INTERNSHIP

Automatic question generator.

Introduction

How would someone tell whether you have read this text? They might ask you to summarize it or describe how it relates to your own research. They might ask you to discuss its strengths and weak-nesses or to compare it to another paper.

At first, they might just ask you questions to check that you remember the basic facts. Then, if it is clear that you read more than the title and abstract, they might move on to more challenging questions. Of course, you are probably a highly skilled and motivated reader, and there would be no need to assess whether you read and retained basic factual information. However, that is not the case with all readers. For example, an, bot to be tested by asking questions, generating questions on all the use cases need lot of manual work and can be time-consuming. In this research, we work toward automating that process.

Process of generating questions

Extractions phrase:

Extraction phrase of the question generation is similar to preprocessing the text. Here we concentrate on constructing simple sentences from the sentences with appositives and participial modifiers.

The following are the clauses of sentences where a sentence can be modified to form a simple sentence.

Our algorithm (which is presented later in this section) extracts simplified, entailed statements by removing the following adjuncts and discourse markers:

- Non-restrictive appositives

  (e.g.,Jefferson, the third U.S. President, . . .)

- Non-restrictive relative clauses

  (e.g.,Jefferson, who was the third U.S. President, . . .)

- Participial modifiers

  (e.g.,Jefferson, being the third U.S. President, . . .)

- Temporal subordinate clauses

  (e.g.,Before Jefferson was the third U.S. President . . .)

The above sentences can be simplified through the below algorithm in ***extractor.py***

Function name : remove_appos(tokens_tag,text)

Arguments : tokens_tag is array of tuples and each tuple is a word in the sentence and the POS tag_name associated with it.

1. Eliminating appositive comma (' , ') if present after nouns which have tag ‘NNP’.

2. Eliminating relative adverbs and pronouns by identifying them by the POS tags namely WRB for adverbs and WP for pronouns.

3. The above two steps are repeated for all the sentence.

4. Sentences obtained after this phrase are simple sentences.

Pronoun resolution

If extracted statements contained unresolved pronouns, they would almost always lead to vague questions since the question appears outside of its original discourse context (e.g., When he was elected President of the United States?).In order to avoid these vague questions, we replace pronouns with their first-occurring antecedent in the discourse (i.e., in the input text). The intuition behind this approach is that the first an-tecedent is likely to be a specific and informative noun phrase since it was the first mention of the entity in the discourse. This simple method seems to work fairly well, although the first mention is not always the most informative (e.g., as in the sentenceAn avid reader, Susan is always at the library).

1. Identifying the parse tree node in the original input sentence that corresponds given pronoun.
2. Using coreference graph for finding the first use of the pronoun.
3. Replace the pronoun with original noun.

Challenges in pronoun resolution:

- Nested pronouns.
- Original noun with an appositive.
- Possessive pronouns.
- Marking as unresolved if we cannot find antecedent.
- Considering only the first pronoun of sentence for replacement.

Nested pronouns : when the first mention of an entity contains mentions of other entities—we recursively replace mentions. For example, from the last sentence in John likes cars. His car is expensive. It has leather seats., we extract John's car has leather seats.
Possessive pronouns: If a pronoun is possessive, then the replacement will also be made possessive if necessary by adding **'s.** Example: John's car.

Question creation

This system mainly generates two types of questions boolean questions which has the answer as True or False. The second type is WH questions which has the answer as a reason or an explanation.

Algorithm for generating boolean question ***yes_or_no.py***

Function name : yes_or_no_ques(text,tokens_tag)

Arguments : tokens_tag is array of tuples and each tuple is a word in the sentence and the POS tag_name associated with it.

1. We will find the presence of a helping verb in a sentence by POS tags, a word becomes a helping verb if it's POS tag is VBD, VBZ or MD and it should be followed by another main verb identified by it's POS tag.
2. If the helping verb is present the flag variable hv_flag is set with the index of the helping verb.
3. If hv_flag is set then move the word present at the index pointed by hv_flag to the start of the sentence.

   Eg: Margery was accused of cheating.

