CARNEGIE MELLON UNIVERSITY QATAR
NATURAL LANGUAGE PROCESSING
FINAL PROJECT REPORT
DECEMBER 24, 2019

Question Generation & Answering

Laila Elbeheiry, Maryam Al-Darwish, Radu Revutchi

ABSRTACT

This report documents our question generation and question answering tool. In Natural Language Processing, many attempts have been made to build question generation and question answering systems. While these two application might have a huge overlap, we address each problem separately. In the first section of this report, we discuss our method and findings for a QG system. In the second section, we discuss our findings for a QA system. The two sections are independent, so readers can skip to the section of interest. In the conclusion, we discuss the overlap between both systems and how can it be utilized for a two-in-one system.

Question Generation

1 Introduction

Among question generation and question answering, generation has been the task that received far less attention. However, this task is extremely useful -especially in the field of education. One particular application of QG is intelligent tutoring systems [?]. The reason question generation is a hard problem is because it falls under the category of the problems that require understanding and generation of natural language. As is the case for many tasks in that category, automatic QG can be based on syntactic relation, semantic relations, or both [?]. We decided to focus on syntactic relations because 1) we didn't want the poor accuracy of most semntic-based NLP tools to affect the accuracy of our systems 2) the state-of-the-art semantic-based QG tool has an accuracy of 86%[?], which -although is a far-fetch from the 70-80% accuracy range that is achieved by syntactic-based systems[?], signifies that semantic approaches still miss out on many questions. Therefore, we decided to attempt at improving syntactic-based QG systems.
Most existing syntactic-based QG systems identify answer phrases by pattern-matching on syntax trees, and then applying general rules to transform these phrases to questions [?][?][?]. What is different about our approach is that: 1) we use constituency parse trees rather than dependency parse trees 2) we enhance our search by using named entities. Overall, our framework for question generation comprises five main steps: preprocessing, filtering, deleting the answer, simplifying, and transforming the sentence.

2 Task Description

Given a Wikipedia article and n (an integer) the tool generates n questions on that article. The system can be easily extended to work on non-Wikipedia text.

3 Framework

3.1 Preprocessing

For preprocessing, we start by using BeautifulSoup[?], to parse the wikipedia article into sentences. In this step we filter out unnecessary texts including tables, figures and references, by only keeping <p> tags.
Then, we use Stanford's CoreNLP deterministic coreference resolution tool [?] to replace pronouns by their referent.
Lastly, we use Stanford CoreNLP's method, annotate, and we include the following annotators: ssplit, parse and NER.

3.2 Question Extraction

The process of question extraction comprises of three stages: identifying potential sentences, identifying answer phrases, and transforming sentences into questions. The question types that our system currently generates are: when, where, what, who, why, how many, and how long for. Adding new different types boils down to identifying rules for each of the three question extraction steps. Hence, it is easy to add all types of questions.

3.2.1 Identifying potential sentence

Identifying potential sentences is achieved by filtering sentences for each question type independently. This filter can be based on named entities, parse trees, or sentence tokens. For example, when_filters filters out sentences that are potential answer phrases for when questions:

Algorithm 1 when_filters

```
1: for <sent in sentences> do
2:     for <mention in sent's entity mentions> do
3:         if mention $\in \{DATE, TIME\}$ then add (sent, mention) to output list
```

One drawback of this filtering algorithm is that it misses out on sentences that don't have explicit date or time. For example, "Dempsey recovered from his knee injury during his stay with his family" will not be used to generate the question "When did Dempsey recover from his knee injury". Appendix A has a description of some of the filters used in our system.

3.2.2 Identifying answer phrases

Once potential answers for a specific question type are identified, we use Stanfod's tsurgeon tool[**?**] pattern-match on the (constituency) parse tree of potential sentences and to delete the answer from the answer phrase if the sentence matches the patterns.
On line 3 in the algorithm, when_filters, we added both the potential sentence and the mention (date or time) in that sentence. That is, we added both the sentence and the keyword to the potential sentences list. In this stage we use keyword to identify and eliminate the answer phrase. The following example, shows how to generate a tsurgeon command for a particular potential sentence:

Algorithm 2 when_generator

```
1: for <(pot_sent, keyword) in potential sentences> do
2:     pattern = "S < PP=head « (IN < (IN [< /in/ | < /on/ | < /In/ | < /On/]) « (CD < /keyword/)
   $.. (/,/=comma $.. VP))"
3:     operation = "delete head, delete comma"
4:     run(pattern, operation, pot_sent)
```

This looks complicated at first glance, but the pattern simply says, "Find a parse tree such that, there exists a sentence that is a parent of a prepositional phrase that's a parent of an element in the set {in, on, In, On} and is a sister of a comma"
The operation says, "Delete the prepositional phrase (and all its children) and the comma".
Running this command on the following sentence:
"In 2004, Dempsey was drafted by Major League Soccer club New England Revolution, where he quickly integrated himself into the starting lineup."
Yields:
"Dempsey was drafted by Major League Soccer club New England Revolution, where he quickly integrated himself into the starting lineup."

