCS) problem is to find the longest subsequence common to all se­quences in a set of sequences (often just two). A subsequence is a sequence that can be derived from another sequence by deleting some elements without chang­ing the order of the remaining elements. The Longest Common Subsequence Ratio between a word w in W and a token s in SMS query S is the ratio of their LCS to the maximum of the lengths the two.

LCS (w, s)  
 maxlength (w, s)

LCSR {w. s) =

Example: LCSR of ‘thru’ and ‘through’ is 3/7=0.4285.



**Fig 2.2-Code of LCSR.**

For a word to have high weight, its LCSR should be high.

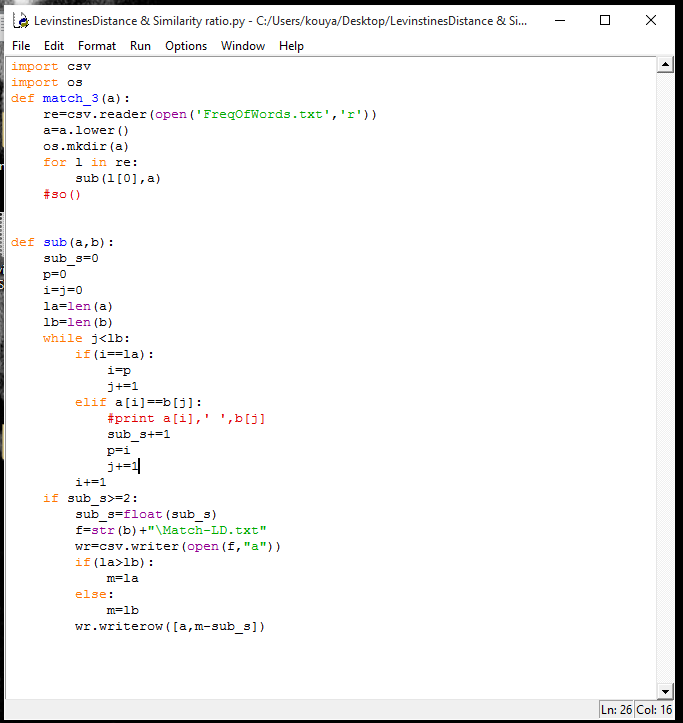
Similarity Ratio. The similarity ratio (SMRatio) of two words is calculated using Ratcliff/Obershelp algorithm for ‘gestalt pattern matching’. Gestalt is a word that describes how people can recognize a pattern as a functional unit that has properties not derivable by summation of its parts. For example, a person can recognize a picture in a connect-the-dots puzzle before finishing of this process of filling in the missing parts by comparing what is known to previous observations is called gestalt. The Ratcliff/Obershelp pattern-matching algorithm uses this same process to decide how similar two strings are.

**Table 1. An example of calculating Similarity Ratio between two strings using the Ratcliff Obershelp Algorithm**



The algorithm works by examining two strings passed to it and locating the largest group of characters in common. The algorithm uses this group of charac­ters as an anchor between the two strings. The algorithm then places any group of characters found to the left or the right of this anchor on a stack for further examination. This procedure is repeated for all substrings on the stack until there is nothing left to examine. The algorithm calculates the score returned as twice the number of characters found in common divided by the total number of characters in the two strings

For example, suppose you want to compare the similarity between the word ‘Pennsylvania’ and a mangled spelling as ‘Pencilvaneya\*. The largest common group of characters that the algorithm would find is ‘Ivan’. The two sub-groups remaining to the left are ’Pennsy’ and ‘Penci’, and to the right are ‘ia’ and ‘eya’. The algorithm places both of these string sections on the stack to be examined and advances the current score to eight, two times the number of characters found in common. The substrings ‘ia’ and ‘eya’ are next to come off of the stack and are then examined. The algorithm finds one character in common: a. the score is advanced to ten. The substrings to the left—‘I’ and ‘ey’—are placed on the stack, but then are immediately removed and determined to contain no character in common. Next, the algorithm pulls ‘Pennsy’ and ‘Penci’ off of the stack. The largest common substring found is ‘Pen.’ The algorithm advances the score by Pi so that, it is now If. There is nothing to the left of ‘Pen’, but. to the right are the substrings ;nsy’ and ‘ci’, which are pushed onto the stack. When the algorithm pulls off ‘nsy’ and ‘ci’ next, it finds no characters in common. The stack is now empty and the algorithm ready to return the similarity value found. There was a score of 16 out of a total of 24.



**Fig 2.3-Code of Similarity ratio and Levenshtein Distance.**



Levenshtein Distance. Levenshtein distance is a ‘distance’ (string metric) between two strings, i.e., finite sequence of symbols, given by counting the mini­mum number of operations needed to transform one string into the other, where an operation is defined as an insertion, deletion, or substitution of a single char­acter, or a transposition of two characters. To find the Levenshtein Distance between w and s, we first disemvowel them and then calculate the Levenshtein distance.

