Question Answering System for Frequently Asked Questions

Divyanshu Bhardwaj, Partha Pakray, Jereemi Bentham, Saurav Saha
Dept. of CSE
NIT Mizoram
India
(divbhardwaj42, parthapakray, jereemibentham, contact.srvsaha)@gmail.com

Alexander Gelbukh
CIC
IPN Mexico
Mexico
gelbukh@gelbukh.com

Abstract

English. Question Answering (QA) is an important aspect of Natural Language Processing. It comprises building a system that automatically answers questions sought in natural language. Frequently Asked Questions (FAQs) are a set of listed questions and answers concerning a specific topic, which are most likely to be enquired by a user. This paper deals with developing an open domain QA system for retrieving a list of relevant FAQs related to the query issued by the user. Our approach combines the orthodox AND/OR searching with the Combinatorics searching technique which is able to produce an exhaustive list of results for a particular query generated.

Italiano. *Question Answering (QA) un aspetto importante di Natural Language Processing. Si compone di costruire un sistema che risponde automaticamente alle domande cercato in linguaggio naturale. Domande frequenti (FAQ) sono un insieme di domande elencate e risposte riguardanti un argomento specifico, che hanno pi probabilit di indagato da un utente. Questo documento si occupa di sviluppo di un sistema di QA dominio aperto per il recupero di un elenco di domande frequenti pertinenti relativi alla query emesso da parte dell'utente. Il nostro approccio combina l'ortodossa e / o la ricerca con la tecnica di ricerca combinatorio che in grado di produrre un elenco esaustivo dei risultati per una determinata query generato.*

1 Introduction

Question Answering (QA) is an emerging topic in today's world. It is an aggregate of Information Retrieval (IR) and Natural Language Processing (NLP) and is concerned with developing an automated engine which is able to respond to the queries presented by users in natural language.
Frequently Asked Questions (FAQs) represent an effective and efficient way to quickly resolve queries posed by users. They are usually represented as an ensembled list of questions and their answers.

Searching within FAQs can be a tedious task. This becomes even more drawn out when paraphrasing comes into fray. As a result the user is pushed into a maze of questions and answers having to manually look for a particular one as shown in figure 1. It is here that a QA system comes of utmost importance retrieving the particular desired query instantly.

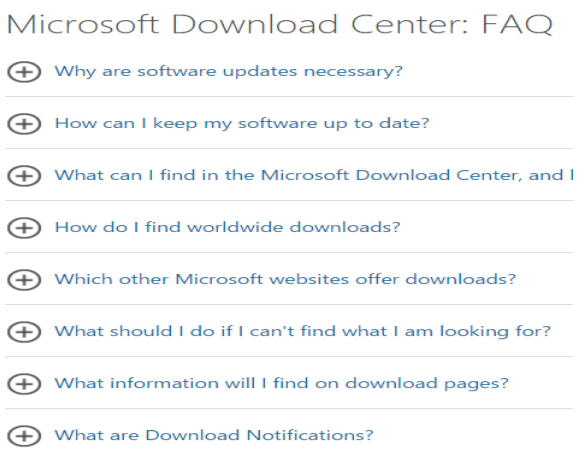

Figure 1: FAQs of Microsoft Download Center

The rest of this paper is organised as follows, Section 2 describes the corpus and its preprocessing, Section 3 describes our system's architecture and the tools used, Section 4 describes the experiment. Section 5 describes the performance of the system, Section 6 analyses the results and Section 7 describes the conclusion and future works.

2 Corpus and Preprocessing

The corpus obtained from the QA4FAQ task website[1] provided us with FAQ in .csv (comma separated values) format, using *;* as separator and in XML format. The CSV file was in UTF-8 format and contained 4 fields viz.,

1. *id:* a number that uniquely identifies the FAQ;
2. *question:* the question text of the current FAQ;
3. *answer:* the answer text of the current FAQ;
4. *tag:* a set of tags separated by ,.