3.2.3 TRANSFORMING SENTENCES INTO QUESTIONS

Once we have the answer phrases for every quetsion type, we apply general transformation rules -again, using Tsurgeon- to transform the sentence (with the answer eliminated) to a question. This rule simply reorders the constituents in the parse tree so that the sentence always starts with an auxiliary verb (do, does, was, were, has, have, etc.) followed by the noun phrase, followed by the main verb in the lemmatized form. Additional constituents such as subordinate clauses are removed because they don't add new information to the question.

4 RESULTS

Our accuracy measure is based on the percentage of acceptable questions (acceptable being syntactically and semantically well-formed, and the answer is in the text).
For each of the categories: football, constellations, languages, and movies, we ran our QG tool on five Wikipedia articles, and we yield the following results:

Article/Category	Football	Constellations	Languages	Movies
1	100%	40%	20%	100%
2	70%	50%	90%	90%
3	70%	60%	40%	50%
4	80%	40%	50%	60%
5	*failed*	20%	40%	50%

4.1 CONCLUSION & FUTURE WORK

When run on simple sentences, the tool achieves very high accuracy; however, when run on more convoluted sentences the tool fails to produce well-formed questions. This means that the patterns and the operations that were used for identifying and transforming sentences need to be improved to work well for complicated sentences.

Question Answering

5 Introduction

For question answering, we use general-purpose rules, along with an enhanced maximum-overlap algorithm for finding the answer, given an article and a question. Our framework for question answering has five steps: document preprocessing, lemmatization, including consecutive-pairs, isolating top answer sentences, and extracting the answer.

6 Framework

6.1 Preprocessing

We conducted preprocessing for two reasons: ensuring against memory overflow and removing non-answer sentences. Initially, we run nltk's built-in module for tokenizing a document into individual sentences. We prevented against run-time errors by fixing the bounds of any given sentence. Sentences shorter than 10 characters and sentences longer than 80 words are removed from consideration. This was done to comply with maximum POS-tagging length within the Stanford parser. We also remove any sentences beginning with special characters such as '*'. Following this, we convert each sentence into a list of tokens using nltk's tokenizer.

"*Berlin Orchestra","My name is"
Yields:
"",""

6.2 Lemmatization

We use Stanford's POS-tagger as a supplement to nltk's lemmatizer pipeline. This ensures maximum-possible accuracy in converting all tokens to the stem word. Once we have the lemmatized version of every sentence, we remove all instances of stopwords from nltk's English stopword list.
[I was running all day long.]
Yields:
[run all day long]

6.3 Adding Consecutive Pairs

Preliminary testing showed that lemmatized words were not enough to find an exact sentence match within an article. Additionally, the existence of two consecutive words together proved more important than having those same two words as independent. We therefore included pairs of each two consecutive word pairs into the list representing a sentence. These word pairs are not lemmatized and stopwords are kept.

"The cat eats red apples."
Yields:
[cat, eat, red, apple, the-cat, cat-eats, eats-red, red-apples]

6.4 Isolating sentence containing answer

After passing each sentence through each of the former three pipeline components, we obtain a list representation of each sentence. We disregard duplicate tokens and convert the sentence into a set. After passing the question string through the same process, we obtain the set representation of the question as well. Following this, we take the set intersection of the question and all sentences in the given article. The three largest set intersections in length are kept as the answer sentence. We pass these top three sentences to the next pipeline component: isolating the answer.

"What does the cat eat?"
"The cat eats red apples."
Yields:
[cat, eat, red, apple, the-cat, cat-eats, eats-red, red-apples] intersection [cat, eat, what-does, does-the, the-cat, cat-eat] = [cat, eat, the-cat]

6.5 Isolating the answer

Once we have obtained the target sentence, we run a function to retrieve the question type ('where','when',etc..). Each question type has a separate mechanism for isolating the answer. In the case that the answer is not found, we return the entire sentence itself to make sure our program always outputs an answer. This ensures high recall.

