`

A DISSERTATION ON

“Improving SMS Based FAQ Retrieval Using Proximity and Length Score”

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE AWARD OF DEGREE OF

MASTER OF TECHNOLOGY
IN
COMPUTER TECHNOLOGY AND APPLICATION
Delhi Technological University, Delhi

SUBMITTED BY

MUKUL RAWAT
UNIVERSITY ROLL NO: 27/CTA/2K10

Under the Guidance of:

Mr. Manoj Kumar

Associate Professor
Delhi Technological University

DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
2010-2012

`

`

CERTIFICATE

This is to certify that the work contained in this dissertation entitled "**Improving SMS Based FAQ Retrieval Using Proximity and Length Score**" submitted in the partial fulfillment, for the award of the degree of M.Tech in Computer Technology and Applications at **DELHI TECHNOLOGICAL UNIVERSITY** by **MUKUL RAWAT, University Roll No. 27/CTA/2K10,** is carried out by him under my supervision. The matter and contents embodied in this project work has not been submitted earlier for the award of any degree or diploma in any University/Institution to the best of my knowledge and belief.

(Mr. MANOJ KUMAR)
Project Guide
Associate Professor
Department of Computer Engineering
Delhi Technological University

`

ACKNOWLEDGEMENT

First of all, let me thank the almighty god and my parents who are the most graceful and merciful for their blessing that contributed to the successful completion of this project.

I feel privileged to offer sincere thanks and deep sense of gratitude to **Mr. MANOJ KUMAR**, project guide for expressing his confidence in me by letting me work on a project of this magnitude and providing their support, help & encouragement in implementing this project.

I would like to take this opportunity to express the profound sense of gratitude and respect to all those who helped us throughout the duration of this project. Delhi Technological University, in particular has been the great source of inspiration. Again, I acknowledge the effort of those who have contributed significantly to this project.

MUKUL RAWAT
University Roll No.: 27/CTA/2K10

`

Abstract

In the present scenario, we all are looking for a better way to access information. Short Messaging Service (SMS) is one of the popularly used services that provide information access to the people having mobile phones. However, there are several challenges in order to process a SMS query automatically. Humans have the tendency to use abbreviations and shortcuts in their SMS. We call these inconsistencies as noise in the SMS. In this paper we present an improved version of SMS based FAQ retrieval system. We have mainly added three improvements to the previous system. They are (i) proximity score, (ii) length score and (iii) an answer matching system. This way we noticed the improvement in the accuracy of the SMS based FAQ retrieval system. We demonstrate the effectiveness of our algorithm by considering many real-life FAQ-datasets from different domains (e.g. Agriculture, Bank, Health, Insurance and Telecom etc).

`

TABLE OF CONTENTS

Page No.

`

`

List of Figures

`

List of Tables

Chapter 1: Introduction

In this era of globalization, information retrieval has become an important part of everybody's life. Therefore, it has become an interesting area of research that is how to make information retrieval system convenient. Nowadays, there are several resources through which users can access information such as internet, telephone lines, mobile phones, etc. With the rapid growth in mobile communication, mobile phone has become a common part for most people. The number of mobile users is growing at a very fast rate. In India alone, there are around 893 million mobile subscribers [2]. The popularity of mobile phones is due to its unmatched portability. This encourages different businesses or information providers to think upon implementing information services based on mobile phones. SMS information service is one of the examples of mobile based information services. Existing SMS services such as service to access CBSE Exam Result requires user to type the message in some specific format. For example, to get the result of a particular student in CBSE examination, the user has to send a message CBSE-HS-XXXX (Where XXXX is the Roll number of the student) [3]. These are the unnecessarily constraints to the users who generally feel it easy and intuitive to type a query in a "texting" language (i.e. abbreviations and the shortcuts). Some businesses such as "ChaCha" [4] allow their users to make query through the SMSs without using any specific format. These queries, on the other hand, are handled by the human experts. However this approach provides users a kind of independence in writing the query yet this is not an efficient way because the system is limited to handle a small number of queries proportional to the number of human experts on the business side. This approach can be efficient if we have some sort of automatically handling of query at the business side. L. Venkata Subramaniam et al., August 2009, IBM India Research Lab, presented a SMS-based question answering system over a SMS interface [1]. This system enabled user to type his/her question in SMS texting language. Such questions might contain short forms, abbreviations, spelling mistakes, phonetic spellings, transliterations etc. The system handled the noise by formulating the SMS query similarity over the FAQ database. This FAQ database was already provided to the system in the pre-processing stage.

`

1.1 Motivation

India is on the brink of a mobile revolution, there are around 893 million mobile subscribers [2] here and the number is just growing. Technology is increasing day by day and new mobile phones are being introduced into the market. These days we can get many smart phones with high-end graphic, fast processors and capable of 3G internet connectivity. But still a large percentage of the mobile subscribers are using cheap phones. SMS is the major mode of communication on these phones, thanks to the low cost of these SMSs. SMS is a very economic and convenient mode of communication these days. Due to the popularity of communication based on SMS, network operators have been encouraged to base a lot of their services on the SMS technology. Most important applications of SMS are:

- Enquiry System for organizations such as schools, colleges and hospitals.
- Customer Support for telecom companies.
- Automatic FAQ support for various applications such as applications, registrations for an event etc.
- Selling advertisements.
- Search Engine for small domain.

The near availability of a mobile network in any part of the country has also fueled the growth of these services and the network operators are looking for more ways where this can be utilized. There is a very high scope for optimization in the field of SMS based information services in terms of accuracy.

1.2 Research Objective

The FAQ Retrieval System involved finding the best possible match from a given set of FAQs for a query written in texting language. Now the problem with texting language is that often there are misspellings, non-standard abbreviations, transliterations, phonetic substitutions and omissions making it difficult to build an accurate FAQ Retrieval System.

`

In this thesis my objective is to present two score based techniques, namely the proximity score and the length score, which takes into consideration the proximity of the tokens in the SMS and a FAQ question and length of matched token from SMS to the FAQ question under consideration. I will be explaining these concepts in the coming chapters and will show the experimental results and analysis which will show how these two techniques can be used to improve the accuracy of a FAQ Retrieval System.

1.3 Related Work

There has been growing interest in providing access to applications on mobile devices using SMS. A lot of SMS-based FAQ retrieval services were formed, but they used human experts to answer questions. L. Venkata Subramaniam et al., August 2009 [1], proposed an approach named SMS based FAQ retrieval. They were the first to handle issues relating to SMS based question-answering. Their method is unsupervised and does not require aligned corpus or explicit SMS normalization to handle noise. They proposed an efficient algorithm that handles noisy lexical and semantic variations.

The problem of FAQ Retrieval has been studied by Harksoo Kim [8], and he presented a way of recovering FAQs by using a clustering of previous query logs. He further improved this approach by employing latent semantic analysis [9], and by using latent term weights [10]. In [10], the use of machine translation techniques are proposed for alignment of questions and answers of FAQ corpus, aiming towards the construction of a bilingual statistical dictionary which is further used for expanding the queries which are used in the information retrieval system. In [11], an approach for domain specific FAQ retrieval is presented based on the concept of “independent aspect”.

1.4 Scope of the Work

There has been little work done to handle the issues relating to SMS based automatic question-answering. Most of the prior work related to SMS based FAQ retrieval services used human experts to answer questions. There has been a big growth in the number of mobile users throughout the world and this number is only going to increase in the future. Since most of

`

these users use low end mobile phones with SMS being the primary mode of communication, lots of services are being developed based around this technology. To develop any service based on the SMS technology related to information retrieval one cannot totally rely on human experts, since there are lots of mobile users. There is a great need for an automated SMS based information retrieval system which is very accurate.

In this thesis I have presented an automated SMS based FAQ retrieval system which is used across two different languages and tested on different domains. I found that by applying two new techniques namely the length score and proximity score, one can easily increase the accuracy of the system without adding much complexity to it. There is a huge scope to improve the accuracy of the system so that it can answer the user's query correctly, as well as to improve the performance of the system so that it can be used to answer the queries in real time.

1.5 Organization of the thesis

In this chapter, I have highlighted the introduction to the SMS Based FAQ Retrieval System, motivation to do this thesis, my objective, prior work in this field, and scope to do the work in the same field. Also, this chapter includes various difficulties that are associated to the SMS processing and various techniques that have been proposed to counter these difficulties. Chapter 2 provides an overview to the SMS Based Question Answering System; various strategies to implement SMS based Question Answering system. This chapter also describes the importance and benefits of choosing FAQ based Question Answering System. Then it describes the challenges in processing SMS queries and the strategy used to implement the system. This chapter also includes the problem formulation and system implementation details. In chapter 3, I presented the proposed techniques, the proximity score and the length score. I also presented the formulas and equations that I have derived while deducing the algorithm. Chapter 4 includes the implementation details and the experimental setup. Here, I defined the datasets that I have taken into the consideration while driving the experimental results. Also, there is a comparative analysis of the results as concluded through different techniques. Finally, Chapter 5 concludes the thesis and gives some suggestions for future work.

`

Chapter 2: Literature Review

2.1 SMS Based Question Answer System Basic Concepts

2.1.1 What is SMS Based Question Answering System?

In this age of the internet where all sorts of information is just a click away, information retrieval has become an important part of everybody's life. As a consequence of this, methods to improve information retrieval systems have become a major area of research. Today everyone is looking for a better and an easy way to access information. Users can access information using several resources such as internet, telephone lines, mobile phones, etc. There has been a rapid growth in mobile communication and mobile phone are becoming more and more common in our daily life A majority of the users still use cheaper models that have limited screen space and a basic keyboard. On such devices SMS is the only mode of test communication. This has encouraged service providers to build information based services around SMS technology. In India alone, there are 811 million mobile subscribers and the number is still growing rapidly. The SMS based Question Answering (QA) services can be one of the cheapest and easiest ways to provide information access to the mobile users on move. This popularity and ease of use encourages the information providers to provide the access to information through mobile phones. The cost of sending SMS and the easy access to the purchase of mobile phones have made instant messaging emerge as the preferred communication medium just after spoken media and e-mails. The number of SMS messages sent in 2010 was 6.9 trillion and for 2011 this number would reach over 8 trillion [6]. SMS messaging is now used not only for personal communication but also for inquiry, advertising, marketing, polling, bill payment, banking, etc. Thus, we can define SMS based question answering system as a way to access information in the form of questions and answers sing SMS service of the mobile phones.