An example of the data provided is given below:

193;Cosa significa AEEGSI?; l'Autorit per l'Energia Elettrica il Gas ed il Sistema Idrico.;acqua, acquedotto, distribuzione, AEEGSI

2.1 Parsing

For the purpose of pre-processing of the training data we developed a CSV parser which could extract the *ID* and the rest of the parts. Development dataset had 406 files with id, question, answer, tag(s). We extracted the question, answer and tags in a file and saved it in the file named *ID.txt*.

2.2 Stopword Removal

In order to increase the efficiency of our input data, we decided to perform stopwords removal. Words which occur in 80% of the documents in the collection are the stop words. However while searching for a list of Italian stopwords, we realised that the existing ones had only 133 to 399 stopwords.[2] [3] [4] So, we merged them and developed our own exhaustive Italian stopword corpus from the existing ones. This corpus[5] had approximately 546 unique stopwords in total. This operation helped us in getting rid of the unwanted words which would hinder the system's performance.

3 System Architecture

The architecture of our system is shown in figure 2.

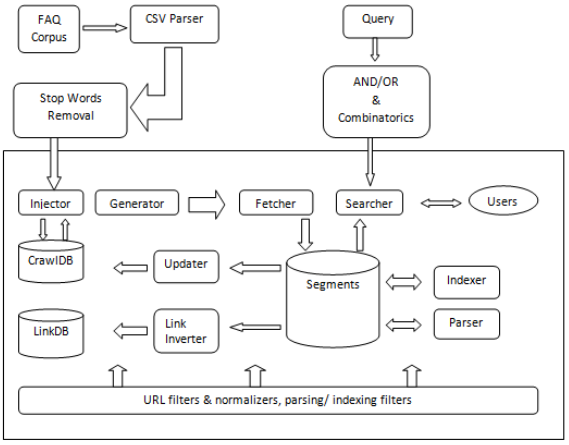

Figure 2: Architecture of the implemented system

The architecture may be divided into two distinct parts as shown in figure. One part contains the architecture of Nutch[6] enclosed in the rectangle. It contains all the basic components essential in the implementation of a Search Engine. The other part represents the aggregation of the searching techniques to be adopted while searching the FAQs. This includes a module that processes the queries obtained for both AND/OR searching as well as combinatorics based searching.The two major steps involved in developing the architecture were *Crawling & Indexing* and *Searching* (described in Section 4).
The steps involved in crawling and indexing are described below:

[1] http://qa4faq.github.io

[2] http://www.ranks.nl/stopwords/italian

[3] http://members.unine.ch/jacques.savoy/clef/italianST.txt

[4] https://github.com/themnd/stopword-it/blob/master/stopwords.txt

[5] The corpus is openly shared in Github for furthur use - https://github.com/SRvSaha/QA4FAQ-EVALITA-16/blob/master/italian_stopwords.txt

[6] https://nutch.apache.org

1. Run a generic Java code taking the ids (taken from *ID.txt*) as the input to generate URL seeds.

2. Injector injects the list of seed URLs into the crawlDB.

3. Generator takes the list of seed URLs from crawlDB, forms fetch list, adds crawl_generate folder into the segments.

4. These fetch lists are used by fetchers to fetch the raw content of the document. It is then stored in segments.

5. Parser is called to parse the content of the document and parsed content is stored back in segments.

6. The links are inverted in the link graph and stored in LinkDB.

7. Indexing the terms present in segments is done and indices are updated in the segments.

8. Information on the newly fetched documents are updated on the crawlDB.

4 Experiments

The corpus obtained after pre-processing was experimented upon by means of various methodologies. A total of 1132 FAQs were available in the test data set. A prototype system was created by feeding the input data into Nutch. We performed two separate runs so as to perform a comparative study between unprocessed and pre processed data.
We used Nutch's own configuration for the Indexing, Searching and Ranking of the data for one of the runs and implemented our own configuration for the other run. The ranking provided by Nutch may be explained using the following equation:

$$score(\vec{q}, \vec{d}) = queryNorm(\vec{q}) * coord(\vec{q}, \vec{d}) * norm(t, \vec{d}) * \sum_{t \epsilon \vec{d}} (tf(t) * idf(t) * t.boost(t.field))$$

Figure 3: Nutch's Ranking Equation

Here,

1. *queryNorm() :* indicates the normalization factor for the query.

2. *coord() :* indicates how many query terms are present in the given document.