6.5.1 Where, When, Who, How Many Questions

The answer to these question types is a Named Entity (NE) of some sort. The SpaCy library[**?**] offers a total of 18 classes in its NE Recognizer. Some of our answers, however, may fit under several classes. Therefore, we setup a priority class for each question type:

Where: [GPE,LOC,FAC,ORG]
When: [DATE,TIME,CARDINAL]
Who: [PERSON, ORG]
How Many: [CARDINAL,QUANTITY,ORDINAL]

Given the top sentence (out of the three top ones) and a question such as 'where', we first look for instances of 'GPE'. If this fails, we search for instances of 'LOC'. We proceed along this path until all classes for the question type are exhausted. If this is the case, we conduct the exact the same process with the second-best answer sentence (out of the three). If we find an answer, we return it without an extra formatting. However, if no instances of the NE classes exist in any of the three top sentences, we simply return the top sentence.

"Where does the cat eat?"
"The cat eats red apples in Berlin."
Yields:
Berlin GPE

6.5.2 Binary Questions

Here, we work with questions seeking binary answers such as YES or NO. Our method for this is to convert the sentence into a non-lemmatized set of tokens and take the intersection of it with the top answer sentence. If the length of the intersection equals the length of the question minus 1, we return YES. Otherwise, we return NO. We do this because we noticed an overwhelming majority of binary questions simply have the main verb placed in front of the question.

"Is the cat an eater of red apples?"
"The cat is an eater of red apples."
Yields:
YES

6.5.3 What, Why, How, etc Questions

In the case of a what and which question, we use SpaCy's dependency parser to isolate the main verb of the sentence. We then look for the main verb in the sentence and return the remaining part of the sentence. In Why and How questions, we return the top answer sentence itself for simplicity purposes.

"What does the cat eat?"
"The cat eats red apples."
Yields:
"red apples"

7 Results

For question answering, we evaluate based on two metrics. The first metric counts the percentage of questions which isolated the correct answer sentence. The second metric calculates the percentage of questions answered correctly by isolating the answer part itself(without returning the entire sentence itself. We first conduct these metrics on pre-existing questions with their respective articles. These questions are generated by students from previous iterations of 11-411 and can be found in the project.zip package. We also conduct evaluation on two sets of questions generated by our Question Generation system. We expect lower accuracy for this part.

Sentence Isolation/Answer Isolation	Flutes	Berlin	Clint Dempsey	Hindi
Sentence Isolation	82%	77%	80%	88%
Answer Isolation	58%	66%	80%	60%

We actually obtained slightly higher accuracy on our own generated questions as opposed to the student-made ones. It is important to note, however, that the generated questions were predominantly 'when' questions which do not test the full capabilities of our answering system. We also notice that the answering system consistently isolates the answer sentence more than 75% time. This shows that our system is capable of at least isolating the sentence with the answer most of the time. However, isolating the answer proved more difficult and consistently gave us lower percentages than just isolating sentences themselves. When examining the Answer Isolation metric, it is important to note that we did not implement a mechanism to make the output answer fluent. An example is a "Where" question returning "Doha" as an answer instead of "In Doha.".

USAGE

Unfortunately, access to the source code on Github requires permission because this is an undergraduate course project. To request access, kindly contact any of the authors. Usage instructions are also on Github.

CONCLUSION

This report has described our approach to solving two widely known problems in NLP: question generation and question answering. We are approaching the question generation problem by using constituency parse trees, rather than dependency parse trees. In addition to that, we are using the named entities so enhance our search. Also, we are approaching the question answering problem using general-purpose rules and an enhanced maximum overlap algorithm to find the answer.

ACKNOWLEDGEMENTS

We would like to extend our gratitude to Professor Kemal Oflazer for helping us throughout the project and teaching us NLP.

APPENDIX A
FILTER FUNCTIONS FOR QG

Question Type	Filter Description
When	sentences entities ∈ {DATE, TIME}
Why	parse tree includes SBAR starting with "because"
Where	sentences entities ∈{COUNTRY, ORGANIZATION, CITY, LOCATION}
Who	sentences entities ∈ {PERSON}
How long for	sentences entities ∈ {DURATION}
How many	sentences entities ∈ {NUMBER}

2 70% 50% 90% 90%