JCSE International Journal of Computer Sciences and Engineering Open Access

Research Paper Volume-3, Issue-5 E-ISSN: 2347-2693

FAQ Retrieval Using Noisy Queries

Vaishnavi Barla[1*], SriMounika Achanti[2] and Rohith Uppala[3]

[1*, 2, 3] *Department of Computer Science and Engineering, National Institute of Technology, Warangal, India,*

www.ijcseonline.org

Received: Apr/26/2015 Revised: May/06//2015 Accepted: May/22/2015 Published: May/30/ 2015

Abstract— In spite of great advances of information retrieval systems and associated natural language processing technologies, domain-specific retrieval systems and retrieval systems used with special types of queries continue to represent a challenge for current technology and to be a topic of active research. The Forum for Information Retrieval Evaluation (FIRE) is a forum for Information Retrieval evaluation that is traditionally mainly focused on Indian languages. India, with its huge population, has a very high rate of using mobile phones, with service costs low enough for even very poor people to use the phones actively. Accordingly, FIRE 2013 included an SMS-based FAQ retrieval task. The goal of this task was to find a question Q* from corpora of FAQs (Frequently Asked Questions) that best answers or matches a given SMS query S

Keywords—Question matching,Question classification,Core NLP,Support vector machines,Unigram matching,Syntactic tree matching,Syntactic tree matching

I. Introduction

There has been a tremendous growth in the number of new mobile subscribers recently. Most of these new subscribers are from developing countries where mobile are the primary information device. Even for users familiar with computers and the internet, the mobile provides unmatched portability. This has encouraged the proliferation of information services built around SMS technology. Several applications, traditionally available on Internet, are now being made available on mobile devices using SMS. Examples include SMS short code services. Short codes are numbers where a short message in a predesigned format can be sent to get specific information. For example, to get the closing stock price of a particular share, the user has to send a message IBMSTOCKPR.

Recent studies have shown that instant messaging is emerging as the preferred mode of communication after speech and email. 4 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. Typical question answering systems are built for use with languages which are free from such errors. It is difficult to build an automated question answering system around SMS technology. This is true even for questions whose answers are well documented like in a Frequently-Asked-Questions (FAQ) database.

Different kinds of noise correction techniques exist which include:

a) Statistical translation model which use large parallel corpora to learn the translation probabilities.
b) Token based noise correction techniques (such as those using LCS, Edit Distance etc.)

In our proposed model, we use token based noise correction techniques i.e. LCS, Edit Distance. These techniques work based on the intuition that the user always types the first few letters correct. In order to cover the cases which are not handled above, we use soundex algorithm to find out the required clean text. (Ex: u-you).

II. Question Classification And Matching

In order to correctly answer a question, usually one needs to understand what the question asks for. Question Classification, i.e. putting the questions into several semantic categories, can not only impose some constraints on the plausible answers but also suggest different processing strategies. For instance, if the system understands that the question "Who was the first American in space?" asks for a person name, the search space of plausible answers will be significantly reduced. In fact, almost all the open-domain question answering systems include a question classification module. The accuracy of question classification is very important to the overall performance of the question answering system.

Classifying a question helps in reducing the search space for matching the input cleaned query with the questions in the archive. We will be using Support Vector Machines to get our task done

Community Question Answering (CQA) services have accumulated large archives of question-answer pairs. To reuse the invaluable resources, it is essential to develop

Corresponding Author: *Vaishnavi Barla*

effective retrieval models to retrieve similar questions, which are semantically equivalent or relevant to the queried questions.

The major challenge for question retrieval, as for most information retrieval tasks, is the *lexical and semantic gap* between the queried question and the historical questions in the CQA archives.

The similar question matching task is, however, not trivial. One of the major reasons is that instead of inputting just keywords or so, users form questions using natural language, where questions are encoded with various lexical, syntactic and semantic features. For example, "*how can I lose weight in a few month?*" and "*are there any ways of losing pound in a short period?*" are two similar questions asking for methods of losing weight, but they neither share many common words nor follow identical syntactic structure. This gap makes the similar question matching task difficult. Similarity measure techniques based purely on the bag-of-word (BoW) approach may perform poorly and become ineffective in these circumstances. Syntactic or semantic features hence become vital for such task. The tree kernel function is one of the most effective ways to represent the syntactic structure of a sentence. In general, it divides the parsing tree into several sub-trees and computes the inner product between two vectors of sub-trees. Although there have been some successful applications using it, like Question Classification, the tree kernel-like function has not been directly applied to finding similar questions in the QA archive.

