SMS based Interface for FAQ Retrieval

Govind Kothari
IBM India Research Lab
gokothar@in.ibm.com

Sumit Negi
IBM India Research Lab
sumitneg@in.ibm.com

Tanveer A. Faruquie
IBM India Research Lab
ftanveer@in.ibm.com

Venkatesan T. Chakaravarthy
IBM India Research Lab
vechakra@in.ibm.com

L. Venkata Subramaniam
IBM India Research Lab
lvsubram@in.ibm.com

Abstract

Short Messaging Service (SMS) is popularly used to provide information access to people on the move. This has resulted in the growth of SMS based Question Answering (QA) services. However automatically handling SMS questions poses significant challenges due to the inherent noise in SMS questions. In this work we present an automatic FAQ-based question answering system for SMS users. We handle the noise in a SMS query by formulating the query similarity over FAQ questions as a combinatorial search problem. The search space consists of combinations of all possible dictionary variations of tokens in the noisy query. We present an efficient search algorithm that does not require any training data or SMS normalization and can handle semantic variations in question formulation. We demonstrate the effectiveness of our approach on two real-life datasets.

1 Introduction

The number of mobile users is growing at an amazing rate. In India alone a few million subscribers are added each month with the total subscriber base now crossing 370 million. The anytime anywhere access provided by mobile networks and portability of handsets coupled with the strong human urge to quickly find answers has fueled the growth of information based services on mobile devices. 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. 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, say June, the user has to type DUPBILLJUN. This unnecessarily constraints users who generally find it easy and intuitive to type in a "texting" language.

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. This mode of communication not only makes economic sense but also saves the customer from the hassle of waiting in a call queue. Most of these contact center based services and other regular services like "AQA 63336"[1] by Issuebits Ltd, GTIP[2] by AlienPant Ltd., "Texperts"[3] by Number UK Ltd. and "ChaCha"[4] use human agents to understand the SMS text and respond to these SMS queries. The nature of texting language, which often as a rule rather than exception, has misspellings, non-standard abbreviations, transliterations, phonetic substitutions and omissions, makes it difficult to build automated question answering systems around SMS technology. This is true even for questions whose answers are well documented like a FAQ database. Unlike other automatic question answering systems that focus on generating or searching answers, in a FAQ database the question and answers are already provided by an expert. The task is then to identify the best matching question-answer pair for a given query.

In this paper we present a FAQ-based question answering system over a SMS interface. Our

[1] http://www.aqa.63336.com/
[2] http://www.gtip.co.uk/
[3] http://www.texperts.com/
[4] http://www.chacha.com/

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.

system allows the user to enter a question in the SMS texting language. Such questions are noisy and contain spelling mistakes, abbreviations, deletions, phonetic spellings, transliterations etc. Since mobile handsets have limited screen space, it necessitates that the system have high accuracy. We handle the noise in a SMS query by formulating the query similarity over FAQ questions as a combinatorial search problem. The search space consists of combinations of all possible dictionary variations of tokens in the noisy query. The quality of the solution, i.e. the retrieved questions is formalized using a scoring function. Unlike other SMS processing systems our model does not require training data or human intervention. Our system handles not only the noisy variations of SMS query tokens but also semantic variations. We demonstrate the effectiveness of our system on real-world data sets.

The rest of the paper is organized as follows. Section 2 describes the relevant prior work in this area and talks about our specific contributions. In Section 3 we give the problem formulation. Section 4 describes the Pruning Algorithm which finds the best matching question for a given SMS query. Section 5 provides system implementation details. Section 6 provides details about our experiments. Finally we conclude in Section 7.

2 Prior Work

There has been growing interest in providing access to applications, traditionally available on Internet, on mobile devices using SMS. Examples include Search (Schusteritsch et al., 2005), access to Yellow Page services (Kopparapu et al., 2007), Email [5], Blog [6] , FAQ retrieval [7] etc. As highlighted earlier, these SMS-based FAQ retrieval services use human experts to answer questions.

