Question-Answering System

Introduction

Vast amounts of data are available on the World Wide Web. There are a few techniques available to fetch the accurate information from these data. One of the effective way is the Question Answering [QA] Systems. QA system is a computer science discipline within the field of Information Retrieval and Natural Language Processing, which revolves around building systems that automatically answers questions posed by humans in a natural language. [1]

This project centers on adding QA System to Yioop, an open source search engine. The Summarizer in Yioop fetches a short summary from the potentially long documents it crawls. Question-Answering System will extract the information stored in the summary in the form of Triplets [SUBJECT-RELATION-OBJECT] and store it in Yioop so that it can be used to answer questions. Extraction of information from the summary will be done by implementing various functionalities of natural language processing. This triplet will be stored keeping the index efficient for faster retrieval.

In this report, I will be discussing my deliverables for the first semester of this project. In the next section I discuss the test set for the evaluation of the QA System. The second section is on the implementation of the Portuguese stemmer for Yioop system. The third section describes methods of information extraction from sentences and how to generate a parse tree for sentence fragments using a recursive descent parser. Following this I discuss how to extract triplets from the tree generated. Lastly, I have mentioned about the future goals and the road map for the CS298.

Deliverable 1: Suitable test set for the Question-Answering System

A comprehensive dataset plays an important role in testing the effectiveness of QA System. The objective of Deliverable 1 was to create Question-Answers set from the summarizer generated by Yioop. A summarizer is a program that extracts a short summary from a potentially long text document. The Yioop crawler runs a summarizer while processing documents and then index the contents of the summary it produces. It is effective to use this summary to produce a QA system. The information in this summary can be used to answer the queries entered by the users. Not all users would form a question in the same manner even if the answers for those questions are meant to be the same. There are many variations of a single question. An ideal QA system should be able to answer the wide range of questions.

In order to build a data set, I chose Wikipedia summary of different categories to add flavors in the data set. The user base of QA system can fall into any domain, so QA system should be able to work on variety of dataset. I selected summary of Apple's wiki page from technology category, Barack Obama and Narendra Modi from Politics category, Indian Cricket team from Sports category, Roger Federer from a sports personality and Kim Kardashian from the entertainment category.

My intention was to choose a sentence containing enough information from which many questions could be formed.

For example: A statement like "Apple was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne."

One can ask questions like Who found apple?, Who found apple with Steve Jobs?, Who were the founders of apple?, Who were the creators of apple?, Who were the co-founders of apple?, Steve Jobs found which company?, When was apple found?, Apple found in which year?.

To evaluate QA system, I have created a test set, which contains set of summaries that can be the source of information processed by Yioop. Related to those summaries, I have created a set of various types of questions that can be expected from the users. Similarly, I have created a set of expected answers for those questions from the summary that Yioop should provide.

Deliverable 2: Portuguese Stemmer for Yioop System

Stemming is the term used in information retrieval to describe the process for reducing any words to their word stem [7]. For example, stemmer for English words “stemmer”, “stemming”, “stemmed” should be reduced to “stem”.

The goal of Deliverable 2 was to develop a Portuguese Stemmer for Yioop. Snowball [4] is a string processing programming language designed for creating stemmer. They have provided stemming algorithms for many different locales. I followed their stemming algorithm for Portuguese language and implemented it in php. Yioop already has many stemming algorithms developed for the locales like English, French, Italian etc. So the Portuguese stemmer should follow the flow of existing stemmer as well as the Yioop’s coding guidelines.

Portuguese Stemmer

In the Snowball stemmer for Portuguese language, all the processing is mainly based on the region, which are identified in the token. Those 3 regions are as follows:

R1: Region after the first non-vowel following a vowel, or is the null region at the end of the word if there is no such non-vowel.

R2: Region after the first non-vowel following a vowel in R1, or is the null region at the end of the word if there is no such non-vowel.

Example for R1 and R2 region: if **Token:** “beautiful” then **R1:** “iful” and **R2:** “ul”

RV: If the second letter is a consonant, RV is the region after the next following vowel, or if the first two letters are vowels, RV is the region after the next consonant, and otherwise (consonant-vowel case) RV is the region after the third letter. But RV is the end of the word if these positions cannot be found.

Example for RV region: If **Token:** “macho” then **RV:** “ho”

After the identification of region below steps needs to be followed.

Step 1: Standard Suffix remove/replace

In this step there are predefined suffix’s given by Snowball. All rules more or less consist of finding longest suffix from the set and whether it falls in a particular region. Upon finding that suffix, either deletion or replacement is given in the rule.