Moreover, its matching scheme is too strict to be directly employed to our question matching problem. We reformulate the tree kernel framework, and introduce a new retrieval model to find similar questions. We study the structural representations of questions to encode not only lexical but also syntactic and semantic features into the matching model. This model does not rely on training, and it is shown to be robust against grammatical errors as well

III. Design Details And Requirements

We have implemented a modified tree kernel algorithm in this project to attain more efficiency. In order to do this, we have used several software packages such as Scikit-Learn for SVM classification, CORENLP for Parts Of Speech (POS) Tagging, ZEROMQ for cross language communication between python and C++. We have also used Soundex algorithm to add vowels to noisy words and Dynamic programming methodologies to calculate similarity between two tree kernels.

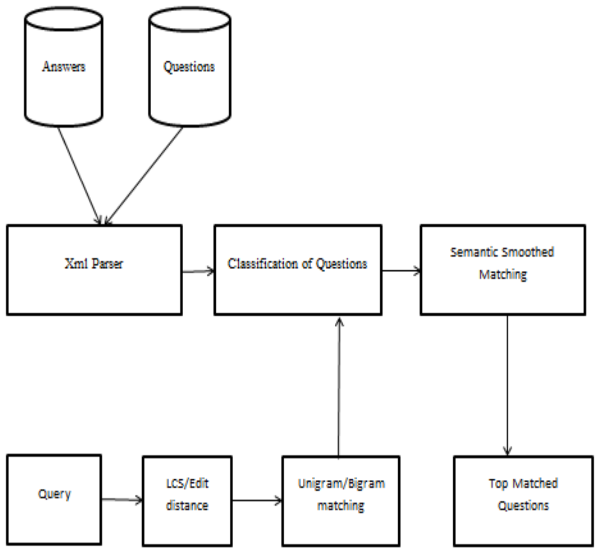


Fig. 1. Overview of Question Matching System

A. CoreNLP

Stanford CoreNLP provides a set of natural language analysis tools which can take raw English language text input and give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, and indicate which noun phrases refer to the same entities. Stanford CoreNLP is an integrated framework, which make it very easy to apply a bunch of language analysis tools to a piece of text. Starting from plain text, you can run all the tools on it with just two lines of code. Its analyses provide the foundational building blocks for higher-level and domain-specific text understanding applications.

Stanford CoreNLP integrates all our NLP tools, including the part-of-speech (POS) tagger, the parser. It is designed to be highly flexible and extensible. With a single option you can change which tools should be enabled and which should be disabled.

B. NLTK

NLTK (Natural Language Tool Kit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, and tagging, parsing, and semantic reasoning. NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language." Some simple things we can do with NLTK are tokenize and tag some text, identify named entities, display a parse tree. We have used WordNet for finding semantic similarity between two words.

C. *Zero MQ*

Zero MQ is a library used to implement messaging and communication systems between applications and processes - fast and asynchronously. One of the types of IPC (Interprocess Communication). When compared to some much larger projects, which offer all necessary parts of enterprise messaging, Zero MQ remains as just a lightweight and fast tool. It enables cross platform messaging. We have used Zero MQ"s for communication between C++ and Python Codes.

D. *SCIKIT-LEARN*

Scikit-Learn is an open source machine learning library for Python. It features various classification, regression and clustering algorithms including support vector machines, logistic regression, naive Bayes, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. We have used SVC with linear kernels for classifying the input query. SVC with linear kernel is somewhat similar to libsvm with parameter linear, but it is found to be more flexible.
Following is the example classification of svm classifier over IRIS data set using different kernels. Small circles represent tuples while colored areas represent different classes.

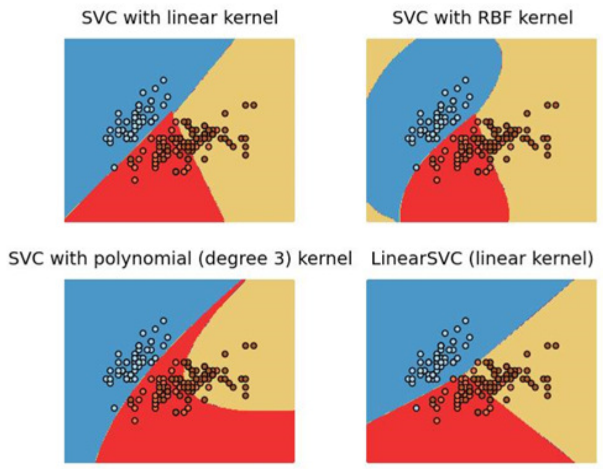


Fig.2 Different kernels of SVM classifier by using IR dataset.

IV. Related Work

The input query is noisy since it is in “SMS language”. Before matching the query with the questions in the archive, cleaning of the query is needed.

A. Soundex Algorithm

The phonetic representation has several applications. It allows searching concepts based on pronunciation rather than on the spelling, as it is traditionally done.

The Soundex phonetic algorithm was mainly used in applications involving searching of people’s names like air reservation systems, censuses, and other tasks presenting typing errors due to phonetic similarity. As shown, the Soundex algorithm evaluates each letter in the input word and assigns a numeric value. The main function of this algorithm is to convert each word into a code made up of four elements. Soundex uses numeric codes for each letter of the string to be codified.