There are other research and commercial systems which have been developed for general question and answering. These systems generally adopt one of the following three approaches: Human intervention based, Information Retrieval based, or Natural language processing based. Human intervention based systems exploit human communities to answer questions. These systems [8] are interesting because they suggest similar questions resolved in the past. Other systems like Chacha and Askme[9] use qualified human experts to answer questions in a timely manner. The information retrieval based system treat question answering as an information retrieval problem. They search large corpus of text for specific text, phrases or paragraphs relevant to a given question (Voorhees, 1999). In FAQ based question answering, where FAQ provide a ready made database of question-answer, the main task is to find the closest matching question to retrieve the relevant answer (Sneiders, 1999) (Song et al., 2007). The natural language processing based system tries to fully parse a question to discover semantic structure and then apply logic to formulate the answer (Molla et al., 2003). In another approach the questions are converted into a template representation which is then used to extract answers from some structured representation (Sneiders, 2002) (Katz et al., 2002). Except for human intervention based QA systems most of the other QA systems work in restricted domains and employ techniques such as named entity recognition, co-reference resolution, logic form transformation etc which require the question to be represented in linguistically correct format. These methods do not work for SMS based FAQ answering because of the high level of noise present in SMS text.

There exists some work to remove noise from SMS (Choudhury et al., 2007) (Byun et al., 2007) (Aw et al., 2006) (Kobus et al., 2008). However, all of these techniques require aligned corpus of SMS and conventional language for training. Building this aligned corpus is a difficult task and requires considerable human effort. (Acharya et al., 2008) propose an unsupervised technique that maps non-standard words to their corresponding conventional frequent form. Their method can identify non-standard transliteration of a given token only if the context surrounding that token is frequent in the corpus. This might not be true in all domains.

2.1 Our Contribution

To the best of our knowledge we are the first to handle issues relating to SMS based automatic question-answering. We address the challenges in building a FAQ-based question answering system over a SMS interface. Our method is unsupervised and does not require aligned corpus or explicit SMS normalization to handle noise. We propose an efficient algorithm that handles noisy

[5] http://www.sms2email.com/
[6] http://www.letmeparty.com/
[7] http://www.chacha.com/
[8] http://www.answers.yahoo.com/
[9] http://www.askmehelpdesk.com/

lexical and semantic variations.

3 Problem Formulation

We view the input SMS S as a sequence of tokens $S = s_1, s_2, \ldots, s_n$. Let $\mathcal{Q}$ denote the set of questions in the FAQ corpus. Each question $Q \in \mathcal{Q}$ is also viewed as a sequence of terms. Our goal is to find the question Q^* from the corpus $\mathcal{Q}$ that best matches the SMS S. As mentioned in the introduction, the SMS string is bound to have misspellings and other distortions, which needs to be taken care of while performing the match.

In the preprocessing stage, we develop a Domain dictionary $\mathcal{D}$ consisting of all the terms that appear in the corpus $\mathcal{Q}$. For each term t in the dictionary and each SMS token s_i, we define a *similarity measure* $\alpha(t, s_i)$ that measures how closely the term t matches the SMS token s_i. We say that the term t is a *variant* of s_i, if $\alpha(t, s_i) > 0$; this is denoted as $t \sim s_i$. Combining the *similarity measure* and the *inverse document frequency* (idf) of t in the corpus, we define a weight function $\omega(t, s_i)$. The *similarity measure* and the *weight function* are discussed in detail in Section 5.1.

Based on the weight function, we define a *scoring function* for assigning a score to each question in the corpus $\mathcal{Q}$. The score measures how closely the question matches the SMS string S. Consider a question $Q \in \mathcal{Q}$. For each token s_i, the scoring function chooses the term from Q having the maximum weight; then the weight of the n chosen terms are summed up to get the score.

$$\text{Score}(Q) = \sum_{i=1}^{n} \left[\max_{t:t \in Q \text{ and } t \sim s_i} \omega(t, s_i) \right] \quad (1)$$

Our goal is to efficiently find the question Q^* having the maximum score.