   Question: Was Margery accused of cheating

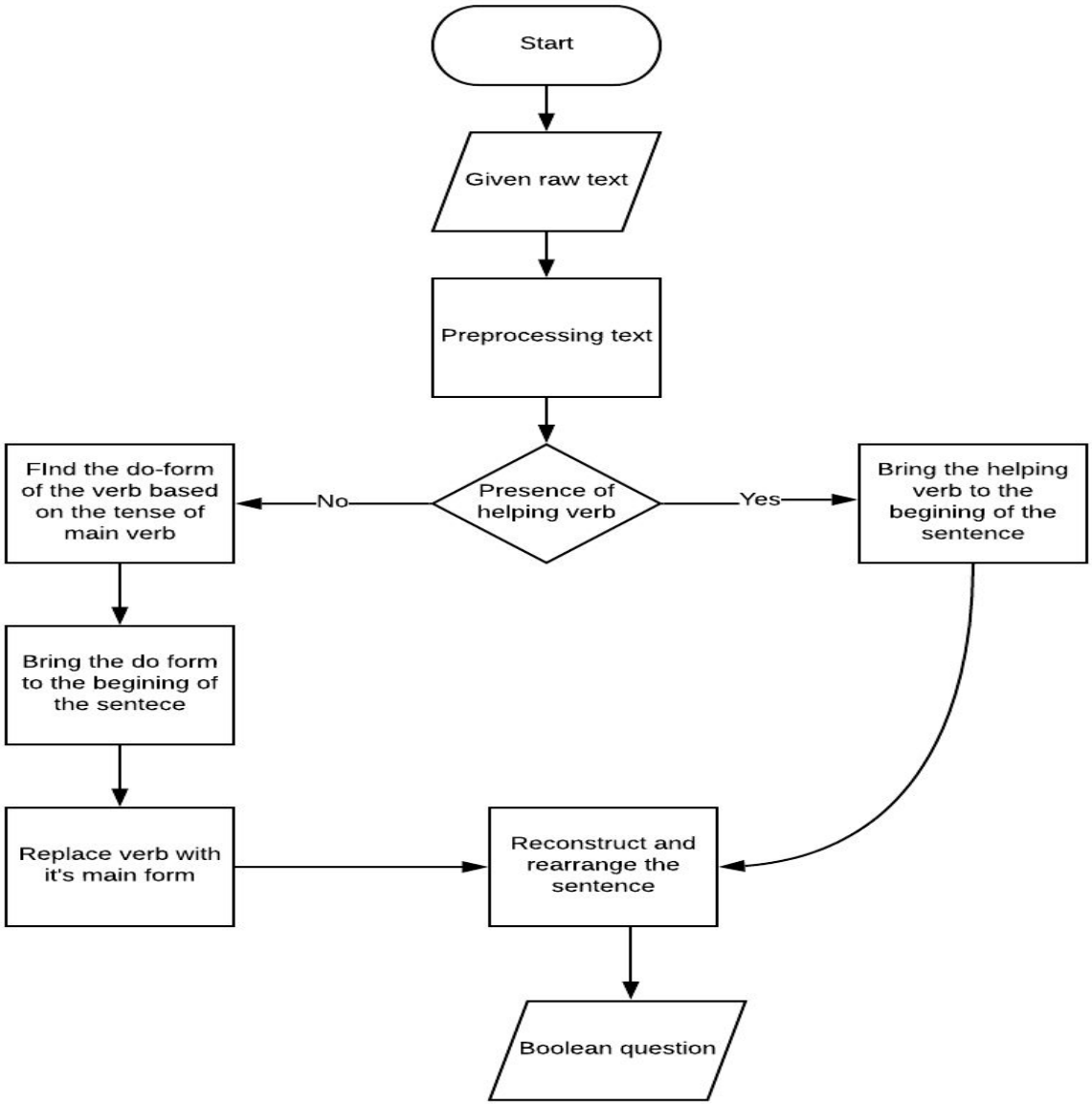
4. If the helping verb is not present then identify the do-form of the main verb and move the do-form to the beginning of the sentence and replace the verb with its main form using lemmatization.

5. We identify the do-form of verb by its tense like the past tense of a verb is denoted by ‘VBD’ POS tag.

Eg: They ran away.

Question : Did they runaway ?

Flow for generating yes or no questions



Function name: reconstruct(list)

Arguments : Takes a list of words.

