`

A
Dissertation
On

"N-Gram Driven SMS Based FAQ Retrieval System"

Submitted in Partial fulfillment of the requirement
For the award of Degree of

MASTER OF TECHNOLOGY
Computer Technology and Application
Delhi Technological University, Delhi

SUBMITTED BY

MUKUL JAIN
University Roll No: 11/CTA/2K10

Under the Guidance of:

Mr. Manoj Kumar

Associate Professor
Delhi Technological University

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
DELHI UNIVERSITY
2010-2012

`

CERTIFICATE

This is to certify that the work contained in this dissertation entitled “**N-Gram Driven SMS Based FAQ Retrieval System**” 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 JAIN, University Roll No. 11/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 JAIN
University Roll No.: 11/CTA/2K10

`

ABSTRACT

In the present scenario, everyone is looking for a better, efficient and easy way to access information. Resource availability and its user friendliness are directly encouraging a large group of people to access information more conveniently. Short Messaging Service (SMS) is one of the most popularly used services that provide information access to the people having mobile phones. In India alone, there are around 811 million mobile subscribers and still growing with a fast rate [3]. So, 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.

However, there are several significant challenges in order to process a SMS query automatically. Humans have tendency to use abbreviations and shortcuts in their SMSes. We call these inconsistencies as noise in the SMSes. Existing SMS services such as service to access Examination result requires user to type the message in some specific format. These are the unnecessarily constraints to the users who generally feel it convenient to type a query in a "texting" language (i.e. including abbreviations and the shortcuts). Some businesses such as "ChaCha" [5] allow their users to make query through the SMSes without using any specific format. But these services are not automatic and the SMSes are handled by human experts. Though this kind of system provides independence to the users in writing the SMS query but this approach is not an efficient way to handle user's queries because the system is limited to handle a small number of queries proportional to the number of human experts on the business side. The approach can be efficient if we have a system which automatically handles the query at business side.

In this thesis work, I presented a novel approach to handle these inconsistencies in the SMSes efficiently. This approach for SMS based FAQ retrieval system took N-gram counts into consideration while calculating the score for the question in the corpus. The experimental results demonstrates that this approach is significantly improves the accuracy of previous SMS based QA system as proposed by L. Venkata Subramaniam et al., August 2009 [1]. I demonstrate my algorithm over many real-life FAQ-datasets from different domains (e.g. Agriculture, Bank, Health, Insurance and Telecom etc.).

`

TABLE OF CONTENTS

`

`

`

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. There are several resources through which users can access information such as internet, telephone lines, mobile phones, etc. Mobile phones are the most convenient and easy way to access information on move. Number of mobile subscription is growing at a very high rate and mobile has become a daily necessity for most of the people. In India alone, there are around 811 million mobile subscribers [3]. The popularity of mobile phones is due to its unmatched portability. This popularity and ease of use encourages the information providers to provide the access to information through mobile phones. The most popular data application on mobile phones is SMS text messaging. The number of SMS messages sent in 2010 was 6.9 trillion and for 2011 this number would reach over 8 trillion [7]. SMS messaging is now used not only for personal communication but also for inquiry, advertising, marketing, polling, bill payment, banking, etc. SMS based QA service is one of the good examples of mobile based information retrieval services. Existing SMS based services such as service to access Examination result (e.g. CBSE Examination Result) requires user to type the message in some specific format. For example, to get the result of a particular student, the user has to send a message CBSE-HS-XXXX (Where XXXX is the Roll number of the student) [4]. 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. using abbreviations and the shortcuts). Some businesses such as "ChaCha" [5] allow their users to make query through the SMSes without using any specific format. But these services are not automatic and handled

`

SMSes through human experts. Though this kind of system provides independence to users in writing SMS query but this is not an efficient way to handle user’s requests because the system is limited to handle a small number of queries proportional to the number of human experts sitting on the business side. This approach can be efficient if we deduce a system that automatically handles the query at business side. The task to automate SMS based QA System is one the most challenging as well as commercially attractive problem in the field of Information Retrieval. 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.

1.1 Motivation

Because of the simplicity, utility and popularity of SMSes, there has been a growing interest in providing access to applications, traditionally available on Internet, on mobile devices using SMSes. The users can file their complaints, ask queries and get information by just sending an SMS. This mode of communication not only makes economic sense but also saves the customer from the hassle of arranging resources such as internet connection and computing devices. These new form of information access has the potential to cover almost all the audience from everywhere such as users from different geographical areas and different living’s classes. Most important applications among several of SMS based FAQ Retrieval systems are:

- Enquiry system for organizations such as school, college or hospital.

`

- Customer support for the companies such as telecom, software, hardware, etc.
- Automatic FAQ support for various activities such as applications, registrations for an event etc.
- Search engine for small domains.

The anytime anywhere access provided by mobile networks and the portability of handsets coupled with the strong human urge to quickly find answers has fueled the growth of information based services on mobile devices. These services can be simple advertisements, polls, alerts or complex applications such as browsing, search and e-commerce. The latest mobile devices come equipped with high resolution screen space, inbuilt web browsers and full message keypads. However, a majority of the users still use cheaper models that have limited screen space and basic keypad. On such devices, SMS is the only mode of text communication. This has encouraged service providers to build information based services around SMS technology. There is a huge scope of optimizations in the field of SMS based Information services in terms of accuracy as well as performance.

1.2 Research Objective

This FAQ Retrieval system is designed to find a match from the given set of FAQs 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 such as short forms, abbreviations, spelling mistakes, phonetic spellings, transliterations etc. This makes SMS processing a tedious task. Our task is to find the best matching FAQ from the FAQ corpus with respect to the input SMS query with accuracy.

`

In this thesis, my objective is **to present a novel approach by developing an N-gram count based algorithm that takes the count of various N-grams (monograms, bigrams, trigrams, etc.) into the account in order to calculate the score of the questions from the corpus in order to calculate the score of the questions in the corpus**. In this way, we can further improve the accuracy of the SMS based FAQ systems significantly by refining the results of the system using N-gram count based scoring function.

1.3 Related Work

The study SMS based QA system for information access has not started long ago. Allowing user to use a “texting” language (i.e. abbreviations and the shortcuts) makes this task more challenging. L. Venkata Subramaniam et al., August 2009, proposed an approach named SMS based FAQ retrieval. The proposed system was a SMS based question answering system in which user is allowed to ask question in the SMS texting language. The system was provided with a FAQ corpus containing all possible frequently asked questions (FAQ). The noise in the SMS query was handled by formulating the query similarity over the FAQ database as a combinatorial search problem. Anwar D. Shaikh et al., December 2011 [2], proposed an idea of proximity based score and length based score that take the proximity of tokens and length of the SMSes into the consideration while calculating the score of each question in the corpus. Their method improved the result of the system with a significant amount. Harksoo Kim has studied the problem of FAQ retrieval as it may be seen in [9], where he presented a trustly way of recovering FAQs by using a clustering of previous query logs. In fact, he also improved this first approach in [10] by employing latent semantic

`

analysis, and also by usinglatent term weights [11]. In [11] it is proposed the use of machine translation techniques for alignment of questions and answers of a FAQ corpus, with the aim of constructing a bilingual statistical dictionary which is further used for expanding the queries introduced in an information retrieval system. In [12] it is presented an approach for domain specific FAQ retrieval based on a concept named “independent aspect”. This concept basically consists of extracting terms and relationships by employing WordNet and Hownet which are then used in a mixture-based probabilistic model with the aim of analyzing queries and query-answer pairs by means of independent aspects.

1.4 Scope of the work

There has been little work on SMS-based FAQ Retrieval System for arbitrary topics due to the initial lack of a well defined business cases. The explosive growth in prevalence of affordable low-end mobile devices throughout the world has created a large market for mobile information services. Since mobile users in many parts of the world use low-end mobile devices with SMS as their primary data transport, SMS-based search has become a critical problem to address on the path to enabling SMS-based services.