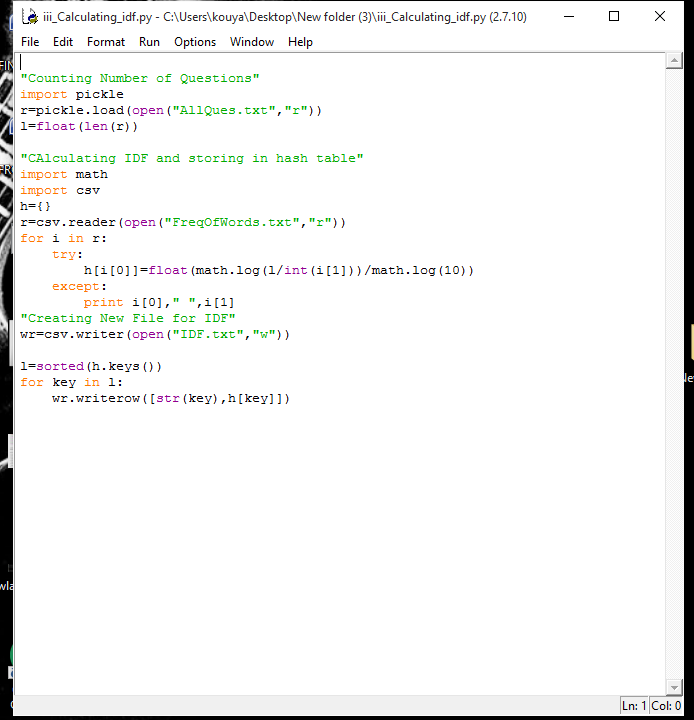
For example, the Levenshtein distance between “kitten" and “sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

Kitten **—>** sitten (substitution of V for "k'). Sitten —» sittin (substitution of T for ’e’). Sittin —» sitting (insertion of ‘g’ at the end).

Words which have a low Levenshtein Distance are the ores which require less number of operations to get transformed into the SMS token. Thus for the weight of a word to be high, its Levenshtein Distance must be low.

Inverse Document Frequency (IDF). If f number of documents in corpus Q contain a term w and the total number of documents in Q is N, the Inverse Document Frequency (IDF) of w in W is





**Fig 2.4-Code of IDF.**

This means that a word which occurs less number of times in the corpus Q will have a high IDF. The reason behind this logic is that queries are composed of more informative words.

We have used a hash table to store IDF for each word in W.

* 1. **Creation of Variant Lists**

In order to calculate the similarity score of each question in Q, we first create a variant list for each SMS token. This is done by calculating weight of each word w in W with respect to each SMS token s using equation. This list is then sorted in descending order. A word is said to be a variant of the SMS token if it starts with the same character and if

Length (LCS (w, s)) >= 1

To get a more accurate and quick result we limit the list: of variants for each SMS token to five. Thus the final variant list for an SMS token s will contain its top five variants, i.e., those words that have the highest weight with respect to s.

* 1. **Creation of Candidate List Q-Poss.**

A search is performed on the corpus Q for the questions that contain the variants for an SMS token and all these questions are added to a candidate list called Q- Poss. Thus, Q-Poss. will contain all the questions that could possibly be the matching question Q\* for the SMS query S.

* 1. **Calculation of Similarity Score**

The similarity score is calculated for all the questions in Q-POSS. In order to calculate similarity score for a question q in Q-POSS, we create a list, Q-words, of the words occurring in q.

In an iterative manner, we select a word from the question q which has the maximum weight with respect to an SMS token s and add its weight is added to the similarity score for q. That word is then removed from the list Q-words. This process is repeated till the word for each SMS token is searched for. Thus, for each token s\*, the scoring function chooses the term from q having the maximum weight; then the weight of the n chosen terms are summed up to get the similarity score.



Where,

**w =** word in the question q with max Weight w.r.t. SMS token Si

* 1. **Keyword Matching**

In this section, we describe the methods used to calculate the second parameter used for calculating the score of a question, keyword score. For this, vowels and stop words are removed from each question in the list Q-POSS and these processed questions sre kept in a separate list.

**Disemvoweling:** We describe the process of removing vowels from a string as disvoweling and the string from which vowels have been removed is said to be disemvoweled. We apply this process of disvoweling to the SMS query. The reason behind using this technique is that while entering text in an SMS, the user tries to compress the text by using slangs and omitting letters and we have observed that in general, it is done by omitting some vowels from the text or difference in the usage of vowels. Vowels can also account for most of the spelling mistakes made by users.

Example: ‘transaction’ **—>** ‘trnsctn’.

Removal of Stop Words. In computing, stop words are words which are filtered out prior to, or after, processing of natural language data (text). It is controlled by human input. There is not one definite list of stop words. The list of stop words that we have used includes the most common short function words such as the, is, at, which, on, etc. and common lexical words as well. The list of stop words is disemvoweled and words occurring in the disemvoweled SMS query that are present in the list of stop words are removed from the query. This is done to increase the performance and effectiveness of the system by saving time and disk space and it also improves the process of keyword matching.