Step 2: Verb Suffix removal

It has a same structure as given in the step 1 with a set of verb suffix.

Step 3 & Step 4 & Step 5: Residual suffix remove/replace

These steps are post-processing steps where residual removal is meant to be the end character replacement.

In the above algorithm, major challenge is to find the suffix in an efficient way. In order to find the longest suffix from the set, use of regex is the best way to search and remove the suffix. Another challenge that I faced was to identify the special characters. For example there are many Portuguese words like “**bobalhões**” stemmed to “**bobalhõ**”. These characters can change the output of the function like string length because the length of this character is more than one byte which would change the outcome of any logic that relies on the index of character in the word. A solution for this is to use a function that works on multi-byte characters. So functions like **mb_strlen()** and **mbStringToArray()** turns out to be a good choice in this scenario.

Snowball has also provided the data set to test the stemmer. They have given a separate file for both original word and the corresponding stemmed word.

Yioop workflow for the stemmer:

Any stemmer for a particular local can be found at **Yioop > locale > Language code (ex. Pt/) > resource > tokenizer.php**.

Once, the tokenizer is developed, in order to test all the sets given by the Snowball, one needs to configure the corresponding test files. In case of Portuguese Stemmer I added **Yioop > tests > pt_tokenizer_test.php** and placed the test files [**input_vocabulary.txt, stemmed_result.txt**] at **Yioop > tests > test_files > portuguese_stemmer.**

There were a total of 32016 words given in the Portuguese dictionary. The stemmer developed according to Portuguese Stemming algorithm at Snowball was able to parse all the words and stem them correctly.

Yioop Patch Update:

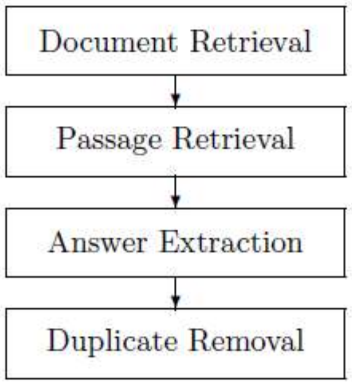
Once the solution works, there are a set of rules mentioned at the official site [3] that needs to be followed before applying this patch to the Yioop repository. My patch is currently applied and the mantis issue id for that patch is: **0000165.**

Deliverable 3: Explore QA system approaches and choose one

To build an effective QA system, there are many approaches available to store the information fetched from the documents on the World Wide Web. The immense amount of freely available unstructured text provides data redundancy, which can be reduced by pattern matching techniques. A processing for a QA system should start from the document retrieval and end when it stores some structure relation from the sentence in the knowledge base. This can be used to answer the questions asked by the users.

For this Deliverable, I looked at 2 different methods explained in the paper presented [3] and [6].

Method 1 presented in paper [3]:



They have presented a pipeline architecture to fetch the answers.

As shown in figure 1, first phase will be the document retrieval, which can be any common method of tf.idf model. There is also a variation given in the document retrieval to cover the query variation. This variation takes advantage of named entity recognition. Example: a name such as “John Fitzgerald Kennedy” can produce legitimate queries involving “John F. Kennedy”, “John Kennedy”, and “Kennedy”, but never “John Fitzgerald” or simply “John”. Another variation is to use the feature of multi word expression that helps identifying word combination. For example “hot dogs”. This multi word expression should be given a preference.

After the document retrieval, passage from the document needs to be extracted. There are many features that help fetching the relevant passage.

Matching word Measure: Sums the idf for word in query and passage

Thesaurus match Measure: Sums the idf for word in query and synonyms of that word in passage

Miss Match word Measure: Sums the idf for word in query but not in passage

Also adding multi-word expression match helps in fetching the relevant passage.

Answer extraction starts once the passage is finalized. The approach would be to find the focus of the question. For example “List journalists that have won the Pulitzer Prize more than once?”. Answer Extraction process should identify journal list as the question focus, and Person as the target type to fetch the exact answer from the passage.

Answer instances extracted from the previous stage typically contains duplicates, which our system removes using a threshold edit-distance measure. Finally, the system computes the number of answer instances to return based on a relative threshold scheme. Each answer candidate is given a score equal to the score of the passage from which it was extracted, and all candidate answers below 10% of the maximum score are discarded. The remaining instances are returned as the final answers.

Method presented in paper [6]:

In this paper they have explained a new technique called as triplet extraction techniques. They proposed QA system architecture which consist of four components:

Natural Language Annotation: Describing contents in machine parsable form

Ternary Expressions: Convert sentence in SUBJECT-RELATION-OBJECT representation

Transformational Rules: Handles the problem of linguistic variation by equating representational structures that have the same meaning.