3. *norm() :* score indicating field based normalization factor.

4. *tf:* term frequency

5. *idf:* inverse document frequency

6. *t.boost() :* score indicating the importance of terms occurrence in a particular field

Apart from this, we developed our own configuration which was a combination of both the traditional AND/OR search along with the Combinatorics approach. To implement this Combinatorics approach, we split the query by space separator and all possible combinations of a word in query were generated. This is the methodology adopted in subset generation from a given set. So, given n number of words in a query after removing stopwords, we would have $2^n - 1$ possible combinations of query. These were then used for searching by Nutch and ranking was done based on the ranking algorithm we developed. Benefit of this approach was that, it was an exhaustive search and maximum number of relevant results would be retrieved using it using proper ranking algorithm.
This approach could be explained using the following example:
Consider the following query:
numero verde aqp
For this query, all the possible combinations would be created in the following order :
numero verde aqp
numero verde
verde aqp
numero aqp
numero
verde
aqp

From this example we can clearly visualize how this approach would be extremely efficient in retrieving the most relevant answers for queries provided by the user. After applying this approach, we were left with 29 unanswered queries.
We also implemented our own ranking system which ranked the retrieved pages in the following

way :
Consider a query of 4 words. We used a 4 point scale to rank the pages with the highest score being assigned to the page with *4*(number of matches)* Thus, for a query of length *n*, the highest match would be assigned to *n*(number of matches)*. Assuming we have a query of *n* words, all possible combinations i.e, $2^n - 1$ possible queries were to be ranked according to the above mentioned algorithm.
Consider the query following query:
numero verde
and let the text be *il numero verde non verde, un numero che pu essere dipinta di verde*.
Ranking of queries would be done as :
1. *numero verde* : 2*1 = 2
2. *numero* : 1*2 = 2
3. *verde* : 1*3 = 3
Since we get the highest score from the query *verde* so the most relevant document will be fetched by *verde*. Our system retrieved results based on this methodology.

5 Performance

The relevant statistics of both the runs based on the experiments performed are outlined in Table 1.

Run 1		
Total no. of queries	No. of queries answered	No. of queries unanswered
1132	684	448
Run 2		
Total no. of queries	No. of queries answered	No. of queries unanswered
1132	1103	29

Table 1: Statistics of both approaches

As can be inferred from Table 1, while during Run 1 there were a large number of unanswered queries, they were significantly reduced in Run 2. This was possible due to the combinatorics approach used in Run 2. The performance of our system in both the runs is depicted in Table 2.

Runs	Score Obtained
Run 1	0.2125
Run 2	0.0168

Table 2: Performance of NLP-NITMZ in both runs

Systems were ranked according to *accuracy@1*. In this method of ranking the precision of the system was computed taking into account only the first answer generated by the system. The formulation of *c@1* is given as below:

$$c@1 = \frac{1}{n}(n_R + n_U\frac{n_R}{n})$$

Figure 4: Formula for c@1

where:

1. n_R : number of questions correctly answered
2. n_R : number of questions unanswered
3. n: total number of questions

6 Discussion

As the evaluation was done according to *accuracy@1* which considered only the first answer retrieved by the systems, the results obtained weren't extremely accurate. We however managed to implement a search engine which was 97.33% accurate in retrieving queries, which resulted in a trivial amount of unanswered queries. This system conveyed a lot of information which made us realise that combinatorics can be an extremely powerful tool for searching if implemented in a proper way. However, the relevancy of the results obtained would depend on how efficiently the ranking is done.

7 Conclusion and Future Direction

In this paper, we intended to frame an automated Question Answering (QA) system for Frequently Asked Questions (FAQs). We described the preprocessing of the corpus and the experiments performed on them. We also described the combinatorics approach used for searching. While the evaluation results were only decent, we did manage to materialise a remarkably accurate search engine for FAQs. Now that we have an adept search engine we would next endeavour towards perfecting our ranking techniques and algorithms in order to take steps towards implementing a state of the art QA system.