2.1.2 Introduction to SMS based Question Answering System techniques

The most important such techniques are as follows:

`

2.1.2.1 Natural Language Processing Based

In these types of systems the user sends the queries to the system via a SMS and such queries are written in natural language which mostly in these cases means texting language. The system on receiving the query analyzes it using Natural Language Processing methods and generates an answer to it. This answer is then sent to the user via a SMS. The main disadvantage of such systems is the complexity of the natural language processing algorithms used. Its pretty hard to decompose and analyze a query and its hard to generate accurate answers. The nature of texting language is such that there is always a chance of misspellings, phonetic substitutions and omissions, non-standard abbreviations and transliterations making it difficult to analyze the query properly.

2.1.2.2 Human Intervention Based

In these systems the users can send the query written in natural language. Such system does not have an automated response, instead there are real human agents who read and understand the query and reply to the user. Human Intervention based systems uses humans to answer the questions. These systems are interesting as they suggest similar questions resolved in the past. Systems like ChaCha and Askme use qualified human experts to answer the question in timely fashion.

Some businesses have recently allowed users to formulate queries in natural language using SMS technology. Many contact centers now allow customers to text their complaints and requests for information over SMS. This mode of communication not only makes economic sense but also saves the customer from the hassle of waiting in a call queue.

2.1.2.3 Frequently Asked Question Based

Unlike other automatic question answering systems that focus on generating or searching answers, in a FAQ database the questions and answers are already provided by an expert. The goal is to identify the best matching question-answer prayer for a given query. A database for possible query along with their answers is maintained in the system. When a user makes a query over the SMS interface, the query is analyzed by the system and searched over the

`

database for the closest match with a query. The answer to the closest matching query is returned the user.

2.2 SMS Based FAQ Retrieval System

2.2.1 Problem Definition

The input SMS S is considered as a sequence of tokens S=s_1, s_2, …….s_n. Let Q denote the set of questions in the FAQ corpus. Now each questions Q belonging to the set Q is also considered as a sequence of terms. The goal of our task is to find the question Q* from the corpus Q such that it best matches the SMS S. The SMS query is written in texting language. The problem with questions asked in texting language is that such text has a lot of noise such as short forms, abbreviations, spelling mistakes, phonetic spellings, transliterations etc. This makes SMS processing a tedious task. The aim of the system is to find Q* which is as close to the input SMS S. The techniques that we apply on this system are only used to increase this accuracy i.e. find the best possible matching question Q* for the input query S.

The task of SMS based FAQ retrieval can be categorized into three categories:

1. **Mono-lingual FAQ Retrieval:** In this task, we are required to find the best matching FAQ for a given SMS query where the FAQ corpus and SMS queries are expressed in the same language.

2. **Cross-lingual:** In this task, we are required to find the matching FAQs in a language different from the SMS queries language. For example, FAQs are written in Hindi and SMS queries are coming in English.

3. **Multi-Lingual:** In this task, we are required to find matching FAQs for SMS queries where both FAQs as well as SMSs can be in any language. For example, FAQs are written in any one of these languages like English, Hindi Malyalam, etc and SMSs are coming may also belong to any one of these languages like English, Hindi, Malyalam, etc.

In this thesis, I have taken Mono-lingual and Multi-Lingual SMS based FAQ retrieval as my question answering system and proposed the algorithm for this system only.

2.2.2 SMS Noise

The millions of users of instant messaging (IM) services and short message service (SMS) generate electronic content in a dialect that does not adhere to conventional grammar, punctuation and spelling standards. This is commonly referred to as the texting language. Words are intentionally compressed by non-standard spellings, abbreviations and phonetic transliteration is used. These short forms, abbreviations, spelling mistakes, phonetic spellings, transliterations inconsistencies in SMS query are known as noise in the SMS. SMS corpus collected from FIRE SMS task has following observations for English.

1. The commonly observed patterns include deletion of vowels, addition of repeated character and truncation. For example, “abt” written after removing the vowels in the word about, “sooo” after repeating characters and “col” after truncating.

2. The SMS data provided belongs to different domains like telecommunication, railways, insurance, etc. Some of the frequently used abbreviations in these areas have been written directly like IRCTC in railways, BSNL in telecommunication, etc.

3. Substitution of spoken words with the actual spelling of the words popularly known as phonetic substitution. For example, usage of “bookin” for booking in tourism domain, etc.

4. Informal usage of different words is common in SMS text. Often multiple words are combined into a single token. For example, “wrt” for with respect to or “asap” for as soon as possible etc.

`

5. Missing words in sentences due to the limitation of text message. SMS query sometimes give keywords and miss the conjunctions, prepositions and other words which connect the key words. For example, “sms packs” used instead of sms packs available for recharge, etc.

Above problems pertaining in the SMS text make it very noisy. This makes SMS processing a tedious task.

2.2.3 Combinational Search Problem

In computer science and artificial intelligence, **combinatorial search** studies search algorithms for solving instances of problems that are believed to be hard in general, by efficiently exploring the usually large solution space of these instances. Combinatorial search algorithms achieve this efficiency by reducing the effective size of the search space or by employing heuristics. Some algorithms are guaranteed to find the optimal solution, while others may only return the best solution found in the part of the state space that was explored. Classic combinatorial search problems include solving the eight queens‟ puzzle or evaluating moves in games with a large game tree, such chess. A study of computational complexity theory helps to motivate combinatorial search. Combinatorial search algorithms are typically concerned with problems that are NP-hard. Such problems are not believed to be efficiently solvable in general. However, the various approximations of complexity theory suggest that some instances (e.g. "small" instances) of these problems could be efficiently solved. This is indeed the case, and such instances often have important practical ramifications. Combinatorial search algorithms are normally implemented in an efficient imperative programming language, in an expressive declarative programming language such as Prolog, or some compromise, perhaps a functional programming language such as Haskell, or a multi-paradigm language such as LISP. In this work, combinatorial search was implemented as the search space for matching the SMS query to a query in the FAQ database is large. As we shall see in subsequent chapters, the combinatorial search technique used employs Naive algorithm to reduce the search space of finding the maximum scoring question.

`

2.3 Problem Formulation and System Implementation

The aim of this chapter is to define various phases and functions used in the system. The purpose of the preprocessing stage is to preprocess the database in order to make the system work fast at the time of actual computation. In later stages, various scoring functions are defined that improves the system accuracy by considering different techniques from NLP and pattern matching.

2.3.1 Preprocessing involves the following steps:

1. Indexing: This is one of the important steps in preprocessing. We create a hash table of words W that contains all the words occurring in all the questions in Q with the keys being characters a-z and numbers 0-9. Example: “I” contains all the words in the set Q that start with “I”, “like”, “insurance”, “improve”, and so on. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For indexing the FAQ corpus we have used LUCENE [9].

2. Creating Domain Dictionary: In the preprocessing stage, we develop a Domain dictionary D consisting of all the terms that appear in the corpus Q.

3. Removing Stop words: Stop words are words which are filtered out prior to, or after, processing FAQ corpus. The stop words are removed from the SMS query S. We now call it processed SMS query. 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. Stop words are removed as they almost never contain the relevant or the keywords.

4. Disemvoweling: The process of removing vowels from a string is known as disemvowelling and the string from which vowels are removed is said to me disemvoweled. We apply this process of disemvowelling to the SMS query.

5. Replacing digits occurring in the SMS with words: Digits occurring in SMS token are replaced by a string based on a manually designed digit-to-string mapping (“8”→“eight”).

6. Removing single character words: Single character words in the SMS query are removed.

2.3.2 Similarity Score

The system views the SMS as a sequence of tokens. Each question in the FAQ corpus views as a list of terms. The goal is to find a question from the FAQ corpus that best matches with the SMS query and return the answer of the selected FAQ as a response of the user query (SMS). SMS string is bound to have misspellings and other distortions, which needed to be taken care of while performing the match. A domain dictionary is created containing all the terms that are present in the FAQ corpus in the developed system. For each term t in the dictionary and each token si in the SMS query, a similarity measure $\alpha(t, si)$ is defined that measures how closely the term t matches with the SMS token si. It is believed that the term t was a variant of si, if $\alpha(t, si) > 0$. A **weight function** $\omega(t, si)$ is defined by combining the similarity measure and the inverse document frequency (idf) of t in the corpus, Based on the weight function, a scoring function is defined for assigning a score to each question in the corpus Q with respect to given SMS query. The score measures how closely the FAQ matches the SMS string S. FAQ having the highest score is believed to be best matches with SMS query. The equation is given as:-

`

$$Similarity\_Score(Q) \ = \sum_{i=1}^{n} max_{t\in Q} \ and \ _{t\sim si}\omega(t, si)$$

Figure 2.1: Similarity Score

Consider a question Q Q. For each token si in SMS string S, the scoring function chooses the term having the maximum weight from Q. Summation of the weight of n chosen terms results in score of question Q. The goal was to find the question Q+ having the maximum score.

2.3.2.1 Weight Function

The weight for a term t in the dictionary w.r.t. a given SMS token si is calculated. The weight function is a combination of **Similarity Measure** between t and si and **Inverse Document Frequency** (idf) of t. The next two subsections explain the calculation of the similarity measure and the idf in detail.

2.3.2.2 Similarity Measure

Let D be the dictionary of all the terms in the corpus Q. For term t 2 D and token si of the SMS, the similarity measure α(t, si) between them is:-

$$\mathrm{\alpha(t, si)} = \begin{cases} \dfrac{\mathrm{LCS\ Ratio\ (t, si)}}{\mathrm{Edit\ distance(t, si)}} & \text{if t and si share same first character} \\ 0 & \text{Otherwise} \end{cases}$$

Figure 2.2: Computation of Similarity Measure

Where $LCS\ Ratio(t, si) = \frac{Length(LCS(t,si))}{Length(t)}$ and LCS(t,si) stands for largest common subsequence between t and si. The **Longest Common Subsequence Ratio** (LCSR) of two strings is the ratio of the length of their LCS and the length of the longer string. Since in SMS text, the dictionary term will always be longer than the SMS token, the denominator of LCSR is taken as the length of the dictionary term. We call this modified LCSR as the LCS Ratio.