4 Pruning Algorithm

We now describe algorithms for computing the maximum scoring question Q^*. For each token s_i, we create a list L_i consisting of all terms from the dictionary that are *variants* of s_i. Consider a token s_i. We collect all the *variants* of s_i from the dictionary and compute their weights. The *variants* are then sorted in the descending order of their weights. At the end of the process we have n ranked lists. As an illustration, consider an SMS query *"gud plc buy 10s strng on9"*. Here, $n = 6$ and six lists of *variants* will be created as shown in Figure 1. The process of creating the lists is speeded up using suitable indices, as explained in detail in Section 5.

gud	plc	buy	10s	strng	on9
god	pal	bay	ten	strung	onto
guide	police	boy	tines	storing	once
....	....	**buy**	....	....	....
gum	**place**	busy	tend	stars	**online**
good	pill	....	**tennis**	**string**	ontario
....	....	....	....	....	....

Figure 1: Ranked List of Variations

Now, we assume that the lists $L_1, L_2, \ldots, L_n$ are created and explain the algorithms for computing the maximum scoring question Q^*. We describe two algorithms for accomplishing the above task. The two algorithms have the same functionality i.e. they compute Q^*, but the second algorithm called the *Pruning algorithm* has a better run time efficiency compared to the first algorithm called the *naive algorithm*. Both the algorithms require an index which takes as input a term t from the dictionary and returns Q_t, the set of all questions in the corpus that contain the term t. We call the above process as *querying* the index on the term t. The details of the index creation is discussed in Section 5.2.

Naive Algorithm: In this algorithm, we scan each list L_i and *query* the index on each term appearing in L_i. The returned questions are added to a collection $\mathcal{C}$. That is,

$$\mathcal{C} = \bigcup_{i=1}^{n} \left[\bigcup_{t \in L_i} Q_t \right]$$

The collection $\mathcal{C}$ is called the *candidate set*. Notice that any question not appearing in the *candidate set* has a score 0 and thus can be ignored. It follows that the *candidate set* contains the maximum scoring question Q^*. So, we focus on the questions in the collection $\mathcal{C}$, compute their scores and find the maximum scoring question Q^*. The scores of the question appearing in $\mathcal{C}$ can be computed using Equation 1.

The main disadvantage with the naive algorithm is that it queries each term appearing in each list and hence, suffers from high run time cost. We next explain the Pruning algorithm that avoids this pitfall and queries only a substantially small subset of terms appearing in the lists.

Pruning Algorithm: The pruning algorithm

is inspired by the Threshold Algorithm (Fagin et al., 2001). The Pruning algorithm has the property that it queries fewer terms and ends up with a smaller *candidate set* as compared to the naive algorithm. The algorithm maintains a *candidate set* $\mathcal{C}$ of questions that can potentially be the maximum scoring question. The algorithm works in an iterative manner. In each iteration, it picks the term that has maximum weight among all the terms appearing in the lists $L_1, L_2, \ldots, L_n$. As the lists are sorted in the descending order of the weights, this amounts to picking the maximum weight term amongst the first terms of the n lists. The chosen term t is queried to find the set Q_t. The set Q_t is added to the candidate set $\mathcal{C}$. For each question $Q \in Q_t$, we compute its score $\mathrm{Score}(Q)$ and keep it along with Q. The score can be computed by Equation 1 (For each SMS token s_i, we choose the term from Q which is a variant of s_i and has the maximum weight. The sum of the weights of chosen terms yields $\mathrm{Score}(Q)$). Next, the chosen term t is removed from the list. Each iteration proceeds as above. We shall now develop a thresholding condition such that when it is satisfied, the candidate set $\mathcal{C}$ is guaranteed to contain the maximum scoring question Q^*. Thus, once the condition is met, we stop the above iterative process and focus only on the questions in $\mathcal{C}$ to find the maximum scoring question.

Consider end of some iteration in the above process. Suppose Q is a question not included in $\mathcal{C}$. We can upperbound the score achievable by Q, as follows. At best, Q may include the top-most token from every list $L_1, L_2, \ldots, L_n$. Thus, score of Q is bounded by

$$\mathrm{Score}(Q) \leq \sum_{i=0}^{n} \omega(L_i[1]).$$

(Since the lists are sorted $L_i[1]$ is the term having the maximum weight in L_i). We refer to the RHS of the above inequality as UB.