In this thesis, I have presented an automated SMS-based search response system that is tailored to work across arbitrary topics. I found that a combination of simple Information Retrieval Algorithms in conjunction with simple optimization algorithms can provide reasonably accurate search responses for SMS queries. The use of N-Gram technique adds to the scalability of the software without adding much to its complexity. By incorporating various other techniques to the SMS based FAQ

`

Retrieval System; it can be further scaled to be used in different areas efficiently and conveniently. 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 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 algorithm, the idea of N-Gram count and its application in the algorithm. I also presented the formulas and equations that I have derived while deducing the algorithm. I also tried to compare my algorithm with other algorithms on the basis of time as well as space complexity. 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 era of globalization, information retrieval has become an important part of everybody’s life. Consequently, the methods that make information retrieval systems convenient become an interesting area of research. Today everyone is looking for a better and an easy way to access information. 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 part and parcel of everyday life. In the present scenario, Short Messaging Service (SMS) is one of the most popularly used services that provide information access to the people having mobile phones. In India alone, there are 811 million mobile subscribers and the number is still growing rapidly. So, 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 most popular data application on mobile phones is SMS text messaging. The cost of sending SMS and the easy access to the purchase of mobile phones have made instant messaging emerge as the preffered 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 [7]. 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 the medium to access information in the form of questioning and answering using SMS service of the mobile phones.

2.1.2 Introduction to SMS based Question Answering System Techniques

The most important data mining techniques are as follows:

2.1.3.1 Natural Language Processing Based:

In these types of systems, the user can send the queries through an SMS in a natural language. The system then analyzes and processes the query using natural language processing methods and then generates an answer. This answer is finally sent to the user. The main disadvantage of these systems is the complexity of the natural language processing algorithms used. Also, it may not always be possible to successfully analyze and understand the query since SMS language contains misspellings, non-standard abbreviations, transliterations, phonetic substitutions and omissions which make it difficult to build such automated systems around SMS technologies.

2.1.3.2 Human Intervention Based:

In these systems, the user can send their queries in any natural language. The queries reach human agents, who then understand the query, find an answer and send the reply to the user. Human intervention based systems exploit human communities to answer questions. These systems are interesting because they suggest similar questions resolved in the past. Other systems like Chacha and Askme use qualified

`

human experts to answer questions in a timely manner. Some businesses have recently allowed users to formulate queries in natural language using SMS. For example, many contact centers now allow customers to "text" their complaints and requests for information over SMS. Most of these contact center based services and other regular services like "AQA 63336" by Issuebits Ltd, GTIP by AlienPant Ltd., "Texperts" by Number UK Ltd. and "ChaCha" use human agents to understand the SMS text and respond to these SMS queries.

2.1.3.3 Information Retrieval Based:

The information retrieval based systems treat question answering as an information retrieval problem. They search a large corpus of text for specific text, phrases or paragraphs relevant to a given question. Today, a majority of SMS based information services require users to type specific codes to retrieve information. For example to get a duplicate bill for a specific month, says June, the user has to type DUPBILLJUN. The main disadvantage of this system is that it unnecessarily puts a constraint on the users who generally find it easy and intuitive to type in a "texting" language.

2.1.3.4 Frequently Asked Question Based:

In FAQ based question answering, where FAQs provide a ready-made database of questions and answers, the main task is to find the closest match for a question to retrieve the relevant answer. In these methods, a database of possible queries is maintained. Whenever a user sends an SMS asking a query, the database is searched for that particular query, or something close to it. The answer is returned to the user

`

when the appropriate match is found. For example, consider a system where 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 question-answer pair for a given query.

2.2 SMS Based FAQ Retrieval System

2.2.1 Problem Definition

This FAQ Retrieval system is designed to find a match from the given set of FAQs 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 such as short forms, abbreviations, spelling mistakes, phonetic spellings, transliterations etc. This makes SMS processing a tedious task. Our task is to find the best matching FAQ from the FAQ corpus with respect to the input SMS query with accuracy. In this task, we have a corpus of FAQs and answers from various domains that have been provided. The corpora of questions in the database are represented by Q. 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.

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

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 SMSes can be in any language. For example, FAQs are written in any one of these languages like English, Hindi Malyalam, etc and SMSes are coming may also belong to any one of these languages like English, Hindi, Malyalam, etc.

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

2.2.2 SMS Noise

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. 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, “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 turism 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, 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 objective of this chapter is to define various phases and functions used in the system. In preprocessing stage, the database is preprocessed in order to make 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

Preprocessing involves the following steps:

1. **Indexing:** 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 [10].
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 s_i in the SMS query, a similarity measure $\alpha(t, s_i)$ is defined that measures how closely the term t matches with the SMS token s_i. It is believed that the term t was a variant of s_i, if $\alpha(t, s_i) > 0$. 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:-

$$\boxed{Similarity\_Score(Q) = \sum_{i=1}^{n} \max\nolimits_{\mathrm{t \in Q\ and\ t \sim si}} \omega\ (\mathrm{t, si})}$$

Figure 2.1: Similarity Score

Consider a question Q $\in$ Q. For each token s_i 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.

`

2.3.2.1.1 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:-

$$\alpha(\mathrm{t},\mathrm{si}) = \begin{cases} \frac{LCS\ Ratio\ (t,si)}{Edit\ \ Distance\ \ (t,si)} & if\ t\ and\ si\ share \\ & same\ first\ character \\ 0 & othervise \end{cases}$$

Figure 2.2: Computation of Similarity Measure

Where $LCS\ Ratio(t,si) = \frac{Lengt\ h(LCS(t,si))}{lengt\ h(t)}$ 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".

2.3.2.1.2 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:

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

Figure 2.4: Weight Fucntion

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.

`

2.3.3 Proximity Score

Anwar D. Shaikh et al., December 2011 [2], stated that the relative position of words in a sentence plays an important role. They preferred a sentence over other sentences having same words but in different proximity using the proximity score. So, In order to find the best match, they have included the proximity of words in their score calculations. I have explained the working of our proximity search technique with an example given in figure 1 and figure 2.

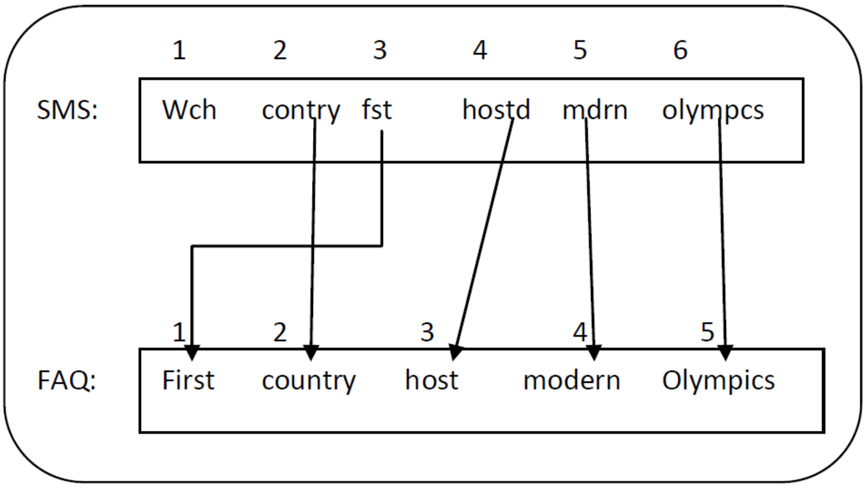

Figure 2.5: Mapping of SMS tokens with FAQ

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:

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

Figure 2.6: Proximity score

Where *totalFaqTokens* = number of tokens in FAQ

matchedToken = number of matched token of SMS in FAQ

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

Where n = number of matched adjacent pairs in SMS

Figure 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.