The **Edit Distance** shown in above equation compares the Consonant Skeleton of the dictionary term and the SMS token. If the consonant keys are similar, i.e. the Levenshtein distance between them is less; the similarity measure defined in Equation will be high. We explain the rationale behind using the EditDistance in the similarity measure α(t, si) through an example. For the SMS token “gud” the most likely correct form is “good”. The two dictionary terms “good” and “guided” have the same LCSRatio of 0.5 w.r.t “gud”, but the EditDistance of “good” is 1 which is less than that of “guided”, which has EditDistance of 2 w.r.t “gud”. As a result the similarity measure between “gud” and “good” will be higher than that of “gud” and “guided”.

2.3.2.3 Inverse Domain Frequency

The **Inverse Document Frequency** is a measure of whether the term is common or rare across all documents. It is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient. If f number of documents in corpus Q containing a term t and the total number of documents in Q is N, then the Inverse Document Frequency (idf) of t is:

$$\boxed{idf(t) = \log \frac{N}{F}}$$

Figure 2.3: Inverse Domain frequency

Mathematically the base of the log function does not matter and constitutes a constant multiplicative factor towards the overall result. Combining the similarity measure and the idf of t in the corpus, they define the weight function ω(t, si) as:

$$\boxed{\omega(t, si) = \alpha(t, si) \ast idf(t)}$$

Figure 2.4: Weight Function

`

The objective behind the weight function is

1. It is preferred that terms having high similarity measure i.e. terms that are similar to the SMS token. Higher the LCSRatio and lower the EditDistance, higher will be the similarity measure. Thus for example, for a given SMS token “byk”, similarity measure of word “bike“ is higher than that of “break”.

2. It is preferred that words that are highly discriminative i.e. words with a high idf score. The rationale for this stems from the fact that queries, in general, are composed of informative words. Thus for example, for a given SMS token “byk”, idf of “bike” will be more than that of commonly occurring word “back”. Thus, even though the similarity measure of “bike” and “back” are same w.r.t. “byk”, “bike” will get a higher weight than “back” due to its idf.

These two objectives are combined into a single weight function multiplicatively.

`

Chapter 3: Proposed Proximity and Length Score Technique

The objective of this chapter is to introduce my approach to compute the score of candidate FAQ by considering the Proximity and Length Score. These techniques are completely different from the earlier used technique of Similarity score in [1].

While developing our system I have taken the work [1] as our base and worked my modifications on this system. In order to increase the accuracy of SMS based FAQ retrieval I have proposed few enhancements in evaluating the score of FAQ from candidate set. I proposed that accuracy can be improved by considering proximity of SMS query and FAQ tokens as well as by considering length of the matched tokens from the SMS query to the FAQ question under consideration. Thus formalizing that:

$$Score(Q) = W_1 * Similarity\_Score(Q,S) + W_2 * Proximity_{Score(Q,S)} - W_3 * Length\_Score(Q,S)$$

Where Q is the FAQ question under consideration and S = {s_1, s_2, …, s_n} is the SMS query. W_1, W_2 and W_3 are real valued weights. Their values determine the portion of Similarity Score, Proximity Score and Length Score from the overall score of the FAQ question. W_1 and W_2 are adjusted such that their sum is 1.0 (or 100%). We have given more than half weightage to Similarity score. W_3 is assigned comparatively less value, as it tries to reduce the overall score if there are variations in the length of SMS and FAQ text. To calculate Similarity Score I have employed the methods proposed in paper by (Kothari et al., 2009). While calculating the Score of the FAQ question in my experiment, I have considered data from domains such as Agriculture, Banking, Railways, Health Care, Insurance and General Knowledge in two languages (Hindi and English).

3.1 Proximity Score

To improve the accuracy of the system proposed in [1] I have introduced the concept of proximity search. I have explained the working of our proximity search technique with an example given in figure 3.1 and figure 3.2.

`

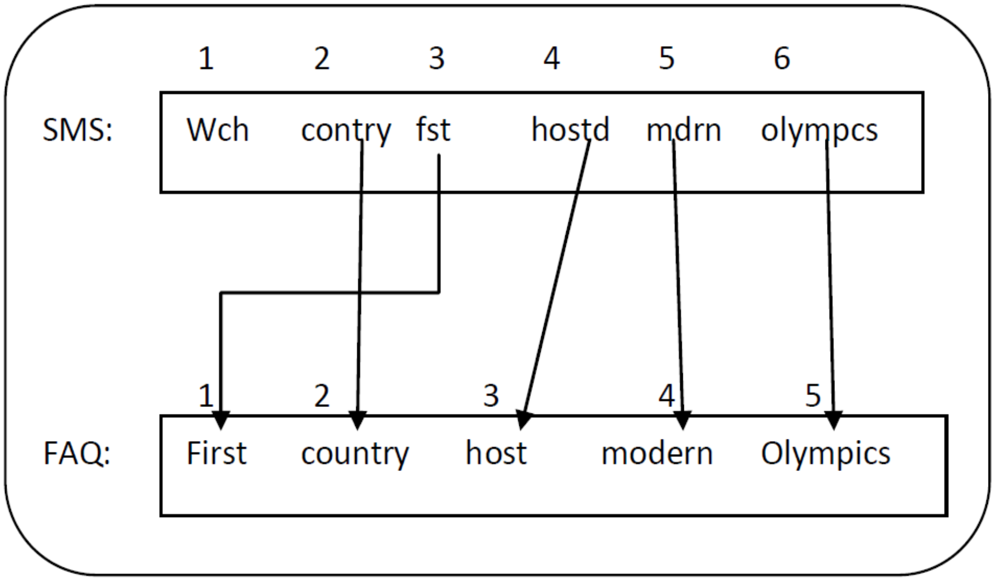

Fig 3.1 Mapping of SMS tokens with FAQ

Relative position of words in a sentence plays an important role; it allows us to differentiate this sentence with various other possible sentences – which have same words but in different order. So while finding a best match, we must consider the proximity of words. In the proposed solution we do not check proximity of a token with all remaining tokens, but we only consider two consequent words. In proximity search process, we store the positions of the matched SMS tokens and FAQ tokens, stop words are removed before storing position of tokens. Based on the distance between two consecutive tokens in SMS text and FAQ the calculation of Proximity Score is done. The proximity score can be calculated as:

$$Proximity\_Score = \frac{matchedToken}{((distance\ + 1) * \ totalFaqTokens\ )}$$

Where *totalFaqTokens* = number of tokens in FAQ

matchedToken = number of matched token of SMS in FAQ

$$distance = \sum_{k=0}^{n} \begin{array}{l}\text{absolute difference between adjacent token} \\ \text{pairs in SMS and corresponding pair in FAQ}\end{array}$$

Where n = number of matched adjacent pairs in SMS.

Figure 3.2 describes the calculation of Proximity Score with an example SMS and FAQ question. For calculating the value of distance we have taken only absolute value of distance as we believe that if two tokens were swapped their positions than in most of the cases the meaning of the SMS and FAQ question is unchanged. Unlike the Length Score, Proximity Score is always positive.

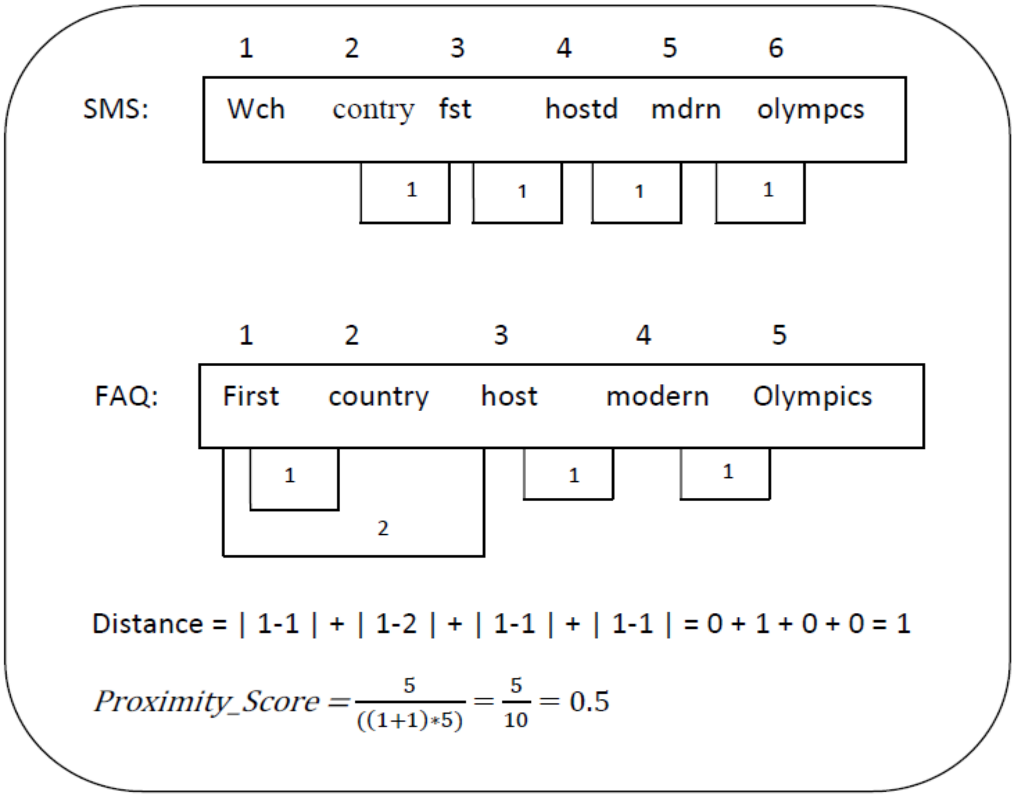
1 2 3 4 5 6
SMS:
Wch contry fst hostd mdrn olympcs
1
1
1
1
1 2 3 4 5
FAQ:
First country host modern Olympics
1
2
1
1
Distance = | 1-1 | + | 1-2 | + | 1-1 | + | 1-1 | = 0 + 1 + 0 + 0 = 1
$Proximity\_Score = \frac{5}{((1+1)*5)} = \frac{5}{10} = 0.5$