The function capitalizes the first letter of the question and joins each word which is followed by adding a question mark at the end of the sentence.

Function name : identify_subject(text,tokens_tag)

Arguments : tokens_tag is array of tuples and each tuple is a word in the sentence and the POS tag_name associated with it.

Subject can be a noun or a pronoun and if we find any greetings or phrases we will skip them until we encounter a subject and we will use the next part of the sentence for generating question.

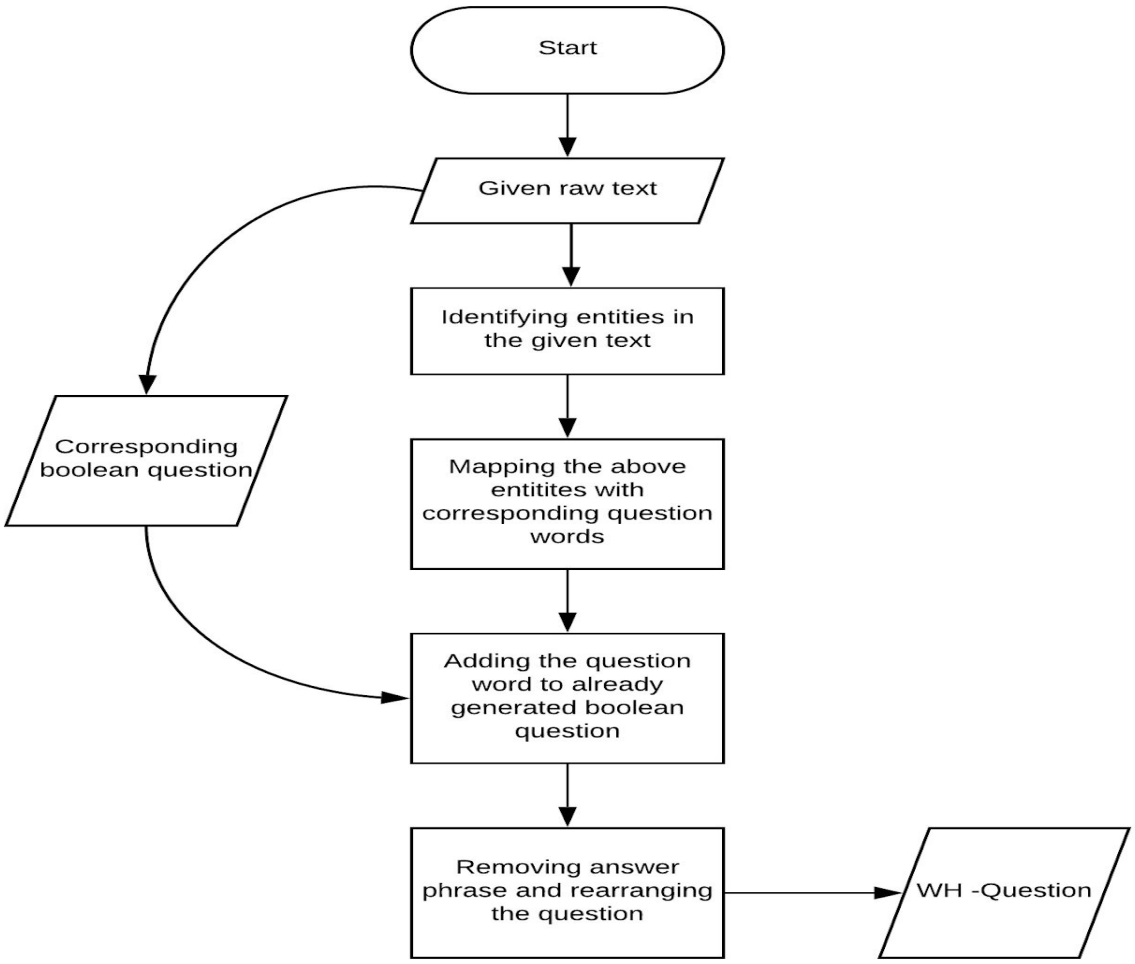
Algorithm for generating wh-questions ***General_ques_gen.py***

Generations of WH questions requires type of the entity which we consider as an answer and we map corresponding question word associated with it.