Numeric code	Letter
0	a, e, i, o, u, y, h, w
1	b, p, f, v
2	c, g, j, k, q, s, x, z
3	d, t
4	l
5	m, n
6	r

Soundex phonetic codes for the English language Numeric

The Soundex algorithm can be depicted as follows:

1. Replace all but the first letter of the string by its phonetic code
2. Eliminate any adjacent repetitions of codes.
3. Eliminate all eliminate vowels.
4. Return the first four characters of the resulting string

The Soundex algorithm has the following features:

1. It is intuitive in terms of operation.
2. The simplicity of the code allows to implement changes according to the objective.
3. The processing time is relatively short.
4. It has a high tolerance for variations in words that sound very similar or are exactly the same.

B. *Statistical Translation Model*

Typical statistical machine translation systems use large parallel corpora to learn the translation probabilities (Brown et al., 2007). Traditionally such corpora have consisted of news articles and other well written articles. Since the translation systems are not trained on SMS language they perform very poorly when translating noisy SMS language. Parallel corpora comprising noisy sentences in one language and clean sentences in another language are not available and it would be hard to build such large parallel corpora to train a machine translation system.

C. *Token Based Techniques*

Token based noise-correction techniques (such as those using edit-distance, LCS etc.) cannot be directly applied to handle the noise present in the SMS query. These noise-correction methods return a list of candidate terms for a given noisy token (E.g. gud – > god, good, guide).

Considering all these candidate terms and their corresponding translations drastically increase the search space for any multi-lingual IR system. Also, naively replacing the noisy token in the SMS query with the top matching candidate term gives poor performance. The algorithm that we used handles these and related issues in an efficient manner.

D. Support Vector Machines For Classification

Support Vector Machines (SVM) are linear functions of the form

$$f(x) = \mathbf{w} \bullet \mathbf{x} + b$$

Where $\mathbf{w} \bullet \mathbf{x}$ is the inner product between the weight vector $\mathbf{w}$ and the input vector $\mathbf{x}$. The SVM can be used as a classifier by setting the class to 1 if $f(x) > 0$ and to -1 otherwise. The main idea of SVM is to select a hyperplane that separates the positive and negative examples while maximizing the minimum margin. This corresponds to minimizing $\mathbf{w} \bullet \mathbf{w}$ subject to $y(\mathbf{w} \bullet \mathbf{x} + b) \geq 1$ for all i. Large margin classifiers are known to have good generalization properties.

To deal with cases where there may be no separating hyperplane, the soft margin SVM has been proposed. The soft margin SVM minimizes $\mathbf{w} \bullet \mathbf{w} + C\sum$ ξ i subject to y $\mathbf{w} \bullet \mathbf{x} + b \geq 1 - \xi$ for all i, where C is a parameter that controls the amount of training errors allowed. A key property of the Support Vector Machines is that the only operation it requires is the computation of dot products between pairs of examples. One may therefore replace the dot product with a Mercer kernel, implicitly mapping feature vectors in R^d into a new space R^m, and applying the original algorithm in this new feature space. The kernel methods provide an efficient way to carry out such computation when m is large or even infinite.

Given a question, we first parse it into a syntactic tree using a parser, and then we represent it as a vector in a high dimensional space which is defined over tree fragments. The tree fragments of a syntactic tree are all its sub-trees which include at least one terminal symbol (word) or one production rule, with the restriction that no production rules can be broken into incomplete parts.

Implicitly we enumerate all the possible tree fragments 1, 2, ..., m. These tree fragments are the axis of this m dimensional space. Note that this could be done only implicitly, since the number m is extremely large.

Every syntactic tree T is represented by an m dimensional vector

$$\mathbf{v}(T) = \{v1(T), v2(T), ..., vm(T)\},$$

where the I th element $vi(T)$ is the weight of the tree fragment in the i th dimension if the tree fragment is in the tree T and zero otherwise. The tree kernel of two syntactic trees T_1 and T_2 is actually the inner product of $\mathbf{v}_1(T)$ and $\mathbf{v}_2(T)$.

Our task is to determine the hyper plane separating two classes. If there happen to be n classes, we get n (n -1)/2 hyper planes. Every time when we try to find out if the query belongs to a particular class, we consider the training tuples of that class to be positive and the rest all to be negative. And then we determine if the query is positive or negative based on its dot product with the hyper plane that separates one class with another. Thus we get n(n-1)/2 results. The class that gets the maximum number of votes is considered to be the class of the input query.