Fig 3.2 Calculation of Proximity Score

3.2 Length Score

We can further improved the accuracy of the system by considering the length of the matched tokens from SMS query to the FAQ question under consideration. Length Score is defined as follows:

$$LengthScore = \frac{totalFAQToken - matchedToken}{1 + totalSMSToken - matchedToken}$$

`

$$= \frac{unmatched\ FAQ\ Token}{unmatched\ SMS\ Token}$$

Where totalFAQToken = total number of Tokens in FAQ question

totalSMSToken = total number of Tokens in SMS

matchedToken =number of SMS which matched from tokens of FAQ question

Since the Length Score is negative score (i.e. this score is subtracted from the overall score), so best Length Score is achieved when all the tokens of the FAQ question were matched from the all tokens in the SMS query. So in the best case Length Score is Zero (i.e nothing to be subtracted from the overall score). There is a drawback of using Length Score when a question having more number of tokens would always have less overall score because there are more number of unmatched FAQ tokens. We provide two possible solutions to the above problem. The first solution is applicable when very few FAQ questions in FAQ database have more number of tokens. Solution to this problem is that rewrite the big FAQ question into the FAQ question having less number of tokens. For e.g.

Original FAQ Question: *"DTU offers various Tech courses. What are the Internship opportunities for M Tech students at DTU?Do all M Tech students get the Internship offer?"*

Corresponding Small Question: *"What are Internship opportunities for M Tech students at DTU?"*

The second solution is applicable when there are many big questions in FAQ database and rewriting them is not possible. In this case instead of subtracting the Length Score (i.e. adding negative score) we add the Length Score (i.e. adding positive score) in the overall score. In this particular case we have modified our Length Score a bit. The new formula for Length Score in this case is as follows:

$$LengthScore = \frac{1 + totalSMSToken - matchedToken}{1 + totalFAQToken - matchedToken}$$

`

$$= \quad \frac{unmatched\ FAQ\ Token}{unmatched\ SMS\ Token}$$

Where totalFAQToken=total number of Tokens in FAQ question

totalSMSToken =total number of Tokens in SMS

matchedToken =number of SMS which matched from tokens of FAQ question

In the best case Length Score would be 1 when all the tokens in FAQ were matched by all tokens in SMS.

3.3 Summary

In this chapter I have explained two new techniques to improve the efficiency of the FAQ based Question Answering System. These two techniques: Proximity Score and Length Score can be used on the original system discussed in [1]. If we combine these two scores along with the Similarity Score discussed in [1] we can further improve the efficiency of the system. The combined scores can be formulized as :

$$Score(Q) = W_1 * Similarity\_Score(Q,S) + W_2 * Proximity_{Score(Q,S)} - W_3 * Length\_Score(Q,S)$$

Where Q is the FAQ for which we are calculating the score. W1, W2 and W3 are real valued weights. Their values determine the weights of Similarity Score, Proximity Score and Length Score from the overall score of the FAQ question. W1, W2 and W3 are adjusted such that their sum is 1.0 (or 100%). More than half weightage is given to Similarity score. W3 is assigned comparatively less value, as it tries to reduce the overall score if there are variations in the length of SMS and FAQ text.

`

Chapter 4: Implementation and Experimental Results

4.1 Environmental Setup

The following configuration has been used while conducting the experiments

4.1.1 Hardware Configuration

Processor	:	Intel Core 2 Duo
Processor Speed	:	2.20GHz
Main Storage	:	4GB RAM
Hard Disk Capacity	:	80GB
Monitor	:	Dell 17”5‟ Color

4.1.2 Software Configuration

Operating System	:	Windows 7
Front end	:	Java
Back end	:	Datasets (explained in 4.2)

4.2 Datasets

Dataset used for evaluation was taken from FIRE [6] (Forum for Information Retrieval Evaluation), it contains data from many domains viz. - Agriculture, Banking, Career, General Knowledge, Health, Insurance, Online railway reservation, Sports, Telecom, Tourism. The SMS queries were also provided by FIRE in order to test the system accuracy. SMS queries were categorized as In-Domain and Out- Domain. SMS Queries for which there is a matching question in the FAQ corpus then its In-Domain SMS Query, other SMS Queries are called Out-Domain SMS Queries.

`

Experiments were conducted for two different languages Hindi and English. In our experiments we have used 200 In-Domain and 124 Out-Domain SMS for Hindi language, and 728 In-Domain and 2677 Out-Domain SMS for English Language.

Dataset Format :

The data is in an XML-based format. FAQs are placed in the input FAQ xml file and SMS queries are placed in SMS query xml file. The two formats are:

```
<FAQ>

<FAQID>ENG_CAREER_1</FAQID>

<DOMAIN>CAREER</DOMAIN>

<QUESTION>What is career counseling?</QUESTION>

<ANSWER> Career counseling is a process designed to help clients discover their passion,
choose satisfying careers, build career management skills, and improve their ability to market
and sell themselves in the job market. We utilize a holistic approach to career counseling and
are interested in helping you achieve greater satisfaction in your life and align your career goals
to match your personal goals. Career counseling is not a job placement or recruitment service
although we can help you find one depending on the career path you select.

 </ANSWER>

</FAQ>
```

Figure 4.1: FAQ Format

`

```
<SMS>

 <SMS_QUERY_ID>ENG_SMS_QUERY_I1</SMS_QUERY_ID>

<SMS_TEXT>whats need 2 change name on pport after a marriage</SMS_TEXT>

<MATCHES>

 <ENGLISH>ENG_VISA_47</ENGLISH>

 </MATCHES>

 </SMS>
```

Figure 4.2: SMS Format

4.3 Analysis and Results

Experiments were conducted for two different languages Hindi and English. The FAQ dataset was provided by Forum for Information Retrieval Evaluation (FIRE) [ref-link]. This dataset contained data from various domains – Agriculture, Banking, Career, General Knowledge, Health, Insurance, Online railway reservation, Sports, Telecom and Tourism. Table 1 show the number of FAQs and the In domain and Out domain SMS queries used in the experiments. SMS queries for which there is a matching question in the FAQ corpus then its In-Domain SMS query, other queries are called Out-Domain SMS queries.

Table 1. Number of FAQs and SMS Queries

Language	*FAQs*	*Indomain SMS*	*Outdomain SMS*	*Total SMS*
Hindi	1994	200	124	324
English	7251	728	2677	3405

As explained above, the matching between FAQ and SMS is performed based on three factors- Similarity, proximity and length. I have conducted experiments to evaluate the correctness of the systems based on these three factors in four different possible ways. As the similarity score is the base of the matching process, I have considered similarity score in all experiments. In first experiments only Similarity alone is considered for matching, in second – Similarity along with Proximity is considered, in third experiment Similarity and Length are considered and in fourth experiment all three factors are considered for matching. Table 2 and table 3 shows the result of experiments conducted for Hindi language and English language respectively. MRR indicates mean reciprocal rank. Those questions, which had score greater than a particular threshold, were considered for matching. The experiments were conducted with the same threshold for all experiments.

Table 2. Results of Hindi FAQ retrieval experiments

	Indomain Correct	*Outdomain Correct*	*MRR*
Similarity	197	6	0.9905
Similarity & Proximity	197	22	0.9905
Similarity & Length	197	97	0.9916
Similarity & Length & Proximity	198	118	0.9949

Table 3. Results of English FAQ retrieval experiments

	Indomain Correct	*Outdomain Correct*	*MRR*
Similarity	519	234	0.7529
Similarity & Proximity	520	393	0.7568
Similarity & Length	538	1981	0.8877
Similarity & Length & Proximity	521	2281	0.9041

`

These results are shown in the form of graph in Figure 4.3 and 4.4:

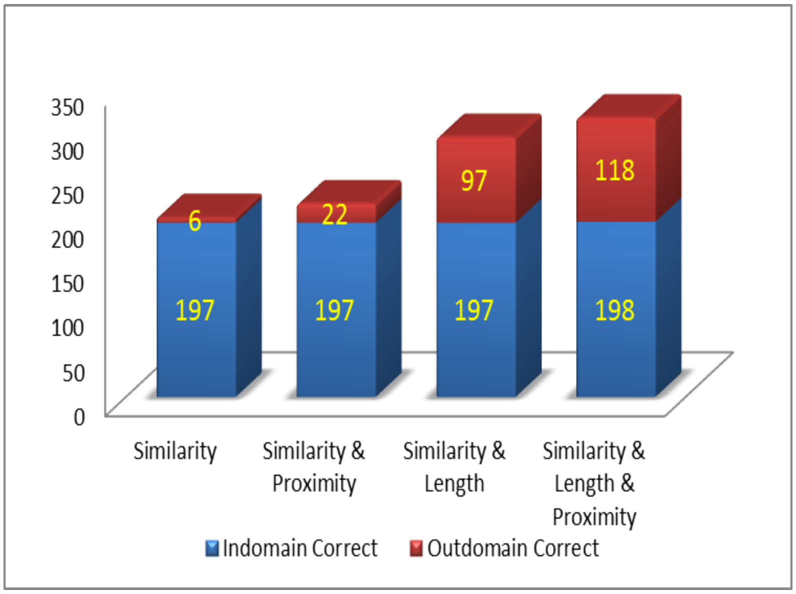

Figure 4.3 Result of Hindi FAQ retrieval task.