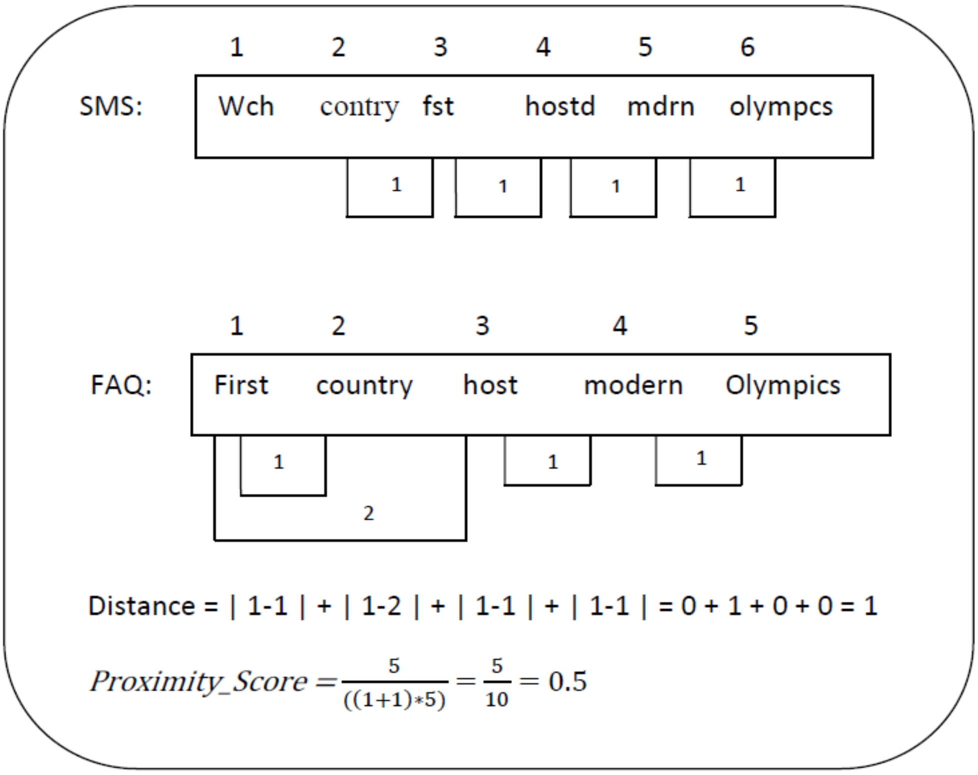
1 2 3 4 5 6
SMS:
Wch contry fst hostd mdrn olympcs
1
1
1
1
1 2 3 4 5
FAQ:
First country host modern Olympics
1
2
1
1
Distance = | 1-1 | + | 1-2 | + | 1-1 | + | 1-1 | = 0 + 1 + 0 + 0 = 1
$Proximity\_Score = \frac{5}{((1+1)*5)} = \frac{5}{10} = 0.5$

Figure 2.7: Calculation of Proximity Score

I have explained the significance of proximity search through an example. Let's say, the SMS query is "hw to buk ticket on internet". Base system will find the same score for both FAQ 1 "How to use Internet reservation facility to book the Ticket?" and FAQ 2 "How to cancel the Ticket which I have booked through the Internet?" because their system only considered the Similarity_Score. Improved version of the base system solved this problem by considering the Proximity_Score along with the Similarity_Score. This system selects FAQ 1 over FAQ 2 since Proximity_Score of FAQ 1 is greater than that of FAQ 2.

`

2.3.4 Length Score

Anwar D. Shaikh et al. [2], took the length of the FAQ as well as the SMS query into the account while calculating the score. This approach helped them to achieve better accuracy than earlier by assigning less priority to the long length questions and high priority to the short length questions. They have formularized the Length score is given below.

$$Length\_Score(Q) \quad = \frac{totalFAQToken\ \ -matc\ hedToken}{1+totalSMSToken\ \ -matc\ hedToken}$$

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

Figure 2.8: Length Score

Where *totalFAQToken*=total number of Tokens in FAQ question.

totalSMSToken= total number of Tokens in SMS query.

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

I have explained the significance of Length_Score through an example. Let's say, the SMS query is “**hw mch b4** d journey I cn **buk e-ticket**”. If we consider only Similarity_Score than the system will find the same score for both FAQ 1 “To **book** a tatkal **ticket** through Internet, **how much** days **before** it open?” and FAQ 2 “**How much** days **before** a **ticket** can be **book**ed?”.But if we consider the Length Score along with the Similarity Score than FAQ 2 will be preferred over FAQ 1 since Length Score of FAQ 2 is greater than the Length Score of FAQ 1.

`

2.4 Summary

The system consists of various stages and scoring function. If we combine all the scoring functions together i.e similarity score, proximity score and length score then it improves the results of the system with a significant amount. They have formulized the score as explained below:

$$\boxed{\begin{aligned} Score(Q) = W_1 * Similarity\_Score(Q) + W_2 * Proximity\_Score(Q) \\ + W_3 * Length\_Score(Q) \end{aligned}}$$

Figure 2.9: Combined Scoring Function

Where Q is the FAQ for which we are calculating the score. W_1, W_2 and W_3 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. W_1, W_2 and W_3 are adjusted such that their sum is 1.0 (or 100%). More than half weightage is given 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.

`

Chapter 3: Proposed N-Gram Driven Algorithm

The objective of this chapter is to introduce my approach to compute the score of the candidate FAQ by considering the frequency of N-Grams (Unigrams, Bigrams and Trigrams). This algorithm is based upon N-Grams frequencies and is completely different from previously existing algorithms.

In the fields of computational linguistics and probability, an 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.

3.1 Proposed Algorithm

In this work, I have presented a novel approach by developing an N-gram count based algorithm that takes the count of various N-grams (monograms, bigrams, trigrams, etc.) into the account in order to calculate the score of the questions in the corpus. In this way, we can further improve the accuracy of the SMS based FAQ system significantly by refining the results of the system using N-gram count based scoring.

`

3.1.1 N-Grams count

As I have explained in earlier section that the system view SMS as a sequence of tokens. I have denoted these tokens as monograms. In this way, we can denote any two adjacent tokens as bigram and three consecutive tokens as trigram and so on. For example, For given SMS query, “hw 2 bk ttkal tkt on net”, system first remove the stop words and consider “hw bk ttkal tkt net” for further calculation of score. The matched terms from the dictionary would be: how, use, internet, tatkal, and reservation. The N-grams for above example would be:

- *Monograms* :- { how, use, internet, tatkal, reservation }
- *Bigrams*:- { [how, use], [use, internet], [internet, tatkal], [tatkal, reservation] }
- *Trigrams*:- { [how, use, internet], [use, internet, tatkal], [internet, tatkal, reservation] }
- And so on…

N-grams count corresponding to a question in the FAQ corpus is the count of total number of N-grams present in the SMS that matches with the N-grams in the question. For example, the FAQ is “How to use the internet reservation facility to book Tatkal ticket?” and after removing the stop words it would become “How use internet reservation facility book Tatkal ticket”. The N-gram counts corresponding to given FAQ are:

- *Monograms* :- { how, use, internet, reservation, facility, book, tatkal, ticket }
- *Bigrams*:- { [how, use], [use, internet], [internet, reservation], [reservation, facility], [facility, book], [book, tatkal], [tatkal, ticket] }

`

- *Trigrams*:- { [how, use, internet], [use, internet, reservation] , [internet, reservation, facility], [reservation, facility, book], [facility, book, tatkal], [book, tatkal, ticket]}
- And so on…

In order to measure the counts for each N-gram we can simply compare the SMS N-gram set with the corresponding FAQ N-gram set. From the above example, we can conclude:

- Monogram count = 5;
- Bigram count = 2;
- Trigram count = 1;
- And so on…

3.1.2 N-Grams Algorithm

After the calculation of N-gram counts corresponding to a question in the corpus, I have calculated the score of the question using these counts. As score is based upon the counts of N-grams, therefore I refer this score as N-gram_Score. In order to calculate the N-gram_Score for a given question I have deduced the following equation:

$$\boxed{N\_gram\_Score(Q) \; = \; \frac{(\,\mathrm{L1} \; + \; \mathrm{L2}^{\,2} + \; \mathrm{L3}^{\,3} + \; ... + \mathrm{LN}^{\mathrm{N}})}{(\,\mathrm{T} \; + \; [\mathrm{T}-1]^{2} + \; [\mathrm{T}-2]^{3} \; + \; ... + [\mathrm{T}-\mathrm{N}]^{\mathrm{N}})}}$$

Figure 3.1: N-Gram Score

`

L1, L2 and L3 are the monograms (matchedToken), bigrams, and trigrams counts respectively corresponding to a given question Q, LN is theN-gram count,. T is the total number of tokens in the SMS query having a matching FAQ term. As SMS queries are short in nature, we could expect bigrams and trigrams occurrences in a SMS corresponding to a FAQ.