E. Unigram Matching

Suppose the text of the SMS statement S′ contains a word pattern that can be expressed as <L1, L2, ..., L*n*>, and the text of the FAQ statement F′ contains the word pattern that can be expressed as <W1, W2, ..., W*k*>. Then we search for the match of each word of S′ in F′. If a direct match occurrs, then we do not search any further for the word L*i* and pick up the next word from the list and search again for that word. We consider that a direct match occurred if there is some FAQ word W*i* such that W*i* = L*j* for some $i <= k$ and $j <= n$.

Now, if there is no direct match, then we looked up the word in WordNet 3.0 and obtained its hyponyms, synonyms, etc., and searched each one of these words in the F′ list. If a match is found, then we pass on to next word. Otherwise, we search for an overlap between the synonym and hyponym list of the word L*j* and the synonym and hyponym list of the word W*i*. If any matching words are found, then we go to the next word in the S′ list and store that word as a matched word. Otherwise we skip the word and proceed for next word.

The expression used as a score for the unigram matching module was:

Unigram Score=(Number of SMS words that match FAQ)÷(Total Number of Words in SMS)

In this module, we aimed to find a match between two statements by considering the bigram occurrences of their words. We took the two consecutive words from the S′ list, represented as <L*i*, L*i* + 1>, and similarly from the F′ list, <W*j*, W*j* + 1>. If a match was found, then we went on to the next consecutive bigram. Otherwise, we looked up in WordNet all hyponyms and synonyms for the words of the bigram <L*i*, L*i*+1>, and similarly we retrieved all hyponyms and synonyms for the words of the bigram <W*j*, W*j* + 1>. Suppose we have a synonym list for L*i* as ($x1$, $x2$, ...,xn) and for L*i* + 1 a list ($y1$, $y2$, …, yk), and similarly for the pair <W*j*, W*j* + 1>, obtaining lists ($s1$, $s2$, ..., sn) and ($t1$, $t2$, …, *tk*), correspondingly. If there was any matching words between the lists for the SMS bigram <L*i*, L*i*+1> and the list for the FAQ bigram <W*j*, *Wj* + 1>, then we considered that a match was found, and proceed to analyze the next bigram sequence.

The expression used as a score for the bigram matching module was as follows:

Bigram Score=(Number of bigram matching)÷(Total number of bigrams in SMS)

F. *Final Ranking*

Finally, the overall scoring was calculated as the harmonic mean of the two particular scores:

Final Ranking=(Unigram Score*Bigram Score)÷(Unigram Score + Bigram Score)

G. *Basic Kernal Tree Approach For Question Matchin*

In order to utilize more structural or syntactical information and capture higher order dependencies between grammar rules, all tree fragments that occur in a parsing tree can be considered. A tree fragment can be defined as any sub-tree that includes more than one node, with the restriction that the entire rule productions must be included.

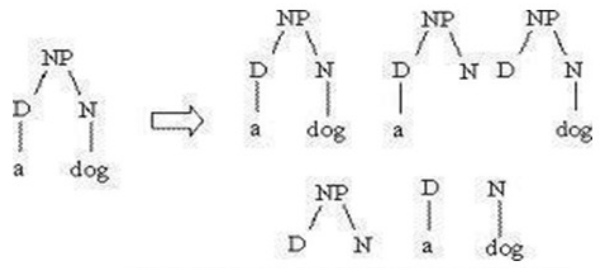


Fig.3 Tree fragments for the phrase "a dog"

The tree kernel was designed based on the idea of counting the number of tree fragments that are common to both parsing trees, and it could be defined as:

$$k(T_1,T_2) = \sum_{n1 \in N1}\sum_{n2 \in N2} C(n1,n2)$$

Where, *N1* and *N2* are sets of nodes in two syntactic trees *T1* and *T2, C (n1, n2)* equals to the number of common fragments rooted in nodes *n1* and *n2*.

However, to enumerate all possible tree fragments is an intractable problem. The tree fragments are thus implicitly represented, and with dynamic programming, the value of *C (n1, n2)* can be efficiently computed as follows:

C (n1, n2) = 0, if n1≠n2

C (n1, n2) =1, if n1=n2 and they are terminal nodes.

C (n1, n2) = λ, if n1=n2 and they are pre-terminal nodes.

C (n1, n2) = λ π j=1 nc (n1) [1 + C (ch (n1, j), ch (n2, j))],

Where,

λ is a weighting factor,

nc (n) is no. of children of node n. ch (n1,j) is the jth child of node n.

H. *SyntacticTtree Matching*

The weighting factor δi denotes the importance of node i in the parsing tree. Its value differs for different types of nodes:

- δi=1.2, where node i is either the POS tag VB or NN.
- δi=1.1, where node i is either VP or NP
- δi=1 for all other types of nodes.