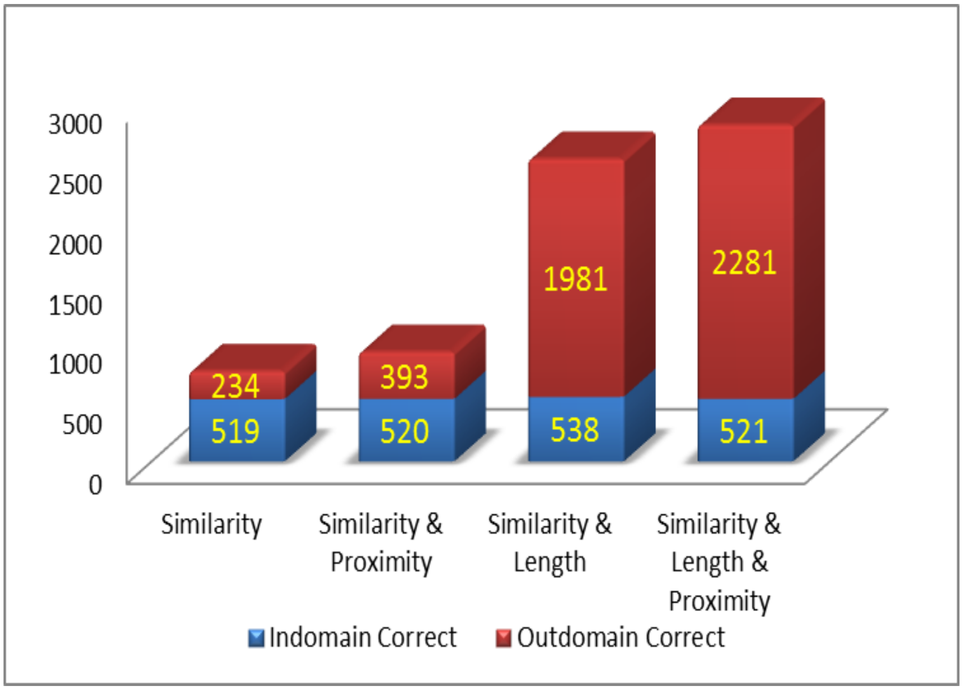

Figure 4.4 Result of English FAQ retrieval task.

`

4.4 Summary

In this chapter I have introduce and explained the experimental setup used to conduct the experiments. I have also explained the various typed of datasets used and have explained them in detail. Various experiments were conducted to test the accuracy of the proposed techniques. The results show that the Proximity factor does not improve the in domain result but improved the out domain result. Length factor improved the result by a large extent as compared to the result obtained by considering Proximity factor. And after combining the effect of Similarity, Length and proximity the results are better as compared to the previous experiments.

`

Chapter 5: Conclusion and Future Scope

5.1 Conclusion

There has been little work done on an automated SMS-FAQ Answering System. Most of the prior work done included some form of human intervention. With the growing mobile revolution that we are at present witnessing, the number of mobile users is only going to increase. A large majority of these mobile users still use cheap mobile phones and SMS is the primary mode of text communication in such phones. Therefore to monetize this fact a large number of service providers are creating applications based around the SMS technology. To make such applications a success we can see that lots of work needs to be done in the field of automated SMS based services.

In this thesis I have presented an automated SMS-FAQ Answering system based upon the work of [1]. I have discussed in detail the two new techniques which can be used to further improve the accuracy of such a system. These techniques the proximity score and the length score are both easy to implement and do not add any complexity in the existing system.

I have shown with my experiments how by applying both the proximity score and length score in combination with the similarity score we can improve the accuracy of the system to around 90% whereas earlier using only the similarity score the system only gave 75% accuracy. There are many more NLP techniques that can be applied to this system to further improve its accuracy, this work of mine only serves as an introduction to a wide practical research domain.

5.2 Future Scope

5.2.1 Stemming

In most cases, morphological variants of words have similar semantic interpretations and can be considered as equivalent for the purpose of IR applications. For this reason, a number of so-called *stemming Algorithms*, or *stemmers*, have been developed, which attempt to reduce a word to its *stem* or root form. Thus, the key terms of a query or document are represented by stems rather than by the original words. This not only means that different variants of a term

`

can be *conflated* to a single representative form – it also reduces the *dictionary size*, that is, the number of distinct terms needed for representing a set of documents. A smaller dictionary size results in a saving of storage space and processing time. For IR purposes, it doesn't usually matter whether the stems generated are genuine words or not – thus, "computation" might be stemmed to "compute" – provided that (a) different words with the same 'base meaning' are conflated to the same form, and (b) words with distinct meanings are kept separate. An algorithm which attempts to convert a word to its linguistically correct root ("compute" in this case) is sometimes called a *lemmatizer*.

Examples of products using stemming algorithms would be search engines such as Lycos and Google, and also thesauruses and other products using NLP for the purpose ` 47
of IR. Stemmers and lemmatizers also have applications more widely within the field of Computational Linguistics.

5.2.2 Inverse Bigram Frequency

Like Inverse domain frequency, we can measure inverse bigram frequency in the preprocessing stage. I believe this will improve the N-Gram score of the question hence improve the accuracy of the system.

5.2.3 Caching the Results

Caching the results would help the system in answering the repetitive queries. In this case, system needs not to search the FAQ in the full corpus every time instead it can first check the question similarity the cache if not found then go to the corpus. In general, it is the common to have a particular set of queries at particular time. For example, during admission time in a university, most of the queries would be related to the admission only.

5.2.4 N-Gram count based algorithm

N-Gram is a contiguous sequence of n items from a given sequence of text or speech. The items in question can be phonemes, syllables, letters, words or base pairs according to the application. N-Grams are collected from a text or speech corpus. An n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram".

Larger sizes are sometimes referred to by the value of n, e.g., "four-gram", "five-gram", and so on. N-Grams can be used for efficient approximate matching. By converting a sequence of items to a set of N-Grams, it can be embedded in a vector space, thus allowing the sequence to be compared to other sequences in an efficient manner. We know empirically that if two strings of real text have a similar vector representation (as measured by cosine distance) then they are likely to be similar.

`

REFERNCES

[1] GovindKothar, SumitNegi, Tanveer A. Faruquie, Venkatesan T. Chakaravarthy, L. Venkata, "SMS based interface for FAQ retrieval," Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 852–860, Suntec, Singapore, 2-7 August 2009.

[2] TRAI Annual report - http://www.trai.gov.in/annualreport/English_Front_Page.pdf.

[3] SMS service- http://results.icbse.com/cbse-result-class-10/

[4] ChaCha - http://www.chacha.com/

[5] FIRE - http://www.isical.ac.in/~clia/

[6] Global mobile statistics 2012 - http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats

[7] Kim H., Seo J., High-Performance FAQ retrieval using an automatic clustering method of query logs. Inf. Process. Manage. 42, 2006, 650-661

[8] Kim H., Lee H., Seo J., A reliable FAQ retrieval system using a query log classification technique based on latent semantic analysis, Inf. Process. Manage. 43, 2007, 420-430.

[9] Kim H., Seo J., Cluster-based FAQ retrieval using latent term weights. IEEE Intelligent Systems 23, 2008, 58-65.

[10] Riezler S., Vasserman A., Tsochabtaridis I., Mittal V., Liu Y., Statistical machine translation for query expansion in answer retrieval. In proceedings pf 45th Annual Meeting of the Association of Computational Linguistic, 2007, 464-471.

`

[11] Wu C. H., Yeh J. F., Chen M. J., Domain Specific FAQ retrieval using independent aspects, ACM Transactions on Asian Language Information Proceeding (TALIP) 4, 2005, 1-17.

[12] Sreangsu Acharya, Sumit Negi, L. V. Subramaniam, Shourya Roy. 2008. Unsupervised learning of multilingual short message service (SMS) dialect from
noisy examples, In Proceedings of the second workshop on Analytics for noisy unstructured text data.

[13] E. Prochasson, Christian Viard-Gaudin, Emmanuel Morin. 2007. Language Models for Handwritten Short Message Services, In Proceedings of the 9th International Conference on Document Analysis and Recognition.

[14] Jeunghyun Byun, Seung-Wook Lee, Young-In Song, Hae-Chang Rim. 2008. Two Phase Model for SMS Text Messages Refinement, Association for the Advancement of Artificial Intelligence. AAAI Workshop on Enhanced Messaging.

[15] Aiti Aw, Min Zhang, Juan Xiao, and Jian Su. 2006. A phrase-based statistical model for SMS text normalization, In Proceedings of COLING/ACL, pages 33−40.

[16] W. Song, M. Feng, N. Gu, and L. Wenyin. 2007. Question similarity calculation for FAQ answering, In Proceeding of SKG 07, pages 298−301.

[17] E. Sneiders. 1999. Automated FAQ Answering: Continued Experience with Shallow Language Understanding,Question Answering Systems. Papers from the 1999 AAAI Fall Symposium. Technical Report FS-99−02, November 5−7, North Falmouth, Massachusetts, USA, AAAI Press, pp.97−107.

[18] Monojit Choudhury, Rahul Saraf, Sudeshna Sarkar, Vijit Jain, and Anupam Basu. 2007. Investigation and Modeling of the Structure of Texting Language, In Proceedings of IJCAI-2007 Workshop on Analytics for Noisy Unstructured Text Data, Hyderabad.

[19] Sunil Kumar Kopparapu, Akhilesh Srivastava and Arun Pande. 2007. 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.