After N-gram score into consideration, I define the new scoring function for a question Q as:

$$Score(Q) = W_1 * Similarity\_Score(Q) + W_2 * Proximity\_Score(Q) + W_3 * N\_Gram\_Score(Q)$$

Figure3.2: New Combined Scoring Function

Where Q is the FAQ for which we are calculating the score. W_1, W_2 and W_3 are real valued weights. Their values determine the weights of Similarity Score, Proximity Score and N-gram Score from the overall score of the FAQ question. W_1, W_2 and W_3 are adjusted such that their sum is 1.0 (or 100%).This score function evaluates the score of the question Q under consideration in order to find the best match. If the score is above certain predefined threshold, then it is considered as match. Value of threshold can be selected based on the system requirement. Threshold value impacts the In Domain and Out Domain queries in opposite manner. Decreasing the threshold may improve the In Domain results and degrade the out domain results and vice-versa.

`

3.2 Advantages

The two core advantages of N-Gram Models (and algorithms that use them) are relative simplicity and the ability to scale up – by simply increasing *n* a model can be used to store more context with a well-understood space–time tradeoff, enabling small experiments to scale up very efficiently. Some of the other advantages are:

- Language independence and simplicity: Character level N-gram models are applicable to any language, and even to non-language sequences such as music or gene sequences.
- Robustness: Character level N-gram models are relatively insensitive to spelling variations and errors, particularly in comparison to word features.
- Completeness: The vocabulary of character tokens is much smaller than any word vocabulary and is normally known in advance. Therefore, the problem of sparse of data is much less serious in character N-gram models of the same order.

3.3 Summary

In this chapter I have presented a novel approach based on N-gram counts to improve the accuracy of the SMS based QA system. Since the SMS are short in nature, I have considered occurrence of bigrams and trigrams. This approach is an effective and simple approach that potentially improves the accuracy of the SMS based FAQ Retrieval Syastem.

`

Chapter 4: Implementation and Experimental Results

4.1 Environmental Setup

I have used the following configuration while finding the experimental results

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 the FIRE in order to test the system accuracy. SMS queries were categorized as in domain if the query belongs to the Domains present in the dataset and out domain otherwise. We have considered 497 in domain and 777 out domain SMS queries for evaluation.

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

In current experiments I have considered the similarity, proximity and N-Gram occurrences as the attributes for evaluation. Four different experiments were conducted. In first experiment I have only considered the Similarity between SMS token and FAQ terms, In Second experiment Proximity score along with Similarity measure was considered and in Third experiment N-Gram score along with Similarity measure was considered. Finally I have observed that when I combine our N-gram approach with the Proximity and Similarity score than the accuracy of the system improved by a great extent.

I have repeated these four experiment three times by taking three different score thresholds. This score threshold determines whether to consider the match of a FAQ with SMS or not. In this way, I observed the percentage correctness of queries in each case. For a given SMS query accuracy was tested based on the top three FAQ returned by the system, if the required FAQ is present in the top three then the answer is

`

marked as true. Table 1 to Table 4 shows the results of our four experiments repeated with three different score thresholds.

I have evaluated our system for three different threshold values T1=0.275, T2=0.300 and T3=0.325. Table 2, 3 AND 4, shows the results for the In-Domain and Out-Domain queries obtained using different threshold values. Figure 1, shows the overall results under different threshold values. From above experiments it is observed that the threshold T2 provides more In-Domain accuracy while threshold T2 provides more Out-Domain accuracy.

Table 4.1: Number of SMS Queries used for Experiments

	In Domain	*Out Domain*	*Total*
Total Queries	497	777	1274

Table 4.2: Experiment Results for threshold T1.

	In Domain	Out Domain	Total Correct
Similarity	394	223	617
Similarity+Proximity	380	388	768
Similarity+N Gram	346	377	723
Smilarity + Proximity + N Gram	343	539	882

`

Table 4.3: Experiment Results for threshold T2.

	In Domain	Out Domain	Total Correct
Similarity	390	257	647
Similarity+Proximity	369	441	810
Similarity+N Gram	364	430	794
Smilarity + Proximity + N Gram	344	561	905

Table 4.4: Experiment Results for threshold T3.

	In Domain	Out Domain	Total Correct
Similarity	386	293	679
Similarity+Proximity	366	487	853
Similarity+N Gram	369	449	818
Smilarity + Proximity + N Gram	322	602	924

`

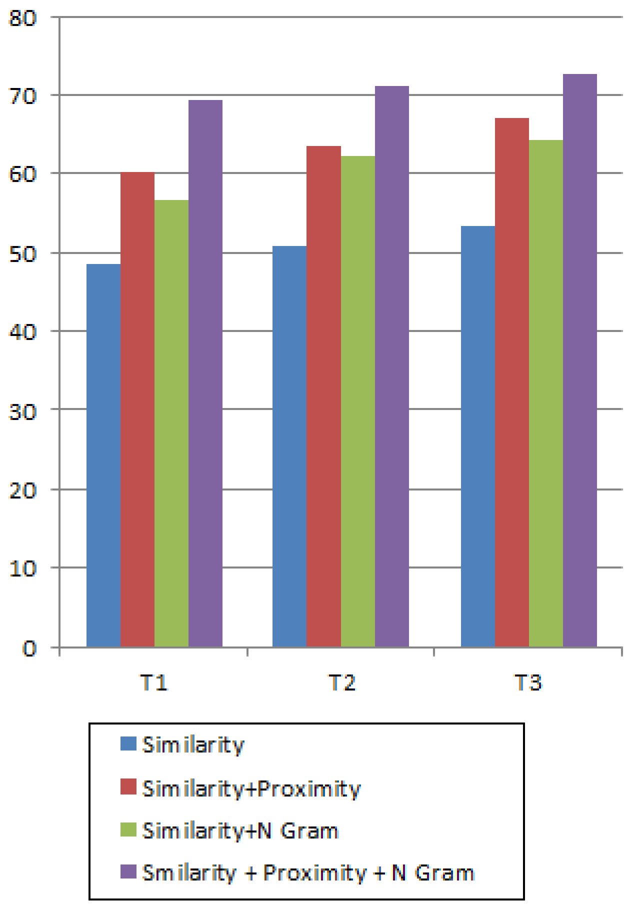

Figure 4.3: Results Graph

4.4 Summary

In this chapter I have introduced the environmental setup which I have used while making the experimental results. In addition to this I have also explain the types of dataset which I have used and explain them in detail. Various experiments were conducted to test the accuracy of various matching techniques. From the experimental results, I can conclude that this approach is able to significantly outperform the previous state-of-the-arts SMS based QA system, particularly in case of out domain queries, the results are best accurate.

`

Chapter 5: Conclusion & Future Scope

5.1 Conclusion

There has been little work on SMS-based search for arbitrary topics due tothe initial lack of a well defined business cases. The explosive growth in prevalence of affordable low-end mobile devices throughout the world has created a large market for mobile information services. Since mobile users in many parts of the world use low-end mobile devices with SMS as their primary data transport, therefore, SMS-based search becomes a critical problem to address on the path to enabling SMS-based services.

In this Thesis, I have presented an automated SMS-based search response system that is tailored to work across arbitrary topics. The use of N-Gram technique adds to the scalability of the software without adding much to its complexity. I have presented a novel approach based on N-gram count to improve the accuracy of the SMS based QA system. Since the SMS are short in nature, I have considered occurrence of bigrams and trigrams to improve the accuracy of the system. Various experiments were conducted to test the accuracy of various matching techniques. From the experimental results, we can conclude that this approach is able to significantly outperform the current state-of-the-art SMS based QA system, particularly in case of out domain queries the results are more accurate.