Steps:

1. Identify the entity types in a sentence using NER models.
2. For each identified entity in extract ‘Wh’ word respective to entity type.
3. Wh word mapper must be constructed based on the position of entity in the sentence.
4. If the entity is subject then just replace the entity with it’s Wh word.
5. Else find relative pronoun of the respective entity type and add it to the yes or no question generated form the sentence.
6. Rearrange and reconstruct the sentence.

Flowchart for generating wh-questions.



NER models (Named Entity Recognizer)

Identifying entities require entity recognizers here we used hybrid of stanford NER which contain a classifier which can detect 4 classes namely person, location,organization and percent whereas the other classifier which can detect 7 classes namely person, time, location, organization, date, precent, money.

The reason behind using this hybrid classifier is the 4 class entity recognizer is working great for basic entity types whereas 7class entity recognizer is working good for complex entity types like time, date. So the idea is to extract entity types from different classifier and make a list of entities based on the classification of both classifiers. The advantage of this is though one classifier not recognizes the entity type the other may do it and we can have it in the final result.

Command line statements to run the classifier and store the result in a file:

For 4class entity recognizer:

java -mx600m -cp "*:lib/*" edu.stanford.nlp.ie.crf.CRFClassifier -loadClassifier classifiers/english.conll.4class.distsim.crf.ser.gz -textFile sample.txt >4class.txt

For 7class entity recognizer:

java -mx600m -cp "*:lib/*" edu.stanford.nlp.ie.crf.CRFClassifier -loadClassifier classifiers/english.muc.7class.distsim.crf.ser.gz -textFile sample.txt >7class.txt

Function name: entity_recognizer( )

This function loads the output files obtained by running above NER models and process all the recognized entities into list of tuples where each tuple has the name of the entity and its type.Here the entity and its type based on two classifiers.

Function name : decompose(text, tokens_tag, ent_type)

Arguments : tokens_tag is array of tuples and each tuple is a word in the sentence and the POS tag_name associated with it. ent_type consists of list of tuples where each tuple has the name of the entity and its type.

STEPS :

1. For each recognized entity we check if it has a known entity type, if it has then we proceed for generating question.
2. For adding particular question word we have a dictionary which maps the entity type to particular question word.
3. Here the mapping have two types ,if the entity is present in the subject of the sentence then just replacing the entity with question word makes a question.
4. Whereas if the entity present in the object part of a sentence the rearranging of the sentence needs to be done.
5. We add this question word to the already generated boolean question and we remove the answer phrase corresponding to the question word.
6. Rearranging and reconstructing of sentence should be done.

Function : group_ind(text,ent_type,prev_type,place)

Arguments : ent_type consists of list of tuples where each tuple has the name of the entity and its type. prev_type tell us the previous entity type which is pointed by place variable.

If there is a consecutive words of same entity type eg: 26th july 1999 all the words [26th , july, 1999] are of same type date and should be considered as one single answer phrase and the we should not generate the multiple questions.

About Cisco question generation:

Program name **cisco_ques_gen.py**

Question generation of cisco is same as the generic question generator but we replace the entity recognizers of stanford with cisco NER.

About Cisco NER

1. As it is impossible to get all the entities of raw text, we proceed with the use case of Cisco documents.
2. We can train an existing NER with entities in Cisco documents.
3. In this use case we trained a NER which identify 18 entity types ,a few of them are : Product, Switches, Configuration, Capabilities etc.
4. These identified entities are now mapped with respective question words.

Deficiency in the question:

1. Ungrammatical - not a valid english sentence.
2. Does not make sense (or) Irrelevant.
3. Wrong 'Wh' word selection.
4. Formatting - Capitalization, punctuation.

Question ranking:

Model - Logistic regression

$$p(U \mid q, t) = 1 - p(A \mid q, t)$$

U - Unacceptance.

A- Acceptance.

q- For a given question 'q'.

t- generated from text 't'.

Final distribution function

Conditional probability is considered as acceptance doesn't depend on one factor.

$$p(a \mid q, t) = \pi\ 1\text{-}p(a_i \mid q, t)$$

$1 <= i <= k$ (k- no of factors of acceptance)

There can be many factors to determine the acceptance of the question,few of them are:

Relevance of the question,Grammatical sense etc.