We believe that different parts of the sentence have different importance, and the nouns and verbs are considered to be more important than other types of terms such as article, adjective or adverb. We also boost up the nodes of verb and noun phrases, to show their higher priority over other ordinary ones. The weighting coefficient θk for tree fragment k conveys the importance of the tree fragment, whose value is the production of all weighing factors of node i that belongs to the tree fragment k, i.e.,

$$\theta_k = \pi_{i \varepsilon k}\, \delta_i$$

The size of the tree fragment Si is defined by the number of nodes that it contains. The size of weighing factor λ is a tuning parameter indicating the importance of the size factor. The depth of the tree fragment Di is defined as the level of the tree fragment root in the entire syntactic parsing tree, with Droot=1. The depth weighing factor μ is a tuning parameter indicating the importance of the depth factor. The weight of a tree fragment wi is defined as $\theta_i \lambda^{Si} \mu^{Di}$, where θi is its weighting coefficient, Si is the size of the sub-tree, λ is the size weighing factor, Di is the depth of the subtree and μ is the depth weighting factor. If two tree fragments TF1 and TF2 are identical, the weight of their resulting matching tree fragment TF is defined to be:

$$w(TF) = w(TF1)w(TF2)\ .$$

In the STM model above, if two parsing trees employ different leaf wordings or slightly transformed production rules, the tree fragments can hardly be matched. This becomes an evident drawback from the semantic point of view, and it motivates a modification to our original matching model. In order to capture more semantic meanings, we:

(a) allow partial contribution from terminal words if they are shown to be closely related.

(b) Relax the production rules to allow for partial matching.

(c)Use answer matching to bring in more semantically related questions.

Firstly, we use WordNet, a freely available semantic network, to help measure the semantic similarity between two words. We employ Leacock's measure, which uses the distance of the shortest path between two synsets to represent the semantic distance between two words, where the value is scaled by the overall depth of the taxonomy. In order to fit our matching model, in which the semantic score needs to be scaled between 0 and 1, we modify the Leacock's measure into the following:

$$Sem(w1, w2) = 1 - distance(w1, w2)/2D$$

Where, *distance (w1, w2)* is the length of the shortest path between two synsets of *w1* and *w2*, and *D* is the maximum depth of the taxonomy. In particular, we define the path length between two identical words to be 0, i.e., *distance(w,w)*=0, or *Sem(w,w)*=1.

Secondly, we allow partial matching of production rules in the way that two nodes with sufficiently similar production rules can be matched. This sufficiency includes omission or reversion of the modifiers, preposition phrases, conjunctions and so on. For instance, "$NP \rightarrow DT \cdot JJ \cdot NN$" is considered to be similar to "$NP \rightarrow DT \cdot NN$", and can be matched.

The weight of the matching tree fragment TF resulted from matching TF1 and TF2 is defined as:

• $w(TF) = Sem(w_1, w_2)\, \delta_1\, \delta_2\, \lambda^{s1+s2} \mu^{d1+d2}$, if TF1 and TF2 are two terminal nodes w1 and w2;

• $w(TF) = \theta_1 \theta_2 \lambda^{s1+s2} \mu^{d1+d2}$, if the root of TF1 and TF2 are identical and their production rules can be partially matched.

V. ALGORITHMS

A. Cleaning of text

Initially a corpus containing the SMS word and its corresponding cleaned text is to be maintained. This corpus is implemented in the form of a dictionary. The data set (as provided by FIRE) is in xml format. Using Element Tree module in python, we parse it and retrieve the required set of FAQs. Thus we get the noisy input queries and FAQs. A dictionary (can be implemented in python) is constructed from the FAQs. This dictionary is used for cleaning of the inputted noisy words. Each input sentence is now split into words (represented as a sequence). A list of lists is created to store the input word and its corresponding matched cleaned texts along with the measure γ. Each word is now checked for in the corpus. If an exact match occurs then the corresponding cleaned text is returned. Else, this input word's first letter is compared with the first letter of each word in dictionary. If a match occurs, the following steps are followed. Let the matched word be W and the input word be S. If a digit happens to occur in the input word, it is replaced with its corresponding spelling. (i.e. 1 is replaced with one) .Length of the longest common subsequence between W and S is determined. The ratio of length of LCS to the maximum of length of S and W is taken (LCS ratio). Leveinstein distance between the consonant skeletons of W and S is found. (Consonant skeletons-words without any of the vowels). Ratio of the length of the longest common subsequence and Leveinstein distance is considered as the measure γ for knowing the similarity between W and S. A list consisting of the words whose first letter matches with the first letter of S is created. If the input word matches with any of the words in the list, the word itself is returned. Else, the word with highest γ is returned. Each time the returned word is appended to a list (this is the cleaned text). Thus the cleaned text is obtained.

B. Question Classification

Scikit-Learn provides utilities for the most common ways to extract numerical features from text content, namely

1. Tokenizing strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
2. Counting the occurrences of tokens in each document.
3. Normalizing and weighting with diminishing importance tokens that occur in the majority of samples / documents.