Using queries across arbitrary topics from a real world SMS question/answering service with human-in-the-loop responses, I show that this software is able to answer up to 75.527% of the queries in the above test set. Although more powerful IR and

`

NLP techniques are bound to improve performance, this work represents a foray into an open and 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

`

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 Extend our work from monolingual (English) to multilingual (English, Hindi, Malayalam, Tamil, etc)

`

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] Anwar Shaikh, Rajiv Ratn Shah, Mukul Jain, Mukul Rawat, Manoj Kumar, “Improving accuracy of SMS based FAQ retrieval,” Proc. Forum for Information Retrieval Evaluation (FIRE 2011), to be published in Springer LNCS unpublished.

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

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

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

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

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

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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

[19] 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.

[20] 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 java.util.StringTokenizer;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.WhitespaceAnalyzer;
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.queryParser.QueryParser;
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("&quot;"," ").toLowerCase().trim();
              answer=answer.replace('"',' ').replace("&quot;","
").toLowerCase().trim();
              question =question.replace('?',' ').replace('*','
').replace('<',' ').replace('>',' ').replace('%',' ').replace('&', '
').replace(';',' ').replace('-',' ').replace("'"," ").replace('[','
').replace('{',' ').replace(']',' ').replace('}',' ').replace('!','
```

```
`
').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', '
').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', '
').trim().toLowerCase();
              answer=answer.replace('?',' ').replace('*',' ').replace('<','
').replace('>',' ').replace('%',' ').replace('&', ' ').replace(';','
').replace('-',' ').replace("'"," ").replace('[',' ').replace('{','
').replace(']',' ').replace('}',' ').replace('!',' ').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
       {
                   term =term.replace('?',' ').replace('*','
').replace('<',' ').replace('>',' ').replace('%',' ').replace('&', '
').replace(';',' ').replace('-',' ').replace("'"," ").replace('[','
').replace('{',' ').replace(']',' ').replace('}',' ').replace('!','
').replace('(', ' ').replace('.', ' ').replace(')', ' ').replace(',', '
').replace('/', ' ').replace('#', ' ').replace(':', ' ').replace('"', '
').trim().toLowerCase();
                   //do not replace comma in synset
                   synset=synset.replace('?',' ').replace('*','
').replace('<',' ').replace('>',' ').replace('%',' ').replace('&', '
').replace(';',' ').replace('-',' ').replace("'"," ").replace('[','
').replace('{',' ').replace(']',' ').replace('/', ' ').replace('#', '
').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
                          term=replaceSpecialChar(term);
                          synset=replaceSpecialChar(synset);
                          if(term.equalsIgnoreCase("sum"))
                                {final int a=3;}
                   if(term.length()>0)
                   {
                                //create document for question
                                Document doc=new Document();
                                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.commit();
            indexWriter.optimize();
            indexWriter.close();
        }
            String replaceSpecialChar(String str)
            {
                return str.replace('/',' ').replace('\\','
').replace('~',' ').replace('?',' ').replace('$',' ').replace('<','
').replace('>',' ').replace('%',' ').replace('&', ' ').replace(';','
').replace('-',' ').replace("'","").replace('[',' ').replace('{','
').replace(']',' ').replace('}',' ').replace('!',' ').replace('(', '
').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', '
').replace('#', ' ').replace(':', ' ').replace('"', ' ').replace('+', '
').trim().toLowerCase();
            }
        public void removeDuplicates() throws ParseException, IOException
            {
                File tempIndex = new
File(System.getProperty("DomainIndexDirCopy"));
                Directory fsDir = FSDirectory.open(tempIndex);
                            IndexReader tmpReader =
IndexReader.open(fsDir);
                IndexSearcher tmpSearcher = new
IndexSearcher(tmpReader);
                Analyzer analyzer = new ~~WhitespaceAnalyzer~~();
                QueryParser parser  = new
QueryParser(Version.LUCENE_30,"term",analyzer);
                for(int i=0;i<tmpSearcher.maxDoc();i++)
                            {
                    String domainTerm,synSet;
                    domainTerm=tmpSearcher.doc(i).get("term");
                    synSet=tmpSearcher.doc(i).get("synset");
                    Query q;
                    if(domainTerm.equalsIgnoreCase("sum"))
                    { int a=3;
                     a++;}
                        if(synSet.length()==0)
                        {
                            q =
parser.parse("term:"+domainTerm);
                        }
                        else
                        {
                            q =
parser.parse("term:"+domainTerm+" AND synset:"+synSet);
                        indexWriter.deleteDocuments(q);
                        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.WhitespaceAnalyzer;
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.IDF;
import similarity.similarity;
import smsProcessing.TextPreprocessing;
public class FAQ_Search {
        private static final double WEIGHT_THRESHOLD = 0.6;
         IndexSearcher searcher;
         QueryParser parser;
        int maxHits=20000;
         double numDocs;
        boolean debug=false;
        private  static final double SYNONYM_WEIGHT_THRESHOLD=0.75;
        //static IDF idf;
        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();
        }
        public DictionarySearchResult searchDomainDictionary(String token)
throws ParseException, IOException
        {
                 IDF idfObject = new IDF(System.getProperty("FAQIndexPath"));
                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;
        if(token.trim().getBytes("UTF-8")[0]!=-17)
            q =
parser.parse("term:"+token.trim().charAt(0)+"*");//search for similar first
character
        else
            q =
parser.parse("term:"+token.trim().charAt(1)+"*");//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");
            String syn=d.get("synset");
            if(similarity.lcs(token,domainTerm)>1)
            {
                //calculate weight
                double
weight=idfObject.calcWeight(token,domainTerm);
                if(weight>WEIGHT_THRESHOLD)
                {
                    result.addNewTerm(domainTerm, weight);
                    if(debug)
                        System.out.println(n+"]
"+domainTerm+"="+syn);
                }
            }
        }
        return result;
    }
    public SynonymSearchResult searchSynonymDictionary(String token,
HashTable hashTable) throws ParseException, IOException
    {
        if(debug)
        System.out.println("Searching Synonym 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(':', ' ').replace('"', '
').trim().toLowerCase();
        //search DOmain dictionary
        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*3);//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;
            //get synset tokens separated by Comma
            StringTokenizer que_tok = new StringTokenizer(synset,"
");
            int total=que_tok.countTokens();
            double alpha,weight;
            for(int i=0;i<total;i++)
            {
                String synonym=que_tok.nextToken();
                if(synonym.equalsIgnoreCase(domainTerm))
                    continue;
                //separate out the tokens
                if(similarity.lcs(token,synonym)>2)
                {
    alpha=similarity.similarityMeasure(synonym,token);
        weight=alpha*hashTable.getValueFromHT(domainTerm);
                    if(weight>SYNONYM_WEIGHT_THRESHOLD)
                    {
                        if(debug)
            System.out.println(domainTerm+"="+synonym);
                        result.addNewTerm(domainTerm,
weight,synonym);
                    }
                }
            }
        }
        return result;
    }
    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('/', ' ').replace('#', '
').replace(':', ' ').replace('"', ' ').trim().toLowerCase();
        Query q = parser.parse("question:"+query+" OR answer:"+query);
        //Query q = parser.parse("question:"+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);
        //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");
            String faq_text = d.get("question");
    if(System.getProperty("removeStopWords").equals("true"))
            {
            if(System.getProperty("smsLanguage").equals("Hindi"))
        faq_text=TextPreprocessing.removeHinidStopWords(faq_text);
            else
if(System.getProperty("smsLanguage").equals("English"))
    faq_text=TextPreprocessing.removeEnglishStopWords(faq_text);
            }
            result.addResult(faq_id,faq_text);
            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 String getQuestionByFaqID(String faqID) throws IOException,
ParseException
        {
                    /** NOTE: the ANALYZER is CHANGED **/
                    IndexSearcher searcher;
                    QueryParser parser;
                    File indexDir = new
File(System.getProperty("FAQIndexPath"));
                    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.toUpperCase());
                    try{
                    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;
                    }
                    catch(Exception e)
                    {
                          return "NOT FOUND";
                    }
        }
        public static void main(String[] args)
        {
              System.setProperty("smsLanguage","English");
                    System.setProperty("considerSynonym","true");
                    System.setProperty("removeStopWords","true");
              System.out.println("&quot;test&quot;".replace("&quot;", ""));
              try {
                     FAQ_Search a = new
FAQ_Search(System.getProperty("DomainIndexPath"));
             } 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 faqIndexing.FAQ_Search;
import smsProcessing.faqRetrieval;
import java.util.StringTokenizer;
public class smsparser extends DefaultHandler{
      static FAQ_Search srch;
      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;
    boolean debug =true;
    /* 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(){
        properties.load("English");
        if( System.getProperty("OUTPUT_NOTHING").equalsIgnoreCase("true"))
              debug=false;
             System.out.println(" Object Created ");
             try {
                   srch=new FAQ_Search(System.getProperty("FAQIndexDir"));
             } catch (Exception e1) {
             }
      }
/*
 * 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(System.getProperty("smsLanguage"))) {
                  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
                     if(System.getProperty("smsLanguage").equals("Hindi"))
                            test=queryid.startsWith("HIN");
                     else
if(System.getProperty("smsLanguage").equals("English"))
                            test=queryid.startsWith("ENG");
                     else
                            test=queryid.startsWith("MAL");
                     if(test == true ){
                    String result = "";
                    //Here we are calling the naive algorithm
                    try {
                                 result =
faqRetrieval.getMatchingQuestion(queryid,text);
                           } catch (Exception e){  System.out.println(e); }
                            //print the question id and the question
                           StringTokenizer st = new
StringTokenizer(result,",");
                           int totalResults = st.countTokens();
                           if(debug)
                           {
                           System.out.println("Expected
Result:"+matchenglish);
                           System.out.println("Generated Result:"+result);
                           }
                    //The compute score algorithm is called here
                    score=computescore(result,matchenglish);
                    totalSMScount++;
                    if(debug)
                           {
                    System.out.println(queryid+":"+text+(score==0?"
:[False]":"   :[True]"));
                    System.out.println("Correct="+sum+"\t
Total="+totalSMScount);
`