Collaborative Knowledge Gathering: Taking help of millions of users all over the web to build the knowledge base

From the couple of methods mentioned above, I selected the "Triplet Extraction Technique" and started working on the relevant modules. For Deliverable 3, I implemented a basic structure to generate a parse tree from the sentence, which will create the base for the Ternary Extraction process. In order to create the ternary extraction algorithm explained in the paper it requires a parse tree generated from the statement where each leaf node is labeled with a word from the statement. A Treebank is a text corpus where each sentence belonging to the corpus has a syntactic structure added to it.

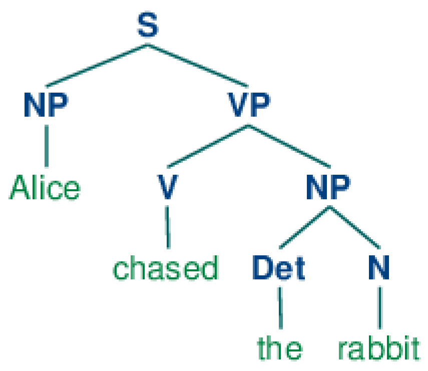
There is an approach given on how to construct the tree from the sentence in the book **Natural Language Processing** [1][2]**.**

In Yioop system, we already have a brill tagger implemented. Tagger basically is a part-of-speech tagging method.

For example: "Alice chased the rabbit" can be tagged as "Alice -> NP chased->V the->Det rabbit->N" where NP is a proper noun, V is a verb, Det is a determiner, N is a Noun. There are set of rules to group this together to form a tree. A basic set of rules is as of follows:

```
NP: {<DT|JJ|NN.*>+}                  # Chunk sequences of DT, JJ, NN
PP: {<IN><NP>}                       # Chunk prepositions followed by NP
VP: {<VB.*><NP|PP|CLAUSE>+$}   # Chunk verbs and their arguments
CLAUSE: {<NP><VP>}                   # Chunk NP, VP
```

To construct a tree for a given statement with a set of rules mentioned above will look like figure x.



Recursive decent parser is the best fit to generate the tree from the tagged sentence. Initially I generated the tree using regex pattern check. Professor suggested that the use of regex may not be a good fit as it is hard to cover variations to parse the tree. So I moved to the recursive decent parser and was able to generate a basic tree with limited set of rules. It can easily cover all the tags with the same structure.

Another complex example that can be covered by the algorithm implemented for the Deliverable 3 for the sentence “A rare black squirrel has become a regular visitor to a suburban garden” is:

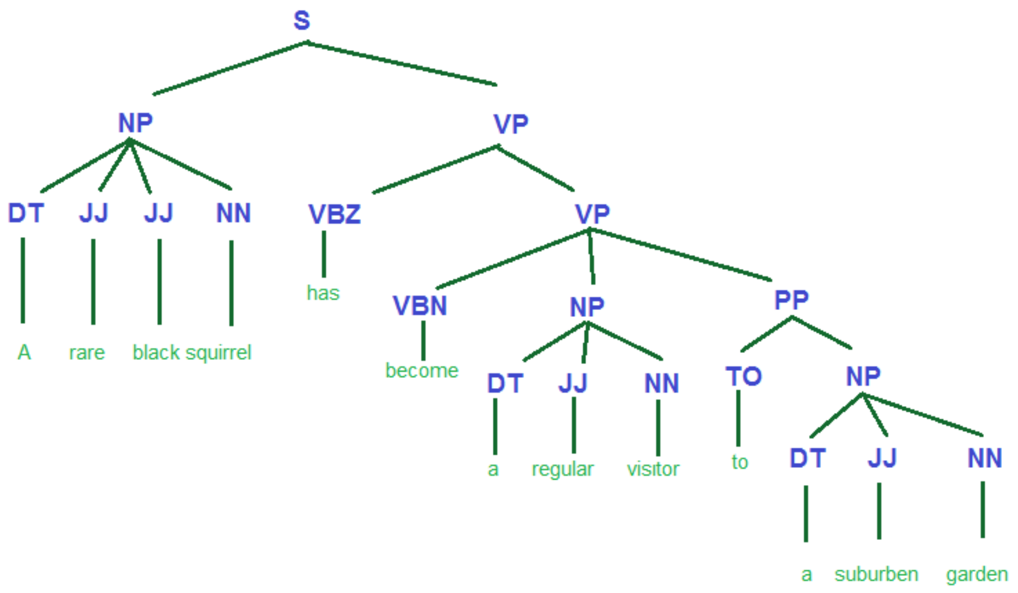


Figure 3: Parse tree generated from the complex sentence "A rare black squirrel has become a regular visitor to a suburban garden.