In this scheme, features and samples are defined as follows

1. Each individual token occurrence frequency (normalized or not) is treated as a feature.
2. The vector of all the token frequencies for a given document is considered a multivariate sample.

Questions are classified using SVM's [scikit-learn has provided us with the required software. Using the already classified data, SVM is trained. Using all the questions in the archive, a list of words for each domain is generated. Stop words from each question are removed. Each question is represented as a vector using the above word i.e. if that word is present in the question, a 1 is put or else a 0. Thus we get vectors representing each question. Now a centroid point is calculated for each domain using these points. Plane joining two centroid points gives us the plane of separation between two classes. Now our input query is classified using these planes. We get n(n-1)/2 planes in total [If n is the total number of classes. For a given cleaned input query, we check which class it falls into using these planes. The class that gets the maximum number of votes is considered as the domain of the input query.

We have used dictionaries for storing the SMS text and the cleaned text and also for storing the matched word in dictionary with the input and its corresponding measure γ and also used for storing the questions and its corresponding answers. We have used lists for storing the cleaned text obtained. And also for storing the entire set of words that are encountered in FAQs. These are used for matching against the input SMS word.

C. Question Matching using Unigram/Bigram score

We categorize the input questions into in - domain and out

– of – domain questions. When the domain for input question could be determined (given by the user), then a match for question is searched for only in that domain. If the domain for the input questions could not be determined then a match for the question is searched for in the entire archive.

The cleaned text is obtained after running the above algorithm on our input query. The cleaned text is then split into words (stored into a list). Thus each word is represented as a sequence of words. Each word in the input query is matched against each word in the FAQ (if in domain, then only with the questions in that domain as mentioned by the user, if out of domain then with all the questions). If there is no direct match, then we looked up the word in WordNet and obtained its hyponyms, synonyms, etc., and searched each one of these words in the F′ list. If a match was found, then we passed on to next word. Then the unigram score is calculated. The sequence is now divided into bigrams (two consecutive words). The FAQs are also divided into the same. These bigrams are then matched. Then the bigram score is calculated. After the entire set of input queries are matched against the FAQs, Mean Reciprocal Rank (MRR) is calculated. MRR= Mean of the reciprocals of ranks of matched question with the input query. The value of this MRR is returned.

D. *Formation of parse tree*

The input query is first converted into a parse tree using the Stanford parser. The output that we get from this parser is not in a proper tree format. Hence we initially try to convert into a tree format. We use stack to achieve the same. The initial opening braces are pushed into the stack. When a character other than an open braces or closed braces is found, it is considered as a string till it encounters a space and that whole string is pushed into the stack. All elements other than closed braces are pushed into the stack. When a closed bracket is encountered, we pop the elements of the stack till the open bracket is popped and the elements are taken into a vector in the same order. The last element in the vector is made the root node and the other elements are made the child nodes to the root node and the root node is pushed into the stack. Finally the root node will be the last one to be left in the stack.

E. *Matching*

Questions in the archive are also converted into parse tree using the above procedure. These two trees are compared using the weighting scheme of tree fragments (i.e., nouns are given higher weight when compared to verb phrase or noun phrase). This weighting factor is denoted by δ. δ for NN and VB tags is taken to be 1.2,VP and NP tags is 1.1 otherwise 1. The weighting coefficient for a tree fragment i is denoted by θ_k where k belongs to the fragment i. The size of the tree fragment Si is defined by the number of nodes that it contains. The size of weighing factor λ is a tuning parameter indicating the importance of the size factor. λ is taken to be 0.5. The depth of the tree fragment Di is defined as the level of the tree fragment root in the entire syntactic parsing tree, with $D_{root}=1$. The depth weighing factor μ is a tuning parameter indicating the importance of the depth factor. μ is taken to be 0.5. The weight of a tree fragment wi is defined as $\theta_i\lambda^{Si}\mu^{Di}$, where θ_i is its weighting coefficient, Si is the size of the tree and Di is the depth of the tree. $W(TF)=W(TF_1).W(TF_2)$ is the weight of the resulting matching tree fragment. In order to find the similarity score between two syntactic parsing trees *T1* and *T2*, we traverse them in post -order, and calculate the pair-wise *node matching scores* between the nodes in these two trees. This results in T_1*T_2 matrix of M(r1,r2).

The similarity score or the distance metrics between two parse trees is given by

$$\text{Sim}(T1, T2) = \sum_{r1\varepsilon T1}\sum_{r2\varepsilon T2} M(r1, r2).$$

This similarity measure is used for returning the questions which match the input question appropriately.