       //System.out.println("**********************************************
*****************");
                           }
                     }
              }
            }
           //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;
                  }
               }

            /* generate output as per given format */
            /* 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){
              int expLen,actLen;
              String x,y;
              StringTokenizer generated = new StringTokenizer(result,",");
              StringTokenizer expected = new
StringTokenizer(matchenglish,",");
              actLen = generated.countTokens();
              expLen = expected.countTokens();
              // expected result is NONE but actual is not
              if(expLen==0 && actLen!=0)
                     return 0;
              //number of results are not equal
              boolean yMatchesX = false;
              //expected result
              y=expected.nextToken().toLowerCase();
              for(int i=0;i<actLen;i++)
              {
                     //check if the current expected result is present in
the actual result
                     x=generated.nextToken().toLowerCase();
                     //match found
                     if(x.equals(y))
                                  { yMatchesX=true; sum++; break ;}

              }
```

`

```
                    if(yMatchesX==false)
                           return 0;
              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[]){
              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;
               source = new
InputSource(System.getProperty("SMSInputPath"));
              xmlReader.parse(source);
            } catch (Exception e) {
                  System.err.println(e);
                  System.exit(1);
            }

             }

}
```

`

CandidateSet.java

```
package smsProcessing;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.StringTokenizer;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.store.LockObtainFailedException;
import similarity.IDF;
import similarity.similarity;
import faqIndexing.FAQ_Search;
import faqIndexing.HashTable;
import faqIndexing.QuestionSearchResult;
import faqIndexing.SynonymSearchResult;
/*
 * Consider proximity for scoring
 * sms tokens -> T1
 * Matching tokens -> M
 * Score = score +   (T1-M)/(T2-M)*(1/[sum of distance between faq
terms+1])* 100
 *
 */
public class CandidateSet {
        private static final boolean debug = false;
    private static final boolean coding =false;
        //TODO for English it is 0.9
        //TODO for hindi it is 0.8
        //private static final double SCORE_THRESHOLD =0.9;;
        //private static final double SCORE_THRESHOLD =0.8;
        private static double SCORE_THRESHOLD;
        //TODO for english SIMILARITY 0.17
        //TODO for HINDI SIMILARITY 0.09
        private static double SIMILARITY_THRESHOLD;
        //private static final double SIMILARITY_THRESHOLD = 0.09;
        String[] faq_id;
        int[] faq_numberOfTokens;
        String[][] faq_term;
        String[][] faq_term_synonym;
        double[] score;
        String[] smsTokens;
        String smsText;
        int matchedFAQTokenPosition[];
        int smsProximityArray[];
        int totalSmsTokens;
        int totalFaqTokens;
        int matchedToken;
        String smsID;
        int count;
        static int MAXQUESTIONS = 3000;
        FAQ_Search srch;
        //Replaced IDF call with HASHTABLE
        //static IDF idf;
        HashTable htable;
```

```
`
        //Constructor to initialize strings
        public CandidateSet(String sms_id,String smsText,int
smsProximityArrayArg[],HashTable hashTable) throws CorruptIndexException,
LockObtainFailedException, IOException
        {
                htable=hashTable;
                faq_id=new String[MAXQUESTIONS];
                faq_term=new String[MAXQUESTIONS][30];
                faq_term_synonym=new String[MAXQUESTIONS][30];
                faq_numberOfTokens=new int[MAXQUESTIONS];
                count=0;
                smsID=sms_id;
                srch=new FAQ_Search(System.getProperty("FAQIndexDir"));
                //Replaced IDF call with HASHTABLE
                //idf=new IDF(System.getProperty("FAQIndexDir"));
                //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();
                }
                initThreshold();
                this.smsText = smsText;
                smsProximityArray = smsProximityArrayArg ;
                if(debug)
                for(int i=0;i<totalSmsTokens;i++)
                {
                        System.out.println(i+"="+smsProximityArray[i]);
                }
        }
        void initThreshold()
        {

                 // SCORE_THRESHOLD = 0.9;
                 //formula 10 & 16
                 //--- this is latest
                //SCORE_THRESHOLD = 0.5;

                //formula 26
                //SCORE_THRESHOLD = 0.1;
                SCORE_THRESHOLD =
Float.parseFloat(System.getProperty("SCORE_THRESHOLD")) ;

                //-------------------------------------- SIMILARITY_THRESHOLD
---------------------------
                 //formula 10 to 19
                 //SIMILARITY_THRESHOLD = 0.2;

                  //formula 22
                  //SIMILARITY_THRESHOLD = 0.17;
                SIMILARITY_THRESHOLD =
Float.parseFloat(System.getProperty("SIMILARITY_THRESHOLD")) ;
        }
        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;
        faq_numberOfTokens[count]=0;
        count++;
    }
    void addCandidate(String que_id, String domain_term, String synonym)
    {
        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].equalsIgnoreCase(que_id))
            {
                for(int j=0;j<faq_numberOfTokens[i];j++)
                {

    if(faq_term[i][j].equalsIgnoreCase(domain_term)&&faq_term_synonym[i]
[j].equalsIgnoreCase(synonym) )
                    {
                        return;
                    }
                }
                //add new term and synonym
                faq_term[i][faq_numberOfTokens[i]]=domain_term;

    faq_term_synonym[i][faq_numberOfTokens[i]]=synonym;

                faq_numberOfTokens[i]++;
                return;
            }
        }
                //add new FAQ ID, term and Synonym
                faq_id[count]=que_id;
    faq_term[count][faq_numberOfTokens[count]]=domain_term;
        faq_term_synonym[count][faq_numberOfTokens[count]]=synonym;
```

`