Let $\widehat{Q}$ be the question in $\mathcal{C}$ having the maximum score. Notice that if $\widehat{Q} \geq \mathrm{UB}$, then it is guaranteed that any question not included in the candidate set $\mathcal{C}$ cannot be the maximum scoring question. Thus, the condition "$\widehat{Q} \geq \mathrm{UB}$" serves as the termination condition. At the end of each iteration, we check if the termination condition is satisfied and if so, we can stop the iterative process. Then, we simply pick the question in $\mathcal{C}$ having the maximum score and return it. The algorithm is shown in Figure 2.

In this section, we presented the Pruning algorithm that efficiently finds the best matching question for the given SMS query without the need to go through all the questions in the FAQ corpus. The next section describes the system implementation details of the Pruning Algorithm.

Procedure Pruning Algorithm
Input: SMS $S = s_1, s_2, \ldots, s_n$
Output: Maximum scoring question Q^*.
Begin
  Construct lists $L_1, L_2, \ldots, L_n$ //(see Section 5.3).
    // L_i lists variants of s_i in descending
    //order of weight.
  Candidate list $\mathcal{C} = \emptyset$.
  repeat
    $j^* = \mathrm{argmax}_i \omega(L_i[1])$
    $t^* = L_{j^*}[1]$
      // t^* is the term having maximum weight among
      // all terms appearing in the n lists.
    Delete t^* from the list L_{j^*}.
    Query the index and fetch Q_{t^*}
      // Q_{t^*}: the set of all questions in $\mathcal{Q}$
      //having the term t^*
    For each $Q \in Q_{t^*}$
      Compute $\mathrm{Score}(Q)$ and
      add Q with its score into $\mathcal{C}$
    $\mathrm{UB} = \sum_{i=1}^{n} \omega(L_i[1])$
    $\widehat{Q} = \mathrm{argmax}_{Q \in \mathcal{C}} \mathrm{Score}(Q)$.
    if $\mathrm{Score}(\widehat{Q}) \geq \mathrm{UB}$, **then**
        // Termination condition satisfied
      Output $\widehat{Q}$ and exit.
  forever
End

Figure 2: Pruning Algorithm

5 System Implementation

In this section we describe the weight function, the preprocessing step and the creation of lists $L_1, L_2, \ldots, L_n$.

5.1 Weight Function

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

5.1.1 Similarity Measure

Let $\mathcal{D}$ be the dictionary of all the terms in the corpus $\mathcal{Q}$. For term $t \in \mathcal{D}$ and token s_i of the SMS, the *similarity measure* $\alpha(t, s_i)$ between them is

$$\alpha(t, s_i) = \begin{cases} \frac{LCSRatio(t,s_i)}{EditDistance_{SMS}(t,s_i)} & \text{if t and } s_i \text{ share same starting character *} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $LCSRatio(t, s_i) = \frac{length(LCS(t,s_i))}{length(t)}$ *and* $LCS(t, s_i)$ *is the Longest common subsequence between* t *and* s_i.

** The rationale behind this heuristic is that while typing a SMS, people typically type the first few characters correctly. Also, this heuristic helps limit the variants possible for a given token.*

The *Longest Common Subsequence Ratio* (LCSR) (Melamed, 1999) 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 *LCSRatio*.

Procedure $EditDistance_{SMS}$
Input: term t, token s_i
Output: Consonant Skeleton Edit distance
Begin
  return $LevenshteinDistance(CS(s_i), CS(t)) + 1$
    // 1 is added to handle the case where
    // Levenshtein Distance is 0
End

Consonant Skeleton Generation (CS)
  1. remove consecutive repeated characters
    // ($call \rightarrow cal$)
  2. remove all vowels
    //($waiting \rightarrow wtng, great \rightarrow grt$)

Figure 3: $EditDistance_{SMS}$

The $EditDistance_{SMS}$ shown in Figure 3 compares the Consonant Skeletons (Prochasson et al., 2007) 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 2 will be high.