I tested this implementation on my different sentences. Even if it does not cover some specific scenarios, modifying developed algorithm is flexible to cover many scenarios. There are various approaches that can take this parse tree as an input and extract the triplet which has SUBJCET-RELATION-OBJECT.

Deliverable 4: Basic implementation of Triplet Extract Algorithm

The objective of Deliverable 4 is to fetch triplet of form [SUBJCET – RELATION – OBJECT] from the sentence. The purpose of the triplet extraction is to identify the context in terms of how subject is related to object.

For Deliverable 4, I implemented an algorithm to extract the triplet from the parse tree generated by the recursive descent parser. In order to find the triplet of form [SUBJCET – RELATION – OBJECT], I took the parse tree generated from Deliverable 3 as an input and applied each step explained as below.

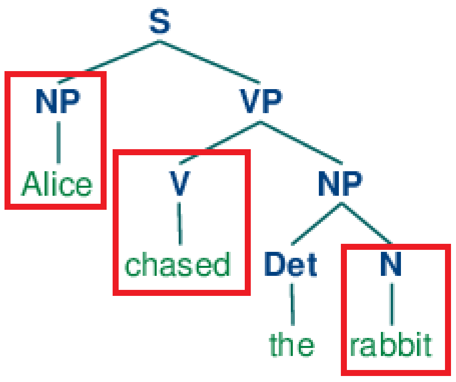
Subject Extraction: To find the subject, we search in the NP subtree. The subject will be found by performing breadth first search and selecting the first descendent of NP that is a noun. They are in the subtree of NN, NNP, NNPS, NNS.

Predicate Extraction: For determining the predicate of the sentence, a search will be performed in the VP subtree. The deepest verb descendent of the verb phrase will give the second element of the triplet. They are in the subtree of VB, VBD, VBG, VBN, VBP, VBZ.

Object Extraction: These can be found in three different subtrees, all siblings of the VP subtree containing the predicate. The subtrees are: PP (prepositional phrase), NP and ADJP (adjective phrase). In NP and PP we search for the first noun, while in ADJP we find the first adjective.

Figure 4 shows the tree generated from my implementation for the following sentence:

"Alice chased the rabbit". It has a brill tagger attached to each term "Alice -> NP chased->V the->Det rabbit->N". After applying triplet extraction algorithm, I was able to fetch the triplet as shown in the image as [ALICE – CHASED – RABBIT].



Storing these triplets from the sentences will help building an organized structure. This structure can be stored in the Yioop system for efficient and faster retrieval.

Conclusion and Future Goals:

During the span of CS 297, I started analyzing the Question-Answering System approaches that were available. I read papers on the approaches of extracting information from the sentences. I implemented small modules like generating parse tree and extracting triplets from the sentences. I also developed a test set to evaluate Question-Answering System. I further added a “Portuguese Stemmer” to Yioop which helped me to understand the workflow of Yioop system.

For the final steps of Question-Answering System, my objective will be to extend the structure for triplet extraction and use these triplets to store it efficiently in Yioop system. This storage of triplets will be useful for the extraction of information at the time of query search. Finally, I will implement an algorithm required to find the objective of the query entered by the user to identify the context of the question.

References:

[1] Bird, S., & Klein, E. & Loper, E. (N.D.). Analyzing sentence structure. *Natural language processing with python – analyzing text with the natural language toolkit* (Version 3.0 ed.,) O'Reilly.

[2] Bird, S., & Klein, E. & Loper, E. (N.D.). Extracting information from text. *Natural language processing with python – analyzing text with the natural language toolkit* (Version 3.0 ed.,) O'Reilly.

[3] Katz, B., Lin, J., Loreto, D., Hildebrandt, W., Bilotti, M., Felshin, S., . . . Marton, G. & Mora, F. (2003). Integrating web-based and corpus-based techniques for question answering. *In proceedings of the twelfth text retrieval conference* (pp. 426-435)

[4] Portuguese stemming algorithm. Retrieved from http://snowball.tartarus.org/algorithms/portuguese/stemmer.html

[5] Question answering. (2015). Retrieved from http://en.wikipedia.org/wiki/Question_answering

[6] Rusu, D., Dali, L., Fortuna, B., & Grobelnik, M. & Mladenic, D. (N.D.). *Triplet extraction from sentences.* Unpublished manuscript.

[7] Stemming. (2015). Retrieved from http://en.wikipedia.org/wiki/Stemming