```
                faq_numberOfTokens[count]++;
                count++;
    }
    public void generateCandidateSet(SynonymSearchResult 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++)
        {
          if(LIST.getSynonymAt(i) == null)
          {
    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));
            }
          }
          else
          {
srchResult=srch.searchQuestionAnswer(LIST.getTermAt(i));
                // add all search results to the Candidate set
                for(int j=0;j<srchResult.getCount();j++)
                {
                    //String faq;
    //faq=srchResult.getFaqTextAt(j).replace(LIST.getTermAt(i),
LIST.getSynonymAt(i));
    addCandidate(srchResult.getFaqIdAt(j),LIST.getTermAt(i),LIST.getSyno
nymAt(i));
                }
          }
        }
    }
    String replaceSpecialChar(String str)
    {
        return str.replace('?',' ').replace('$',' ').replace('<','
').replace('>',' ').replace('%',' ').replace('&', ' ').replace(';','
').replace('-',' ').replace("'","").replace('[',' ').replace('{','
').replace(']',' ').replace('}',' ').replace('!',' ').replace('(', '
').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', '
').replace('#', ' ').replace(':', ' ').replace('"', '
').trim().toLowerCase();
    }
    double calculateScore(int faqID) throws IOException, ParseException
    {
        String faq=srch.getQuestionByFaqID(this.faq_id[faqID]);
        faq=replaceSpecialChar(faq);
        if(System.getProperty("removeStopWords").equals("true"))
        {
        if(System.getProperty("smsLanguage").equals("Hindi"))
            faq=TextPreprocessing.removeHinidStopWords(faq);
        else if(System.getProperty("smsLanguage").equals("English"))
```

```
`
            faq=TextPreprocessing.removeEnglishStopWords(faq);
        }
        StringTokenizer faq_tok = new StringTokenizer(faq," ");
        totalFaqTokens=faq_tok.countTokens();
        String[] faq_tokens=new String[totalFaqTokens];
        if(coding)
        {
        System.out.println("SMS:\t"+smsText);
        System.out.println("FAQ:\t"+faq);
        }
        //copy the FAQ tokens
        for(int i=0;i<totalFaqTokens;i++)
        {
    faq_tokens[i]=replaceSpecialChar(faq_tok.nextToken());
        }
        double score=0;
        String smsToken;
        String faqToken;
        matchedToken=0;
        //int matchedFAQTokenPosition[]=new int[totalSmsTokens];
        matchedFAQTokenPosition=new int[totalSmsTokens];
        //1.For each sms token
        for(int i=0;i<totalSmsTokens;i++)
        {
            smsToken = smsTokens[i];
            matchedFAQTokenPosition[i]=-1;
            smsToken = replaceSpecialChar(smsToken);
            double maxWeight=0;
            double weight = 0;
            boolean matchFound=false;
            //2. compare it with each Term present in the question.
            //   get the maximum weight of the term and smsToken
            int j;
            double alpha;
            double IDF = 0;
            for(j=0;j<totalFaqTokens;j++)
            {
                faqToken=faq_tokens[j];

                faqToken = replaceSpecialChar(faqToken);

                boolean synonymExists=false;
                String synonym = null;
              //check if synonym exists
                for(int k=0;k<faq_numberOfTokens[faqID];k++)
                {
    if(faqToken.equalsIgnoreCase(replaceSpecialChar(faq_term[faqID][k]))
)
                    {
                        synonymExists=true;
                        synonym =
faq_term_synonym[faqID][k];
                    }
                }
                if(synonymExists)
`
                                {
        alpha=similarity.similarityMeasure(synonym,smsToken);
                                }
                                else
                                {
        alpha=similarity.similarityMeasure(faqToken,smsToken);
                                }
                                if(alpha>SIMILARITY_THRESHOLD)
                                {
                                          //TODO - change this formula
                                          //Replaced IDF call with HASHTABLE
                                          //double IDF1=idf.getIDF(faqToken);
                                          IDF= htable.getValueFromHT(faqToken);
                                          //if(IDF1!=IDF)
                                        //        System.err.println("MISMATCH-
"+IDF+","+IDF1);
                                          weight=alpha*IDF;
                                          //weight=alpha;
                                          if(weight>maxWeight)
                                          {
                                                  maxWeight=weight;
                                                  matchedFAQTokenPosition[i]=j;//
store the matched FAQ's position
                                          }
                                          matchFound=true;
                        }
                        score=score+maxWeight;
                        if(matchFound)
                                {
                                        matchedToken++;
                                }
                }
                if(matchedToken>0)
                        matchedToken--;
                //print the Token positions
                if(debug)
                {
                        System.out.println("SMS:\t"+smsText);
                        System.out.println("FAQ:\t"+faq);
                        System.out.println("matched:\t"+matchedToken);
                        for(int i=0;i<totalSmsTokens;i++)
                        {
        System.out.println(matchedFAQTokenPosition[i]);
                        }
                }
                // Forumula 1 : consider length score
                // double NormalizedScore =
2.0*(score*matchedToken/(totalSmsTokens*totalSmsTokens)) -
0.2*(totalFaqTokens-matchedToken)/(totalSmsTokens-matchedToken);
                // Forumula 2 : No length score
                double NormalizedScore =
2.0*(score*matchedToken/(totalSmsTokens*totalSmsTokens));
                //System.out.println("Score:"+NormalizedScore);
                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];
        double proximityScore = 0;//[]=new double[count];
        int checkProximity=0;
    if(System.getProperty("CONSIDER_PROXIMITY").equalsIgnoreCase("true")
)
                checkProximity=1;
            else
                checkProximity=0;
        //calculate score of all candidate questions
        double similarityScore=0;
        for(int i=0;i<count;i++)
        {
            similarityScore=this.calculateScore(i)/1.5;
            if( checkProximity==1 ||
System.getProperty("BIAGRAM_TRIGRAM").equalsIgnoreCase("true"))
            {
                //if there is only 1 token then proximty is zero
                if(matchedToken>2)
                {
    proximityScore=this.calculateProximity(checkProximity);
                }
                else
                    proximityScore=0;
```

`

```
                if(debug)
                System.out.println("SCORE:"+similarityScore);
                if(debug)
                System.out.println("PROX:"+proximityScore);

                //give 50% weight age to SCORE and 50% to
PROXIMITY
            }
            score[i]=similarityScore*0.5 + proximityScore*0.5;
            if(debug)
            System.out.println("Final SCORE:"+score[i]);
        }
        //sort the questions
        this.sortByScore();
        String result="NONE";
        //show the matching questions
        //this.showMatchingQuestions();
        result=generateFIREOutputFormat();
        if(debug)
        System.out.println("Result = "+result);
        return result;
    }
    private double calculateProximity(int checkProximity) {
        // TODO Auto-generated method stub
        if(debug)
        {
            System.out.println("Faq tokens:"+totalFaqTokens);
            System.out.println("Matched:"+ matchedToken);
            System.out.println("SMS distance:");
            for(int i=0;i<totalSmsTokens;i++)
            {
    System.out.println(this.smsProximityArray[i]);
            }
            System.out.println("Matching FAQ token position:");
            for(int i=0;i<totalSmsTokens;i++)
            {
    System.out.println(this.matchedFAQTokenPosition[i]);
            }
        }
        //find the total distance
        int faqProximity[]=new int[matchedToken+1];
        int smsProximity[]=new int[matchedToken+1];
        int f1=0,f2=0;
        int index=0;
        //store the distance into another array
        for(int i=0;i<totalSmsTokens;i++)
        {
            if(matchedFAQTokenPosition[i]!=-1)
            {
    faqProximity[index]=matchedFAQTokenPosition[i];
                smsProximity[index]=smsProximityArray[i];
                index++;
            }
        }
        //find the actual distance
        int distance=0;
```