The SMS query obtained after disvoweling and removing stop words is called processed SMS query Sp.

**Calculation of the Keyword Score**:In order to calculate the keyword score of a question q in Q-POSS, we find the number of words of the SMS query it 

contains. We call the tokens (words) of the SMS query, keywords. The keyword score is a ratio of the number of keywords matched for each question to the number of keywords in the processed SMS query. Thus for a question q in Q,



Thus, a disemvoweled question that contains all the keywords has a keyword score of 1.

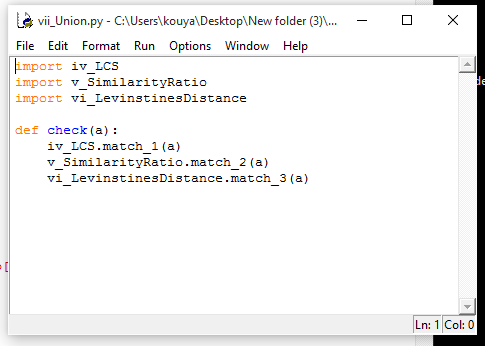
* 1. **Total Score**

The total score for a question q in Q-POSS is calculated by adding its keyword score and similarity score and is kept along with q. Finally, the score(s) of the matching question(s) is converted to a ratio by dividing it with the score of Q\*. Thus, if there is only one match its score is 1. Otherwise the scores are less than or equal to 1.



The question with the maximum total score is returned as the match Q\*, for the SMS query S.

**CHAPTER3. EXPERIMENTS AND RESULTS**

****

**Fig 3.1-Encapsulating Modules.**

This system has been tried on the given SMS queries and the provided FAQ dataset and has proved to be quite efficient. The system returns up to top 5 matching questions from the FAQ set with Q\* as the best match for the SMS query S. The score of these matches are between 0 and 1 with the score of Q\* being 1. Thus if only one match is found, its score is 1.

As many of the given queries were irrelevant and had no matching FAQ ques­tion, a minimum threshold total score was defined. If all the matching questions for an SMS query had a total score lower than this threshold score, then the query was considered to be irrelevant and “NULL” was given as the output. This threshold score also helps in determining the number of matches for an SMS query if there are any.

|  |  |
| --- | --- |
| In-domain Correct | 396/728 |
| Out-domain Correct | 1940/2677 |
| Total Score | 69% |
| MRR | 0.863 |

**3.1 Performance of the Various Components of the System**

It was found that using the technique of calculating Inverse Document Frequency (IDF) for each word gave more accurate results than what we got without using it. This has been because of the fact that the queries consist of more informative words and thus words occurring in fewer questions should haw a higher weight in comparison to common words.

The contribution of each component is given below.



The performance of the system when one of the components for calculating the similarity were excluded one at a time are tabulated below.



The method of storing the words occurring in all FAQ questions in a hash table arranged in alphabetical order proved to be much more time efficient than storing the words in a list. This was because by using this method, weights for less number of FAQ words w.r.t. to SMS tokens are calculated and also because a hash table is much more efficient than a list.

**\**

**CHAPTER4. CONCLUSION AND FUTURE WORK**

Thus, developing such an automated systems has been a challenge but this sys­tem gives a smart and efficient algorithm for answering FAQ's asked in SMS language. The results obtained for this system have been good.

As future work I would like to address the following issues:

* Using a synonym dictionary that can add similar meaning words to the variant list for an SMS token.
* Improving the accuracy of the system with respect to in-domain queries.

**CHAPTER 5 .BIBLIOGRAPHY**

1. Ratcliff, J.W., Metzener, D.: Pattern Matching: The Gestalt Approach. Dr. Dobb’s Journal, 46 (1988)
2. Gunawardena, T., Lokuhetti, M., Pathirana, N., Ragel, R., Deegalla, S.: An Auto­matic Answering System with Template Matching for Natural Language Questions. In: Proceedings of the 5th International Conference on Information and Automation for Sustainability (ICIAfS 2010), Colombo, Sri Lanka, pp. 353-358 (December 2010)
3. Contractor, D., Kothari, G., Faruquie, T.A., Subramaniam, L.V., Negi, S.: Handling noisy queries in cross language faq retrieval. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, Stroudsburg. PA, USA, pp. 87-96 (2010)
4. Kopparapu, S.K., Srivastava, A., Pande, A.: SMS based Natural Language Interface to Yellow Pages Directory'. In: Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology, Singapore (2007)
5. Sneiders, E.: Automated FAQ Answering: Continued Experience with Shallow Lan­guage Understanding. In: Question Answering Systems. Papers from the 1999 AAAI Fall Symposium. North Falmouth, Massachusetts, USA, November 5-7, pp. 97-107. Technical Report FS-99-02. AAAI Press (1999) .