We explain the rationale behind using the $EditDistance_{SMS}$ in the *similarity measure* $\alpha(t, s_i)$ 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 $EditDistance_{SMS}$ of "good" is 1 which is less than that of "guided", which has $EditDistance_{SMS}$ 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".

5.1.2 Inverse Document Frequency

If f number of documents in corpus $\mathcal{Q}$ contain a term t and the total number of documents in $\mathcal{Q}$ is *N*, the *Inverse Document Frequency (idf)* of t is

$$idf(t) = \log \frac{N}{f} \quad (3)$$

Combining the *similarity measure* and the *idf* of t in the corpus, we define the weight function $\omega(t, s_i)$ as

$$\omega(t, s_i) = \alpha(t, s_i) * idf(t) \quad (4)$$

The objective behind the weight function is

1. We prefer terms that have high *similarity measure* i.e. terms that are similar to the SMS token. Higher the *LCSRatio* and lower the $EditDistance_{SMS}$, 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. We prefer 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*.

We combine these two objectives into a single weight function multiplicatively.

5.2 Preprocessing

Preprocessing involves indexing of the FAQ corpus, formation of Domain and Synonym dictionaries and calculation of the Inverse Document Frequency for each term in the Domain dictionary.

As explained earlier the Pruning algorithm requires retrieval of all questions Q_t that contains a given term t. To do this efficiently we index the FAQ corpus using *Lucene*[10]. Each question in the FAQ corpus is treated as a *Document*; it is tokenized using whitespace as delimiter and indexed.

[10]http://lucene.apache.org/java/docs/

The Domain dictionary $\mathcal{D}$ is built from all terms that appear in the corpus $\mathcal{Q}$.

The weight calculation for Pruning algorithm requires the *idf* for a given term t. For each term t in the Domain dictionary, we *query* the Lucene indexer to get the number of Documents containing t. Using Equation 3, the *idf(t)* is calculated. The *idf* for each term t is stored in a Hashtable, with t as the *key* and *idf* as its *value*.

Another key step in the preprocessing stage is the creation of the Synonym dictionary. The Pruning algorithm uses this dictionary to retrieve semantically similar questions. Details of this step is further elaborated in the List Creation sub-section. The Synonym Dictionary creation involves mapping each word in the Domain dictionary to it's corresponding *Synset* obtained from WordNet[11].

5.3 List Creation

Given a SMS S, it is tokenized using white-spaces to get a sequence of tokens $s_1, s_2, \ldots, s_n$. Digits occurring in SMS token (e.g '10s' , "4get") are replaced by string based on a manually crafted digit-to-string mapping ("10" $\rightarrow$ "ten"). A list L_i is setup for each token s_i using terms in the domain dictionary. The list for a single character SMS token is set to *null* as it is most likely to be a stop word . A term t from domain dictionary is included in L_i if its first character is same as that of the token s_i and it satisfies the threshold condition

$$length(LCS(t, s_i)) > 1.$$

Each term t that is added to the list is assigned a weight given by Equation 4.

Terms in the list are ranked in descending order of their weights. Henceforth, the term "list" implies a ranked list.

For example the SMS query *"gud plc 2 buy 10s strng on9"* (corresponding question *"Where is a good place to buy tennis strings online?"*), is tokenized to get a set of tokens {*'gud', 'plc', '2', 'buy', '10s', 'strng', 'on9'*}. Single character tokens such as *'2'* are neglected as they are most likely to be stop words. From these tokens corresponding lists are setup as shown in Figure 1.

5.3.1 Synonym Dictionary Lookup

To retrieve answers for SMS queries that are semantically similar but lexically different from questions in the FAQ corpus we use the Synonym dictionary described in Section 5.2. Figure 4 illustrates some examples of such SMS queries.

[11] http://wordnet.princeton.edu/

SMS: *"frm wch ntion tenis strtd"*
Question: Which country did tennis originate from?
SMS: *"uz of warin wrstb&"*
Question: What is the purpose of wearing a wristband while playing tennis?