```
`
             double proximityScore=0;
             if(checkProximity==1)
             {
                    for(int i=0;i<matchedToken-1;i++)
                    {
                          int currentDistance=Math.abs(
(faqProximity[i+1] - faqProximity[i]) - (smsProximity[i+1] -
smsProximity[i]));
                          distance = distance + currentDistance;
                    }
                    /*
                     * Consider proximity for scoring
                     * sms tokens -> T1
                     * FAQ tokens -> T2
                     * Matching tokens -> M
                     * Proximity Score =   (T1-M)/(T2-M)*(1/[sum of
distance between faq terms+1])* 100
                     *
                     */
                     //FORMULA 10 this.SCORE_THRESHOLD = 5
                     proximityScore =1.0* matchedToken / ((distance +1)*
totalFaqTokens ) ;
                     f1=1;
             };
             double nGramScore=0;
             double totalScore=0;
             //Consider Bigram and Trigram Occurance
             if(System.getProperty("BIAGRAM_TRIGRAM").equals("true"))
             {
                    int bigrams=0,trigrams=0;
                    bigrams=bigramOccurance(faqProximity,smsProximity);
                    trigrams=trigramOccurance(faqProximity,smsProximity);

                    //new formula 12-feb-2012
                    nGramScore = ( matchedToken+ Math.pow(bigrams, 2)+
Math.pow(trigrams, 3) ) / ( totalSmsTokens + Math.pow(totalSmsTokens-1, 2)
+ Math.pow(totalSmsTokens-2, 3));
                    f2=1;
             }
             else
                    totalScore=proximityScore+nGramScore;

             if(f1==1&&f2==1)
                    totalScore/=2;
             //print array
             if(debug)
             {
                    for(int i=0;i<matchedToken;i++)
                    {
                          System.out.println("faq = "+faqProximity[i]);
                          System.out.println("sms = "+smsProximity[i]);
                    }

                    System.out.println("Distance = "+distance);
                    System.out.println("Proximity Score =
"+proximityScore);
```

```
`
        }
        return totalScore;
    }
    private int bigramOccurance(int faqProximity[],int smsProximity[])
    {
        //check for bigrams
        int totalBiagrams=0;
        // store biagram starting position in array
        for(int i=0;i<matchedToken-1;i++)
        {
            if( (faqProximity[i+1]-faqProximity[i])==1 &&
(smsProximity[i+1] - smsProximity[i])==1)
            {
                totalBiagrams++;
            }
        }
        return totalBiagrams;
    }
    private int trigramOccurance(int faqProximity[],int smsProximity[])
    {
        //check for tri grams
        int totalTrigrams=0;

        // store trigrams  starting position in array
        for(int i=0;i<matchedToken-2;i++)
        {
            if( (faqProximity[i+1]-faqProximity[i])==1 &&
(faqProximity[i+2]-faqProximity[i+1])==1 && (smsProximity[i+1] -
smsProximity[i])==1 && (smsProximity[i+2] - smsProximity[i+1])==1)
            {
                totalTrigrams++;
            }
        }
        return totalTrigrams;
    }
    private String generateFIREOutputFormat() throws IOException,
ParseException
    {
        int top=3;
        if(count<3)
            top=count;
        String output="";
        String result="";
        int higerScore=0;
        DecimalFormat df = new DecimalFormat("#.###############");
        if(top>0)
        for(int i=0;i<top;i++)
        {
                if(score[i]<SCORE_THRESHOLD)
                    break;
                output=output+","+df.format(score[i]);
                result=result+","+faq_id[i].trim();
             higerScore++;
        }
        else
        {
```

```
`
                        output=output+",NULL";
                        result="NONE";
                }
                if(top>0 && higerScore==0)
                {
                        output=output+",NULL";
                        result="NONE";
                }
        if(!System.getProperty("OUTPUT_NOTHING").equalsIgnoreCase("true"))
                        System.out.println("SCORE:  "+output);
                return result;
        }
        private void showMatchingQuestions() throws IOException,
ParseException
        {
                System.out.println();
                System.out.println("----------------------------------Matched
Question---------------------------------------");
                //4-feb
                // int top=5;
                int top=3;
                if(count<top)
                        top=count;
                for(int i=0;i<top;i++)
                {
                        //System.out.println(i+"."+faq_id[i]+" "+score[i]+"%
"+srch.getQuestionByFaqID(faq_id[i]));
                        if(score[i]>0)
                         System.out.println(i+"."+faq_id[i]+"
"+Math.round(score[i])+"% "+/*srch.getQuestionByFaqID(faq_id[i]));
        System.out.println(*/TextPreprocessing.removeEnglishStopWords(srch.g
etQuestionByFaqID(faq_id[i])));
                }
        }
        public static void main(String[] args)
        {
        System.out.println(TextPreprocessing.removeEnglishStopWords("the how
what are there is anwar"));
        }
}
```

`

faqRetrieval.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 parsing.properties;
import faqIndexing.FAQ_Search;
import faqIndexing.HashTable;
public class faqRetrieval
{
      static boolean debug=false;
      static HashTable hashTable=new HashTable();
      public static String getMatchingQuestion(String sms_id,String sms)
throws CorruptIndexException, LockObtainFailedException, IOException,
ParseException
      {
             if(debug)
             System.out.println("SMS Query:"+sms);
             sms = sms.replace('?',' ').replace('*',' ').replace('<','
').replace('>',' ').replace('%',' ').replace('&', ' ').replace(';','
').replace('-',' ').replace("'"," ").replace('[',' ').replace('{','
').replace(']',' ').replace('}',' ').replace('!',' ').replace('(', '
').replace('.', ' ').replace(')', ' ').replace(',', ' ').replace('/', '
').replace('#', ' ').replace(':', ' ').replace('"', '
').trim().toLowerCase();
             //String original_smstext=sms;
             //for dictionary lookup
             FAQ_Search srch=new
FAQ_Search(System.getProperty("DomainIndexPath"));
             String question = null;
             //remove single characters
             String smsText=listcreation.removeSingleLetters(sms);
             if(debug)
             System.out.println("Single Char removed :"+smsText);
             if(System.getProperty("removeStopWords").equals("true"))
             {
             //replace the stop words
             if(System.getProperty("smsLanguage").equals("Hindi"))
             smsText=TextPreprocessing.removeHinidStopWords(smsText);
             else if(System.getProperty("smsLanguage").equals("English"))
             smsText=TextPreprocessing.removeEnglishStopWords(smsText);
             if(debug)
             System.out.println("Stop word removed:"+smsText);
             }
             int smsProximityArray[];
             smsProximityArray = findSmsProximity(sms,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(sms_id,sms,smsText,srch,smsProximit
yArray,hashTable);
             //retrieve matching questions
             return question;
         }
         private static int[] findSmsProximity(String smsOriginal, String
smsProcessed) {
             // TODO Auto-generated method stub
             //separate out question tokens
             StringTokenizer original = new StringTokenizer(smsOriginal,"
");
             StringTokenizer processed = new StringTokenizer(smsProcessed,"
");
             String tokenProcessed,tokenOriginal;
             int totalProcessed=processed.countTokens();
             int smsProximity[]=new int[totalProcessed] ;
             int OriginalCount=0;
             for(int
ProcessedCount=0;ProcessedCount<totalProcessed;ProcessedCount++)
             {
                 tokenProcessed = processed.nextToken();
                 tokenOriginal  = original.nextToken();
                 while(!tokenOriginal.equalsIgnoreCase(tokenProcessed))
                 {
                          OriginalCount++;
                          tokenOriginal  = original.nextToken();
                 }
                 smsProximity[ProcessedCount]= OriginalCount;
                 OriginalCount++;
                 if(debug)
                 System.out.println(smsProximity[ProcessedCount]);
             }
             //calculate distance between two consecutive terms
             if(debug)
             {
                 System.out.println(smsOriginal);
                 System.out.println(smsProcessed);
             }
             return smsProximity;
         }
         public static void main(String[] args)
         {
             try {
                 properties.load("English");
                 getMatchingQuestion("ENG_SMS_QUERY_C157","who ws the
1st indian ldy chief minstr?");
                 getMatchingQuestion("ENG_SMS_QUERY_C157","where can i
find information about pesticide establishment registration and
reporting");
             } 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]
}
```