Acknowledgments

This work presented here falls under the research project Grant No. YSS/2015/000988 and supported by the Department of Science & Technology (DST) and Science and Engineering Research Board (SERB), Govt. of India. The authors would like to acknowledge the Department of Computer Science & Engineering, National Institute of Technology, Mizoram for providing infrastructural facilities in order to facilitate research on this task.

References

Annalina Caputo, Marco de Gemmis, Pasquale Lops, Franco Lovecchio and Vito Manzari 2016. *Overview of the EVALITA 2016 Question Answering for Frequently Asked Questions (QA4FAQ) Task*, Proceedings of Third Italian Conference on Computational Linguistics (CLiC-it 2016) & Fifth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2016). Associazione Italiana di Linguistica Computazionale (AILC)

Deepak Ravichandran and Eduard Hovy 2002. *Learning Surface Text Patterns for a Question Answering System*, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)

Lynette Hirschman and Robert Gaizauskas 2001. *Natural language question answering: the view from here*, Natural Language Engineering

Marius Pasca and Sanda Harabagiu 2001. *High Performance Question/Answering*, ACM SIGIR-2001

Narendra K Gupta, Mazin G Rahim, Giuseppe and Riccardi, 2007. *System for handling frequently asked questions in a natural language dialog service*, Google Patents

Partha Pakray 2014. *Yamraj: Binary-class and Multi-class based Textual Entailment System for Japanese (JA) and Chinese Simplified (CS)*, Proceedings of the 11th NTCIR Conference

Partha Pakray and Petr Sojka 2014. *An Architecture for Scientific Document Retrieval Using Textual and Math Entailment Modules*, Recent Advances in Slavonic Natural Language Processing, Karlova Studnka, Czech Republic

Partha Pakray, Pinaki Bhaskar, Somnath Banerjee, Bidhan Chandra Pal, Sivaji Bandyopadhyay and Alexander Gelbukh 2011. *A Hybrid Question Answering System based on Information Retrieval and Answer Validation*, CLEF 2011 Workshop on Question Answering For Machine Reading Evaluation (QA4MRE), CLEF 2011 Labs and Workshop

Pinaki Bhaskar, Amitava Das, Partha Pakray and Sivaji Bandyopadhyay 2010. *Theme Based English and Bengali Ad-hoc Monolingual Information Retrieval in FIRE 2010*, FIRE 2010

Partha Pakray, Pinaki Bhaskar, Santanu Pal, Dipankar Das, Sivaji Bandyopadhyay and Alexander Gelbukh 2010. *JU_CSE_TE: System Description QA@CLEF 2010 - ResPubliQA*, CLEF 2010 Workshop on Multiple Language Question Answering (MLQA 2010)

Pinaki Bhaskar, Partha Pakray, Somnath Banerjee, Samadrita Banerjee, Sivaji Bandyopadhyay and Alexander F Gelbukh 2012. *Question Answering System for QA4MRE@ CLEF 2012*, CLEF (Online Working Notes/Labs/Workshop)

Pinaki Bhaskar, Partha Pakray, Somnath Banerjee, Samadrita Banerjee, Sivaji Bandyopadhyay and Alexander Gelbukh 2012. *Question Answering System for QA4MRE@CLEF 2012*, Workshop on Question Answering For Machine Reading Evaluation (QA4MRE), CLEF 2012 Labs and Workshop

Pinaki Bhaskar, Somnath Banerjee, Partha Pakray, Samadrita Banerjee, Sivaji Bandyopadhyay and Alexander F Gelbukh 2013. *A hybrid question answering system for Multiple Choice Question (MCQ)*, CEUR-WS

Robin D Burke, Kristian J Hammond, Vladimir Kulyukin, Steven L Lytinen, Noriko Tomuro, and Scott Schoenberg 1997. *Question answering from frequently asked question files: Experiences with the faq finder system*, AI magazine

Somnath Banerjee, Pinaki Bhaskar, Partha Pakray, Sivaji Bandyopadhyay and Alexander F Gelbukh 2013. *Multiple Choice Question (MCQ) Answering System for Entrance Examination*, CLEF (Working Notes)

Valentin Jijkoun and Maarten de Rijke 2010. *Retrieving answers from frequently asked questions pages on the web*, Proceedings of the 14th ACM international conference on Information and knowledge management