Figure 4: Semantically similar SMS and questions

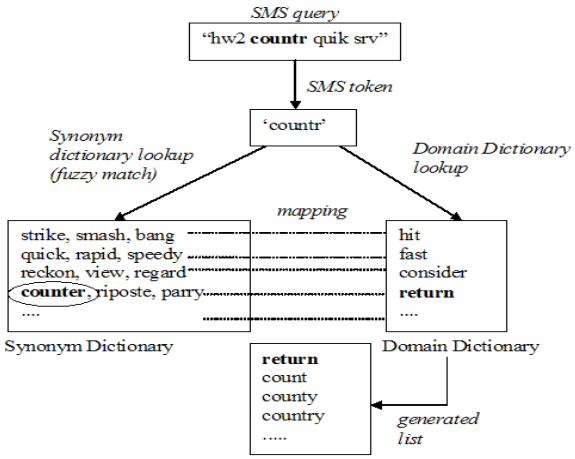

Figure 5: Synonym Dictionary LookUp

For a given SMS token s_i, the list of variations L_i is further augmented using this Synonym dictionary. For each token s_i a fuzzy match is performed between s_i and the terms in the Synonym dictionary and the best matching term from the Synonym dictionary, δ is identified. As the mappings between the Synonym and the Domain dictionary terms are maintained, we obtain the corresponding Domain dictionary term β for the Synonym term δ and add that term to the list L_i. β is assigned a weight given by

$$\omega(\beta, s_i) = \alpha(\delta, s_i) * idf(\beta) \quad (5)$$

It should be noted that weight for β is based on the *similarity measure* between Synonym dictionary term δ and SMS token s_i.

For example, the SMS query *"hw2 countr quik srv"*(corresponding question *"How to return a very fast serve?"*) has two terms *"countr"* $\rightarrow$ *"counter"* and *"quik"* $\rightarrow$ *"quick"* belonging to the Synonym dictionary. Their associated mappings in the Domain dictionary are *"return"* and *"fast"* respectively as shown in Figure 5. During the list setup process the token *"countr"* is looked

up in the Domain dictionary. Terms from the Domain dictionary that begin with the same character as that of the token *"countr"* and have a $LCS > 1$ such as *"country","count", etc.* are added to the list and assigned a weight given by Equation 4. After that, the token *"countr"* is looked up in the Synonym dictionary using Fuzzy match. In this example the term *"counter"* from the Synonym dictionary fuzzy matches the SMS token. The Domain dictionary term corresponding to the Synonym dictionary term *"counter"* is looked up and added to the list. In the current example the corresponding Domain dictionary term is *"return"*. This term is assigned a weight given by Equation 5 and is added to the list as shown in Figure 5.

5.4 FAQ retrieval

Once the lists are created, the Pruning Algorithm as shown in Figure 2 is used to find the FAQ question Q^* that best matches the SMS query. The corresponding answer to Q^* from the FAQ corpus is returned to the user.

The next section describes the experimental setup and results.

6 Experiments

We validated the effectiveness and usability of our system by carrying out experiments on two FAQ data sets. The first FAQ data set, referred to as the Telecom Data-Set, consists of 1500 frequently asked questions, collected from a Telecom service provider's website. The questions in this data set are related to the Telecom providers products or services. For example queries about call rates/charges, bill drop locations, how to install caller tunes, how to activate GPRS etc. The second FAQ corpus, referred to as the Yahoo DataSet, consists of 7500 questions from three Yahoo! Answers[12] categories namely Sports.Swimming, Sports.Tennis, Sports.Running.