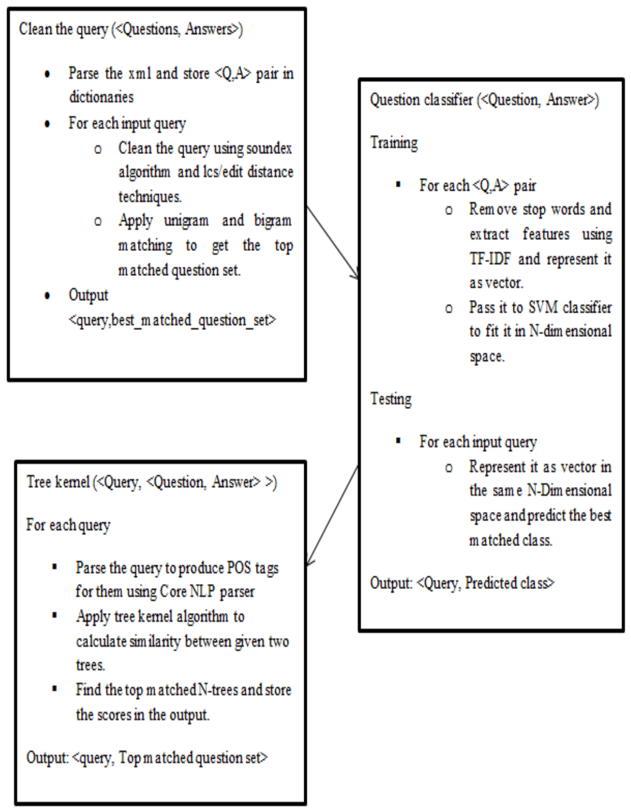
F. *Filling the Matrix*

We use memorization technique to fill the matrix. Each node has the following members: the string, label, depth, a vector to store indices of its children, pointers to its children and a Boolean variable to know if the string is a terminal or not. Labels are assigned to each node in order to represent the nodes in the matrix. Each node in tree T1 is compared against each node in tree T2. The node matching score is then updated in the matrix. Labels are assigned while doing Level order traversal. Depth of each node is calculated and stored as a member (Post order Traversal). η is a measure of the total number of matched tree fragments under that node. This value is calculated by summing the number of non-zero values that got filled in the rows just below the required node. (Filling of the matrix goes from bottom to top). Initially each node is checked if it is a terminal or not. If it is, then the node matching score is calculated using the above algorithm (in Matching). Else, the value for the node is updated using the values for the child filled in the matrix. The node matching score for two nodes is stored in the matrix by Matrix[label1][label2]. Label1 is the label of the node in tree T1 and Label 2 is the label of the node in tree T2. We have used stacks for formation of parse tree. [After we get output from the Stanford parser]. and trees for implementing Question Matching [Syntactic Tree Matching].

The algorithm followed is robust against grammatical errors too. This is because whenever any grammatical error occurs, the chunks or phrases at the lower level are preserved and the ones at the higher level are modified. And weight of the tree fragment at the lower level carry more weight when compared to those at the higher level. And thus the matching score is not degraded much due to such kind of errors.

Semantics are incorporated using Wordnet (freely available semantic network) which helps in measuring the similarity between two words. Hypernyms are determined for each word (input cleaned word and word in the archive).If a match occurs then it is considered that the words are semantically similar.

Sem(w1,w2)=1 in that case. Thus when the weight of the resulting tree fragment is being calculated, it is multiplied with this extra measure which helps in ensuring that the semantics are taken care of.



Clean the query (<Questions, Answers>)

- Parse the xml and store <Q,A> pair in dictionaries
- For each input query
    - Clean the query using soundex algorithm and lcs/edit distance techniques.
    - Apply unigram and bigram matching to get the top matched question set.
- Output <query,best_matched_question_set>

Question classifier (<Question, Answer>)

Training

- For each <Q,A> pair
    - Remove stop words and extract features using TF-IDF and represent it as vector.
    - Pass it to SVM classifier to fit it in N-dimensional space.

Testing

- For each input query
    - Represent it as vector in the same N-Dimensional space and predict the best matched class.

Output: <Query, Predicted class>

Tree kernel (<Query, <Question, Answer> >)

For each query

- Parse the query to produce POS tags for them using Core NLP parser
- Apply tree kernel algorithm to calculate similarity between given two trees.
- Find the top matched N-trees and store the scores in the output.

Output: <query, Top matched question set>

Module Flow of BOW + TK + SVM Algorithm

VI. Results And Analysis

A. Cleaning noisy query using soundex & LCS algorithm

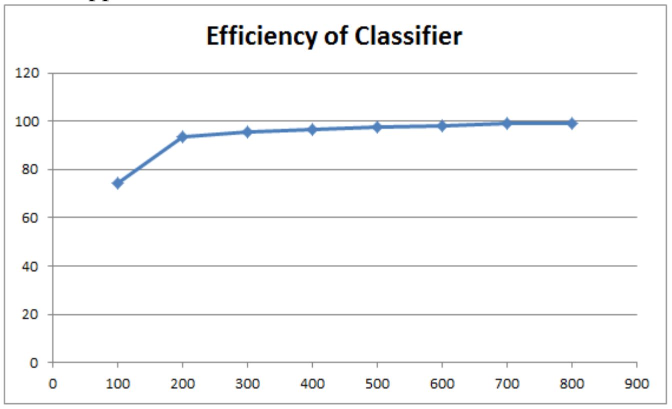
Noisy words and corresponding cleaned texts can be obtained after applying token based Correction and Soundex Algorithm. We observed that almost all of the cases have been handled by our above approach. The observed efficiency of this approach was found to be 89.97%.