`

Appendix A: Coding

FAQ_INDEXER.java

```
package faqIndexing;
import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.SimpleAnalyzer;
import org.apache.lucene.analysis.WhitespaceAnalyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.*;

import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.Term;
import org.apache.lucene.index.IndexWriter.MaxFieldLength;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.store.LockObtainFailedException;
import org.apache.lucene.util.Version;
```

`

```
public class FAQ_INDEXER
{

        IndexWriter indexWriter;

        public FAQ_INDEXER(String index_dir) throws CorruptIndexException,
LockObtainFailedException, IOException
        {

                File indexDir = new File(index_dir);
                Directory fsDir = FSDirectory.open(indexDir);
                //Analyzer an = new  StandardAnalyzer(Version.LUCENE_30);
                Analyzer an = new  WhitespaceAnalyzer();
                indexWriter= new IndexWriter(fsDir,an,MaxFieldLength.UNLIMITED);
        }
         public void add_question(String faq_id,String domain,String question,String answer)
throws IOException
        {
                // question=question.replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', '
').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
                 question=question.replace('"',' ').replace("&quot;"," ").toLowerCase().trim();
                 answer=answer.replace('"',' ').replace("&quot;"," ").toLowerCase().trim();
                // answer=answer.replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', '
').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
                //create document for question
                Document doc=new Document();
                doc.add(new Field("faq_id",faq_id,Store.YES,Index.ANALYZED));
                doc.add(new Field("domain",domain,Store.YES,Index.ANALYZED));
                doc.add(new Field("question",question,Store.YES,Index.ANALYZED));
                doc.add(new Field("answer",answer,Store.YES,Index.ANALYZED));
```

`

```
                //add document to index
                indexWriter.addDocument(doc);

        }
         public void add_domain_term(String term,String synset) throws CorruptIndexException, IOException
         {
                        //create document for question
                        Document doc=new Document();
                        term = term.replace('"',' ').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
                        synset= synset.replace('"',' ').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();

                        if(term.length()>0)
                        {
                        doc.add(new Field("term",term,Store.YES,Index.ANALYZED));
                        doc.add(new Field("synset",synset,Store.YES,Index.ANALYZED));

                        //add document to index
                        indexWriter.addDocument(doc);
                        }
         }
         public void destructor() throws CorruptIndexException, IOException
        {
                //print the number of documents in index
                int numDocs = indexWriter.numDocs();
                System.out.println("Number of Documents INDEXED = "+numDocs);
                //optimize it
                indexWriter.optimize();
```

`

```
        indexWriter.close();
    }

        public void removeDuplicates() throws ParseException, IOException
        {
            File tempIndex = new File("D:\\Documents\\M tech\\IBM-SMS based FAQ retrieval\\Lucene\\INDEX\\DomainIndex - Copy\\");
            Directory fsDir = FSDirectory.open(tempIndex);

            //Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_30);

            IndexReader tmpReader = IndexReader.open(fsDir);
            IndexSearcher tmpSearcher = new IndexSearcher(tmpReader);


            for(int i=0;i<tmpSearcher.maxDoc();i++)
            {
                String domainTerm,synSet;
                domainTerm=tmpSearcher.doc(i).get("term");
                synSet=tmpSearcher.doc(i).get("synset");
                //delete all similar
                indexWriter.deleteDocuments(new Term("term",domainTerm));
                System.out.println(domainTerm);

                add_domain_term(domainTerm,synSet);
            }

            destructor();

        }
}
```

`

FAQ_Search.java

```
package faqIndexing;

import java.io.File;
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.SimpleAnalyzer;
import org.apache.lucene.analysis.WhitespaceAnalyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.store.LockObtainFailedException;
import org.apache.lucene.util.Version;

import similarity.similarity;


public class FAQ_Search {
```

```
`

        private static final double WEIGHT_THRESHOLD = 0.75;
        IndexSearcher searcher;
        QueryParser parser;
        int maxHits=3000;
        double numDocs;
        boolean debug=false;

        public FAQ_Search(String directory_path) throws CorruptIndexException,
LockObtainFailedException, IOException
        {
                File indexDir = new File(directory_path);
                Directory fsDir = FSDirectory.open(indexDir);
                //Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_30);
                Analyzer analyzer = new WhitespaceAnalyzer();

                IndexReader reader = IndexReader.open(fsDir);
                searcher = new IndexSearcher(reader);

                String defaultField = "question";
                parser  = new QueryParser(Version.LUCENE_30,defaultField,analyzer);

                numDocs  = reader.numDocs();
                System.out.println("Total Docs="+numDocs);
        }

        public DictionarySearchResult searchDomainDictionary(String token) throws
ParseException, IOException
        {
                if(debug)
                System.out.println("Searching DOmain dictionary for : "+token);
```

`

```
        token=token.replace('?',' ').replace('$',' ').replace('<',' ').replace('>',' ').replace('%','
').replace('&', ' ').replace(';',' ').replace('-',' ').replace("""," ").replace('[',' ').replace('{',' ').replace(']','
').replace('}',' ').replace('!',' ').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', '
').replace('#', ' ').replace(':', ' ').replace("", ' ').trim().toLowerCase();
        //search DOmain dictionary
        Query q = parser.parse("term:"+token.trim().charAt(0)+"*");//search for similar
first character
        TopDocs hits = searcher.search(q,maxHits);
        ScoreDoc[] scoreDocs = hits.scoreDocs;
        if(debug)
        System.out.println("number of mtching terms:"+scoreDocs.length);

        DictionarySearchResult result= new
DictionarySearchResult(token,scoreDocs.length);
        //loop over all the searched document
        for (int n = 0; n < scoreDocs.length; ++n)
        {
            ScoreDoc sd = scoreDocs[n];
            int docId = sd.doc;
            Document d = searcher.doc(docId);
            String domainTerm = d.get("term");

            if(debug)
                System.out.println(n+"] "+domainTerm);

            if(similarity.lcs(token,domainTerm)>1)
            {
                //calculate weight
                double weight=calcWeight(token,domainTerm);

                if(weight>WEIGHT_THRESHOLD)
```

`

```
                    result.addNewTerm(domainTerm, weight);
            }
        }

        return result;
    }

    public SynonymSearchResult searchSynonymDictionary(String token) throws
ParseException, IOException
    {
        if(debug)
        System.out.println("Searching Synonym dictionary for : "+token);

        //search DOmain dictionary
        //Query q = parser.parse("synset:"+token.charAt(0)+"*");//search for similar first
character

        token= token.replace('?',' ').replace('$',' ').replace('<',' ').replace('>',' ').replace('%','
').replace('&', ' ').replace(';',' ').replace('-',' ').replace('"',' ').replace('[',' ').replace('{',' ').replace(']','
').replace('}',' ').replace('!',' ').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', '
').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
        //FUZZZY match of the synonym of the SMS token
        Query q = parser.parse("synset:"+token+"~");

        TopDocs hits = searcher.search(q,maxHits);
        ScoreDoc[] scoreDocs = hits.scoreDocs;
        if(debug)
        System.out.println("number of mtching terms:"+scoreDocs.length);

        SynonymSearchResult result= new
SynonymSearchResult(token,scoreDocs.length*2);//CHECK IT
```

```
//loop over all the searched document
for (int n = 0; n < scoreDocs.length; ++n)
{
    ScoreDoc sd = scoreDocs[n];
    int docId = sd.doc;
    Document d = searcher.doc(docId);
    String domainTerm = d.get("term");
    String synset = d.get("synset");

    //check if the DOmain term is already processed or not
    if(result.isProcessed(domainTerm)==true)
            continue;

    if(debug)
    System.out.println(n+"] "+domainTerm+"-->"+synset);

    //get synset tokens separated by Comma
    StringTokenizer que_tok = new StringTokenizer(synset,",");

    int total=que_tok.countTokens();

    for(int i=0;i<total;i++)
    {
        String synonym=que_tok.nextToken();

        if(debug)
            System.out.println(token+"="+synonym);

        //separate out the tokens
        if(synonym!=domainTerm && similarity.lcs(token,synonym)>1)
```

`

```
                {
                    //calculate weight
                    double
weight=calcSynonymWeight(token,domainTerm,synonym);

                    if(weight>WEIGHT_THRESHOLD)
                        result.addNewTerm(domainTerm,
weight,synonym);
                }
            }

        }

        return result;
    }
    public double calcWeight(String smsToken,String domainTerm) throws
ParseException, IOException
    {
        double alpha;
        double weight;
        alpha=similarity.similarityMeasure(domainTerm,smsToken);
        weight=alpha*getIDF(domainTerm.replace("!", ""));

        if(debug)
            System.out.println("ALpha="+alpha);

        return weight;

    }
    public double calcSynonymWeight(String smsToken,String domainTerm,String
synTerm) throws ParseException, IOException
```

`

```
          {
                double alpha;
                double weight;
                alpha=similarity.similarityMeasure(synTerm,smsToken);
                weight=alpha*getIDF(domainTerm);
                return weight;

          }
        public QuestionSearchResult searchQuestionAnswer(String query) throws IOException,
ParseException
        {



                query= query.replace('?',' ').replace('$',' ').replace('<',' ').replace('>',' ').replace('%','
').replace('&', ' ').replace(';',' ').replace('-',' ').replace(""","  ").replace('[',' ').replace('{',' ').replace(']','
').replace('}',' ').replace('!',' ').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', '
').replace('#', ' ').replace(':', ' ').replace("", ' ').trim().toLowerCase();

                Query q = parser.parse("question:"+query+" OR answer:"+query);
                TopDocs hits = searcher.search(q,maxHits);
                ScoreDoc[] scoreDocs = hits.scoreDocs;



                //calculate IDF
                int docFreq  = scoreDocs.length;
                double idf = 1+ Math.log10(numDocs/(docFreq+1));

                if(debug)
                {
                System.out.println("Query = "+query+" Found ="+docFreq+" IDF ="+idf);
                }
```

`

```
        //variable to return the results
        QuestionSearchResult result=new QuestionSearchResult(scoreDocs.length);

        //result.faqID=new String[scoreDocs.length];
        //result.count=scoreDocs.length;

        //loop over all the searched document
        for (int n = 0; n < scoreDocs.length; ++n)
        {
            ScoreDoc sd = scoreDocs[n];
            float score = sd.score;
            int docId = sd.doc;
            Document d = searcher.doc(docId);
            String faq_id = d.get("faq_id");

            /* store faqid in results */
            //result.faqID[n] = faq_id;
            result.addResult(faq_id);

            if(debug)
            {
                String question= d.get("question");
                //String answer = d.get("answer");
                String domain = d.get("domain");
                System.out.println("-----------------FAQ_ID="+faq_id+"
Domain="+domain+" Score="+score+"-------------------------");
                System.out.println("Question="+question);
                //System.out.println("ANswer="+answer);

            }
```