To measure the effectiveness of our system, a user evaluation study was performed. Ten human evaluators were asked to choose 10 questions randomly from the FAQ data set. None of the evaluators were authors of the paper. They were provided with a mobile keypad interface and asked to "text" the selected 10 questions as SMS queries. Through that exercise 100 relevant SMS queries per FAQ data set were collected. Figure 6 shows sample SMS queries. In order to validate that the system was able to handle queries that were out of

[12]http://answers.yahoo.com/

whr cn i fnd d cart kit
watz d most comfy bke seat
whr cn IC d gmes of d Roland Garros n www
w@ r d br&s of 10s balz uz n tourneys
watz d diff btw a midsize n a mid+ rqts

Figure 6: Sample SMS queries

Data Set	Relevant Queries	Irrelevant Queries
Telecom	100	50
Yahoo	100	50

Table 1: SMS Data Set.

the FAQ domain, we collected 5 irrelevant SMS queries from each of the 10 human-evaluators for both the data sets. Irrelevant queries were (a) Queries out of the FAQ domain e.g. queries related to Cricket, Billiards, activating GPS etc (b) Absurd queries e.g. "ama ameyu tuem" (sequence of meaningless words) and (c) General Queries e.g. "what is sports". Table 1 gives the number of relevant and irrelevant queries used in our experiments.

The average word length of the collected SMS messages for Telecom and Yahoo datasets was 4 and 7 respectively. We manually cleaned the SMS query data word by word to create a clean SMS test-set. For example, the SMS query *"h2 mke a pdl bke fstr"* was manually cleaned to get *"how to make pedal bike faster"*. In order to quantify the level of noise in the collected SMS data, we built a character-level language model(LM)[13] using the questions in the FAQ data-set (vocabulary size is 44 characters) and computed the perplexity[14] of the language model on the noisy and the cleaned SMS test-set. The perplexity of the LM on a corpus gives an indication of the average number of bits needed per n-gram to encode the corpus. Noise will result in the introduction of many previously unseen n-grams in the corpus. Higher number of bits are needed to encode these improbable n-grams which results in increased perplexity. From Table 2 we can see the difference in perplexity for noisy and clean SMS data for the Yahoo and Telecom data-set. The high level of perplexity in the SMS data set indicates the extent of noise present in the SMS corpus.

To handle irrelevant queries the algorithm described in Section 4 is modified. Only if the Score(Q^*) is above a certain threshold, it's answer is returned, else we return *"null"*. The threshold

[13]http://en.wikipedia.org/wiki/Language_model
[14]bits = $\log_2$(perplexity)

		Cleaned SMS	Noisy SMS
Yahoo	bigram	14.92	74.58
	trigram	8.11	93.13
Telecom	bigram	17.62	59.26
	trigram	10.27	63.21

Table 2: Perplexity for Cleaned and Noisy SMS

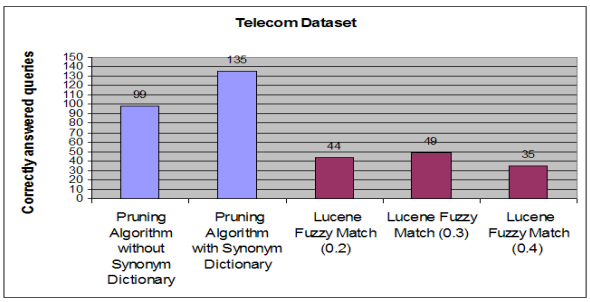

Figure 7: Accuracy on Telecom FAQ Dataset

was determined experimentally.

To retrieve the correct answer for the posed SMS query, the SMS query is matched against questions in the FAQ data set and the best matching question(Q^*) is identified using the Pruning algorithm. The system then returns the answer to this best matching question to the human evaluator. The evaluator then scores the response on a binary scale. A score of 1 is given if the returned answer is the correct response to the SMS query, else it is assigned 0. The scoring procedure is reversed for irrelevant queries i.e. a score of 0 is assigned if the system returns an answer and 1 is assigned if it returns *"null"* for an "irrelevant" query. The result of this evaluation on both data-sets is shown in Figure 7 and 8.