Noisy word	Corrected Word
Wch	Which
Contry	Country
Frst	First
Hostd	Hosted
Mdrn	Modern
Olympcs	Olympics

Table: 1.Noisy word and its corresponding corrected word

B. Efficiency of question classifier using SVM

Training data consists of 8400 queries in <Question, Answer, Domain> format. We have trained all those questions using SVM with features like TF-IDF(Term Frequency-Inverse Document Frequency) and BOW(Bag of Word Approach).



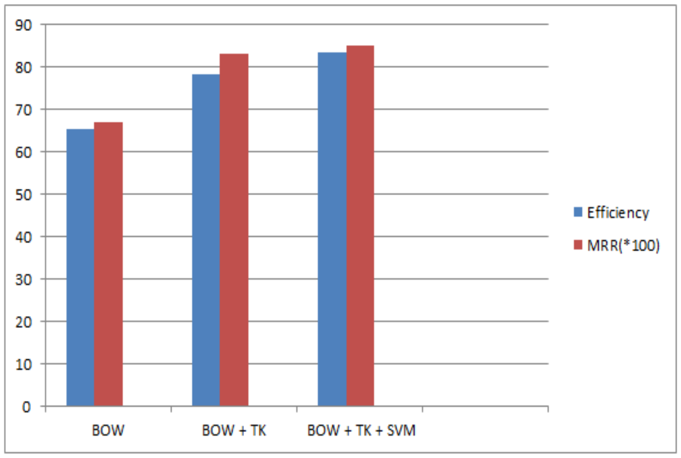
X-axis: Number of testing queries
Y-axis: Efficiency
Graph 1. Efficiency versus sampling steps(number of testing queries) during classification

C. MRR(Mean Reciprocal Rank) Scores and calculation of efficiency for different approaches

MRR:- For an input query, we retrieve top n best matches. if our best matched question for given query is at 3rd position ,then reciprocal rank will be 1/3. if it is at nth position, it will be 1/n. if it's not present in best matched questions, then reciprocal rank will be 0. MRR(Mean Reciprocal Rank) will be calculated as the average of all reciprocal ranks of given queries.

Method	Training Data Set	In-domain Queries (2577)	Out-Domain Queries (5400)	MRR	Efficiency
BOW	8414	1714	2513	0.67	65.4
BOW +TK	8414	2234	3223	0.83	78.4
BOW	8414	2323	3552	0.85	83.6

+TK +SVM					



Graph 2. MRR scores and Efficiency for different approaches
BOW: Bag of Word Approach
TK: Tree Kernel Approach
SVM: Support Vector Machine Approach

Hence we see that Semantic smooth matching (BOW + TK + SVM) performs better over unigram and bigram matching. But Questions like "I am too fat, please help?!!" are actually similar to "In what ways can we lose weight?" our proposed model doesn't handle such cases. Handling such cases is left behind for future work using potential answer matching.

REFERENCES

[1] Danish Contractor ,Govind Kothari, Tanveer A.Faruquie,L.Venkata Subramaninan,Sumit Negi. Handling Noisy Queries in Cross Language FAQ retrieval. EMNLP Proceedings of the Conference on Empirical Methods in Natural Language Processing, Pages 87-96, ACM 2010.

[2] Kai Wang, Zhaoyan Ming, Tat-Seng Chua. A syntactic matching approach to finding similar questions in Community based QA services. SIGIR ‟09 Proceedings of the 32nd ACM SIGIR conference on Research and development in information retrieval, Pages 187-194, ACM 2009.

[3] M. Collins and N. Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14*. Proceedings of the Neural Information Processing Systems Conference NIPS. Pages 625-632.Dec 2001.

[4] M. Collins and N. Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14*. Proceedings of the Neural Information Processing Systems Conference NIPS. Pages 625-632.Dec 2001.

[5] Dell Zhang, Wee Sun Lee. Question Classification using Support Vector Machines. SIGIR ‟03 Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval.Pages 26-32. ACM 2003.

[6] Jiwoon Jeon, Bruce Croft ,Joon Lee. Finding Similar Questions in Large Question and Answer Archives. CIKM ‟05 Proceedings of the 14th ACM international conference on information and knowledge management. Pages 84-90. ACM 2005