`

```
                }
                return result;
        }
//       public double getIDF(String query) throws ParseException, IOException
//      {
//
//               //IDF is calculated over the questions
//              Query q = parser.parse(query);
//              TopDocs hits = searcher.search(q,maxHits);
//              ScoreDoc[] scoreDocs = hits.scoreDocs;
//
//              double docFreq  = scoreDocs.length;
//
//              double IDF= 1+Math.log10(numDocs/(docFreq+1));
//
//              if(debug)System.out.println("Query = "+query);
//              if(debug)System.out.println("IDF ="+IDF);
//
//              return IDF;
//      }
         public double getIDF(String query) throws ParseException, IOException
                {
                        IndexSearcher searcher;
                        QueryParser parser;

                        query = query.replace('?',' ').replace('$',' ').replace('<',' ').replace('>','
').replace('%',' ').replace('&', ' ').replace(';',' ').replace('-',' ').replace("'"," ').replace('[','
').replace('{',' ').replace(']',' ').replace('}',' ').replace('!',' ').replace('(', ' ').replace('.', ' ').replace(')', '
').replace(',', ' ').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
                        File indexDir = new File("D:\\Documents\\M tech\\IBM-SMS based FAQ
retrieval\\Lucene\\INDEX\\FAQIndex\\");
```

`

```
        Directory fsDir = FSDirectory.open(indexDir);
        //Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_30);
        Analyzer analyzer = new WhitespaceAnalyzer();

        IndexReader reader = IndexReader.open(fsDir);
        searcher = new IndexSearcher(reader);

        String defaultField = "question";
        parser  = new QueryParser(Version.LUCENE_30,defaultField,analyzer);

        double numDocs  = reader.numDocs();

         //IDF is calculated over the questions
        Query q = parser.parse("question:"+query);
        TopDocs hits = searcher.search(q,maxHits);
        ScoreDoc[] scoreDocs = hits.scoreDocs;

        double docFreq  = scoreDocs.length;

        double IDF;

        if(docFreq==0)
            IDF=0;
        else
            IDF= Math.log10(numDocs/(docFreq+1));



        if(debug)System.out.println("***Query = "+query+"\tTotal
Docs="+numDocs+"\tMatching Docs="+docFreq+"\tIDF ="+IDF);

        return IDF;
```

```
`

        }

    public String getQuestionByFaqID(String faqID) throws IOException, ParseException
    {
            /** NOTE: the ANALYZER is CHANGED **/

            IndexSearcher searcher;
            QueryParser parser;

            File indexDir = new File("D:\\Documents\\M tech\\IBM-SMS based FAQ
retrieval\\Lucene\\INDEX\\FAQIndex\\");
            Directory fsDir = FSDirectory.open(indexDir);
            //Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_30);
            Analyzer analyzer = new WhitespaceAnalyzer();

            IndexReader reader = IndexReader.open(fsDir);
            searcher = new IndexSearcher(reader);

            String defaultField = "question";
            parser  = new QueryParser(Version.LUCENE_30,defaultField,analyzer);

            /** END NOTE **/

            Query q = parser.parse("faq_id:"+faqID);
            TopDocs hits = searcher.search(q,maxHits);
            ScoreDoc[] scoreDocs = hits.scoreDocs;
            int docFreq  = scoreDocs.length;

            if(debug)
            {
            System.out.println("Query = "+faqID);
```

`

```
            System.out.println("Found ="+docFreq);
            }

            ScoreDoc sd = scoreDocs[0]; //get first doc as there is only one matching question;

            int docId = sd.doc;
            Document d = searcher.doc(docId);

            String question = d.get("question");

            return question;

    }

    public static void main(String[] args)
    {

        System.out.println("&quot;test&quot;".replace("&quot;", ""));
        try {
            FAQ_Search a = new FAQ_Search("D:\\Documents\\M tech\\IBM-SMS based FAQ retrieval\\Lucene\\INDEX\\FAQIndex\\");
            a.getIDF("terminator");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

`

smsparser.java

```
package parsing;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.XMLReader;

import smsProcessing.faqRetrieval;

import java.util.StringTokenizer;

public class smsparser extends DefaultHandler{

        static String queryid;
        static String text;
        static String matchenglish;

        static int score;
        static int sum=0;
        static int totalSMScount=0;
        boolean queryFlag = false;
   boolean textFlag = false;
   boolean matchenglishFlag = false;
   boolean test = false;
```

```
`



   /* The main class smsparser. This class parses an XML file consisting of all the sms.
    * The sms queryid , the text and the resulting answers are extracted. Then the queryid and text
    * are sent to the naive algorithm. The resulting answers are matched with the expected
answers
    * and the score is calculated.
    */

   public smsparser(){


                   System.out.println(" Object Created ");


          }



/*
 * This event marks the start of the XML document
 */
          public void startDocument() throws SAXException {
              System.out.println("SMS XML BEGINS HERE ");
            }



          /*
           * This event marks the start of an element in the XML document,
           * we match out required elements here
           */
         public void startElement(String uri, String localName,
                String qName, Attributes attributes)
                throws SAXException {
```

`

```
      if (qName.equalsIgnoreCase("SMS_QUERY_ID")) {
        queryFlag = true;
      }

      if (qName.equalsIgnoreCase("SMS_TEXT")) {
        textFlag = true;
      }

      if (qName.equalsIgnoreCase("ENGLISH")) {
        matchenglishFlag = true;
      }



    }


/*
 * This marks the end of element in the XML document.
 */
   public void endElement(String uri, String localName,
       String qName)
       throws SAXException {


         // Here we check wether the SMS element has ended or not.
         if(qName == "SMS"){

                 //We are here checking for the start of
```

`

```
                     test=queryid.startsWith("ENG");

                     if(test == true ){

                    String result = "";
                    //Here we are calling the naive algorithm
                    try {
                                            result = faqRetrieval.getMatchingQuestion(text);
                                } catch (Exception e){}

                    //result=dummy(queryid,text);

                    //The compute score algorithm is called here
                    score=computescore(result,matchenglish);



                    totalSMScount++;

                    System.out.println("\n"+queryid+" "+text+"\tScore is "+score);



                     }

         }


 }


//For each required element, the string is stored here
 public void characters(char ch[], int start, int length)
```

`

```
  throws SAXException {
  if (queryFlag) {
    queryid=new String(ch, start, length);
    queryFlag = false;
   }

   if (textFlag) {
     text=new String(ch, start, length);
     textFlag = false;
    }

   if (matchenglishFlag) {
     matchenglish=new String(ch, start, length);
     matchenglishFlag = false;
    }



  }


/* The compute score algorithm. Here we match the returned result with the
 * stored answers. For exact matches a 1 is returned otherwise a 0 is returned.
 */
int computescore(String result, String matchenglish){

    boolean check;
    int a,b,i,n;
    String x,y;
```

`

```
if(result==null)
        return 0;

StringTokenizer st = new StringTokenizer(result,",");
StringTokenizer st1 = new StringTokenizer(matchenglish,",");

a = st.countTokens();
b = st1.countTokens();



if(a!=b){
        sum=sum+0;
        return(0);
}

for(i=0;i<a;i++){
        x=st.nextToken();
        y=st1.nextToken();

       x=x.trim();
       y=y.trim();



        if(!x.equals(y)){
                sum=sum+0;
                return(0);


        }

}
```

`

```
                 System.out.println("Match");
                 System.out.println(result);
                 System.out.println(matchenglish);
                 System.out.println();
                 sum=sum+1;
                 return(1);

            }



/*
 * Marks the end of the document. Here we print the final sum of scores.
 */
        public void endDocument(){

                System.out.println("Total SMS queires "+ totalSMScount);
                System.out.println(" The Final score is "+ sum);

        }




        public static void main(String args[]){

                smsparser f1= new smsparser();

                 XMLReader xmlReader = null;

            try {

              SAXParserFactory spfactory = SAXParserFactory.newInstance();
`


                spfactory.setValidating(false);

                SAXParser saxParser = spfactory.newSAXParser();

                xmlReader = saxParser.getXMLReader();

                xmlReader.setContentHandler(new smsparser());

                xmlReader.setErrorHandler(new smsparser());

               // InputSource source = new InputSource("D:\\Documents\\M tech\\IBM-SMS based
FAQ
retrieval\\FIRE_PREVIEW_DATA_ok\\FIRE_TRAINING_DATA\\FIRE_TRAINING_DATA\\
SMS_Queries\\Monolingual Task\\English\\eng-mono.xml");
                //InputSource source = new InputSource("D:\\Documents\\M tech\\IBM-SMS based
FAQ retrieval\\Sample SMS queries\\SMS Queries\\agricultureSMS.xml");
                InputSource source = new InputSource("D:\\Documents\\M tech\\IBM-SMS based
FAQ
retrieval\\FIRE_PREVIEW_DATA_ok\\FIRE_TRAINING_DATA\\FIRE_TRAINING_DATA\\
SMS_Queries\\training.xml");
                xmlReader.parse(source);

              } catch (Exception e) {
                  System.err.println(e);
                  System.exit(1);
              }

                  }

}
```

`

candidateset.java

```
package smsProcessing;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.store.LockObtainFailedException;

import similarity.similarity;

import faqIndexing.DictionarySearchResult;
import faqIndexing.FAQ_Search;
import faqIndexing.QuestionSearchResult;

public class CandidateSet {
        private static final double SCORE_THRESHOLD = 46;
        private static final double SIMILARITY_THRESHOLD = 0.16;
        String[] faq_id;
        double[] score;
        String[] smsTokens;
        int totalSmsTokens;

        int count;
        static int MAXQUESTIONS = 5000;
        FAQ_Search srch;

        //Constructor to initialize strings
```

`

```
    public CandidateSet(String smsText) throws CorruptIndexException,
LockObtainFailedException, IOException
    {
        faq_id=new String[MAXQUESTIONS];
        count=0;

        srch=new FAQ_Search("D:\\Documents\\M tech\\IBM-SMS based FAQ
retrieval\\Lucene\\INDEX\\FAQIndex\\");

        //separate out question tokens
        StringTokenizer que_tok = new StringTokenizer(smsText," ");
        totalSmsTokens=que_tok.countTokens();

        smsTokens=new String[totalSmsTokens];

        for(int i=0;i<totalSmsTokens;i++)
        {
            smsTokens[i]=que_tok.nextToken();
        }