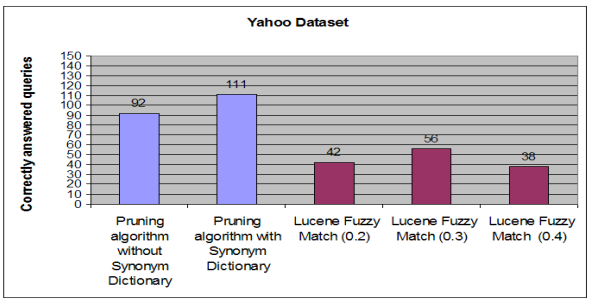

Figure 8: Accuracy on Yahoo FAQ Dataset

In order to compare the performance of our system, we benchmark our results against Lucene's [15] Fuzzy match feature. Lucene supports fuzzy searches based on the Levenshtein Distance, or Edit Distance algorithm. To do a fuzzy search we specify the $\sim$ symbol at the end of each token of the SMS query. For example, the SMS query "romg actvt" on the FAQ corpus is reformulated as "romg~ 0.3 actvt~ 0.3". The parameter after the $\sim$ specifies the required *similarity*. The parameter value is between 0 and 1, with a value closer to 1 only terms with higher similarity will be matched. These queries are run on the indexed FAQs. The results of this evaluation on both data-sets is shown in Figure 7 and 8. The results clearly demonstrate that our method performs 2 to 2.5 times better than Lucene's Fuzzy match. It was observed that with higher values of *similarity* parameter (~ 0.6, ~ 0.8), the number of correctly answered queries was even lower. In Figure 9 we show the runtime performance of the Naive *vs* Pruning algorithm on the Yahoo FAQ Dataset for 150 SMS queries. It is evident from Figure 9 that not only does the Pruning Algorithm outperform the Naive one but also gives a near-constant runtime performance over all the queries. The substantially better performance of the Pruning algorithm is due to the fact that it queries much less number of terms and ends up with a smaller candidate set compared to the Naive algorithm.

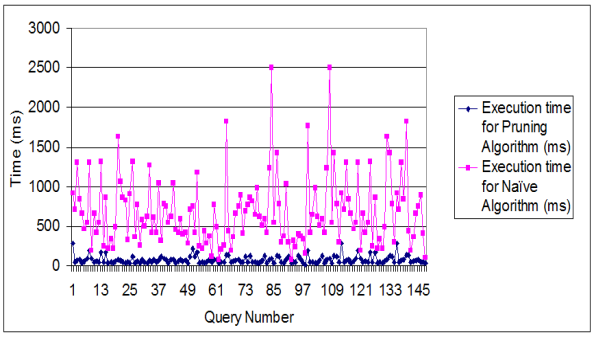

Figure 9: Runtime of Pruning *vs* Naive Algorithm for Yahoo FAQ Dataset

7 Conclusion

In recent times there has been a rise in SMS based QA services. However, automating such services has been a challenge due to the inherent noise in SMS language. In this paper we gave an efficient algorithm for answering FAQ questions over an SMS interface. Results of applying this on two different FAQ datasets shows that such a system can be very effective in automating SMS based FAQ retrieval.

[15] http://lucene.apache.org

References

Rudy Schusteritsch, Shailendra Rao, Kerry Rodden. 2005. Mobile Search with Text Messages: Designing the User Experience for Google SMS. *CHI*, Portland, Oregon.

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.

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.

E. Voorhees. 1999. The TREC-8 question answering track report.

D. Molla. 2003. NLP for Answer Extraction in Technical Domains, In *Proceedings of EACL*, USA.

E. Sneiders. 2002. Automated question answering using question templates that cover the conceptual model of the database, In *Proceedings of NLDB*, pages 235−239.

B. Katz, S. Felshin, D. Yuret, A. Ibrahim, J. Lin, G. Marton, and B. Temelkuran. 2002. Omnibase: Uniform access to heterogeneous data for question answering, *Natural Language Processing and Information Systems*, pages 230−234.

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

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

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.

Catherine Kobus, Franois Yvon and Graldine Damnati. 2008. Normalizing SMS: are two metaphors better than one?, In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 441−448 Manchester.

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*

Ronald Fagin , Amnon Lotem , Moni Naor. 2001. Optimal aggregation algorithms for middleware, In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.*

I. Dan Melamed. 1999. Bitext maps and alignment via pattern recognition, *Computational Linguistics*.

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.*

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.*