    }

    void addCandidate(String que_id)
    {
        if(count>=MAXQUESTIONS)
        {
            System.err.println("Candidate set exceeds maximum number of
questiosn");
            return;
        }
        //check if que is repeated
```

`

```
		for(int i=0;i<count;i++)
		{
			if(faq_id[i].equals(que_id))
				return;
		}

		faq_id[count]=que_id;
		count++;
	}

	public void generateCandidateSet(DictionarySearchResult LIST) throws
CorruptIndexException, LockObtainFailedException, IOException, ParseException
	{
		//for dictionary lookup
		QuestionSearchResult srchResult;

		//for each term in question - find out the corresponding question
		for(int i=0;i<LIST.getCount();i++)
		{
			srchResult=srch.searchQuestionAnswer(LIST.getTermAt(i));
			// add all search results to the Candidate set
			for(int j=0;j<srchResult.getCount();j++)
			{
				addCandidate(srchResult.getFaqIdAt(j));
			}
		}
	}

	double calculateScore(String faqID) throws IOException, ParseException
	{
		String faq=srch.getQuestionByFaqID(faqID);
```

`

```
        StringTokenizer faq_tok = new StringTokenizer(faq," ");
        int totalFaqTokens=faq_tok.countTokens();
        String[] faq_tokens=new String[totalFaqTokens];

        //copy the FAQ tokens
        for(int i=0;i<totalFaqTokens;i++)
        {
            faq_tokens[i]=faq_tok.nextToken().replace('(', ' ').replace('!',' ').replace('.', '
').replace(')', ' ').replace(',', ' ').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ');
        }

        double score=0;

        String smsToken;
        String faqToken;

        //1.For each sms token
        for(int i=0;i<totalSmsTokens;i++)
        {
            smsToken = smsTokens[i];

            smsToken = smsToken.replace('(', ' ').replace('.', ' ').replace(')', '
').replace(',', ' ').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();



    if(faqID.equals("ENG_AGRICULTURE_83")||faqID.equals("ENG_AGRICULTURE_7
6")||faqID.equals("ENG_AGRICULTURE_78"))
                System.out.println("--------------------------------SMS
token:"+smsToken+"----------------------------------");

            double maxWeight=0;
```

`

```
            double weight = 0;

            //2. compare it with each Term present in the question.
            //   get the maximum weight of the term and smsToken
            for(int j=0;j<totalFaqTokens;j++)
            {
                faqToken=faq_tokens[j];

                faqToken = faqToken.replace('(', ' ').replace('.', ' ').replace(')', '
').replace(',', ' ').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', ' ').trim().toLowerCase();

                double alpha=similarity.similarityMeasure(faqToken,smsToken);

                double IDF = 0;

                if(alpha>SIMILARITY_THRESHOLD)
                {
                    //TODO - change this formula
                    IDF=srch.getIDF(faqToken);
                    weight=alpha*IDF;
                    //weight=alpha;

                    if(weight>maxWeight)
                        maxWeight=weight;
                }

    if(faqID.equals("ENG_AGRICULTURE_83")||faqID.equals("ENG_AGRICULTURE_7
6")||faqID.equals("ENG_AGRICULTURE_78"))
                    System.out.println("#FAQ ID:"+faqID+"
Token:"+faqToken+" Alpha="+alpha+" IDF="+IDF+" weight="+weight+"
maxWeight="+maxWeight);
```

`

```
        }
        score=score+maxWeight;
    }
    //convert the score into scale of 100 .
    // 3 is assumed as 100%
    double NormalizedScore = score*33.33 / totalSmsTokens;
    return NormalizedScore;
}
void sortByScore()
{
    double tempScore;
    String tempFaqID;

    for(int i=0;i<count;i++)
        for(int j=i+1;j<count;j++)
        {
            if(score[i]<score[j])
            {
                tempScore=score[i];
                score[i]=score[j];
                score[j]=tempScore;

                tempFaqID=faq_id[i];
                faq_id[i]=faq_id[j];
                faq_id[j]=tempFaqID;


            }
        }
}
public void printCandidateSet()
{
```

```
`

        System.out.println("Candidate set of questions ="+count);

        for(int i=0;i<count;i++)
        {
            if(i%5==0)
                System.out.println();

            System.out.print("\t"+faq_id[i]);
        }
    }

    String NaiveAlgorithm() throws IOException, ParseException
    {
        //initialize score variable
        score=new double[count];
        //calculate score of all candidate questions
        for(int i=0;i<count;i++)
        {
            score[i]=this.calculateScore(faq_id[i]);
        }
        //sort the questions
        this.sortByScore();

        String result=null;

        boolean start=false;

        if(count>0)
            result=faq_id[0];

//      for(int i=0;i<count;i++)
```

```
`

//      {
//          if(score[i]>SCORE_THRESHOLD){
//              if(start)
//              {
//                  result = result + ","+ faq_id[i];
//              }
//              else
//              {
//                  result= faq_id[i];
//                  start=true;
//              }
//          }
//      }

        //show the matching questions
        this.showMatchingQuestions();

        return result;

    }

    private void showMatchingQuestions() throws IOException, ParseException
    {
        System.out.println();
        System.out.println("-----------------------------------------------------------------------
------------------");

        int top=10;

        if(count<10)
            top=count;
```

`

```
                    for(int i=0;i<top;i++)
                    {
                              System.out.println(i+"."+faq_id[i]+" "+score[i]+"%
"+srch.getQuestionByFaqID(faq_id[i]));
                    }
          }
}
```

`

faqretrieval.java

```
package smsProcessing;

import java.io.IOException;

import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.store.LockObtainFailedException;

import faqIndexing.FAQ_Search;

public class faqRetrieval
{
	static boolean debug=false;
	public static String getMatchingQuestion(String sms) throws CorruptIndexException, LockObtainFailedException, IOException, ParseException
	{
		System.out.println("SMS Query:"+sms);

		//for dictionary lookup
		FAQ_Search srch=new FAQ_Search("D:\\Documents\\M tech\\IBM-SMS based FAQ retrieval\\Lucene\\INDEX\\DomainIndex\\");

		String question = null;
		//remove single characters
		String smsText=listcreation.removeSingleLetters(sms);
		if(debug)
		System.out.println("Single Char removed :"+smsText);
		//replace number by string
		smsText = listcreation.replaceNumByWord(smsText.toLowerCase());
		if(debug)
`

        System.out.println("Number Replacement  :"+smsText);
        //create list of tokens
        question=listcreation.createList(smsText,srch);
        //retrieve matching questions

        return question;
    }

    public static void main(String[] args)
    {
        try {
            //getMatchingQuestion("How do I reduce financial risk to my farm from natural disasters like floods or drought");
            //getMatchingQuestion("What topics will I Study for online agriculture programs");
            //getMatchingQuestion("wts composing");
            getMatchingQuestion("are the carier conselling sessionss confidensial");
            //getMatchingQuestion("What products will no longer be provided when the agriculture weather service program is eliminated");
            //getMatchingQuestion("if due dat is gon whr to submit form");
            //getMatchingQuestion("whr can i find info abt pesticide estb reg and rep");
            //getMatchingQuestion("can you gimme info abt bulk repakaging pesticides");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

`

APPENDIX B: LEVENSHTEIN DISTANCE

The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character. 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:

1. **k**itten → **s**itten (substitution of 's' for 'k')
2. sitt**e**n → sitt**i**n (substitution of 'i' for 'e')
3. sittin → sittin**g** (insertion of 'g' at the end).

APPLICATIONS

In approximate string matching, the objective is to find matches for short strings, for instance, strings from a dictionary, in many longer texts, in situations where a small number of differences is to be expected. Here, one of the strings is typically short, while the other is arbitrarily long. This has a wide range of applications; for instance, spell checkers, correction systems for optical character recognition, and software to assist natural language translation based on translation memory.

The Levenshtein distance can also be computed between two longer strings, but the cost to compute it, which is roughly proportional to the product of the two string lengths, makes this impractical. Thus, when used to aid in fuzzy string searching in applications such as record linkage, the compared strings are usually short to help improve speed of comparisons. Levenshtein distance is not the only popular notion of edit distance. Variations can be obtained by changing the set of allowable edit operations: for instance,

- Length of the longest common subsequence is the metric obtained by allowing only addition and deletion, not substitution;

- The Damerau–Levenshtein distance allows addition, deletion, substitution, and the transposition of two adjacent characters;

- The Hamming distance only allows substitution (and hence, only applies to strings of the same length).

Edit distance in general is usually defined as a parametrizable metric in which a repertoire of edit operations is available, and each operation is assigned a cost (possibly infinite). This is further generalized by DNA sequence alignment algorithms such as the Smith–Waterman algorithm, which make an operation's cost depend on where it is applied.

COMPUTING LEVENSHTEIN DISTANCE

Computing the Levenshtein distance is based on the observation that if we reserve a matrix to hold the Levenshtein distances between all prefixes of the first string and all prefixes of the second, then we can compute the values in the matrix by flood filling the matrix, and thus find the distance between the two full strings as the last value computed. A straightforward implementation, as pseudo code for a
function *LevenshteinDistance* that takes two strings, *s* of length *m*, and *t* of length *n*, and returns the Levenshtein distance between them:

```
`

intLevenshteinDistance(char s[1..m], char t[1..n]) {
    // for all i and j, d[i,j] will hold the Levenshtein distance between
    // the first i characters of s and the first j characters of t;
    // note that d has (m+1)x(n+1) values
    declareint d[0..m, 0..n]
    for i from 0 to m d[i, 0] := i // the distance of any first string to an empty second string
    for j from 0 to n d[0, j] := j // the distance of any second string to an empty first string
    for j from 1 to n
    {
        for i from 1 to m
        {
            if s[i] = t[j]
            then d[i, j] := d[i-1, j-1] // no operation required
            else
                d[i, j] := minimum
                (
                    d[i-1, j] + 1, // a deletion
                    d[i, j-1] + 1, // an insertion
                    d[i-1, j-1] + 1 // a substitution
                )
        }
    }
    return d[m,n]
```

`

```
}
```