Project Progress Report

Weidong Yuan `weidongy`

Shunyuan Zhang `shunyuaz`

Yazhi Gao `yazhig`

Zijun Shi `zijuns`

1 Overview

We summarize the current status of the project, including work that has been done in preprocessing, ideas in question generation step, and progress in question answering step. We also report the problems we encountered.

2 Progress

Our work in the first stage has largely focused on preprocessing, which provides a good basis for question generation and question answering.

2.1 Preprocessing

We describe what we have achieved and how we implement tasks in preprocessing. For each task, we state how the output can be saved for further use in question generation and question answering steps.

- Raw Input Cleaning and Tokenization
  Input:*.htm text files. Preprocessing:python library such as BeautifulSoup. Output:Structured text (tokenized sentence) with html tag. This step can be used in question answering step to locate the paragraphs containing candidate answers. For example, retrieve paragraphs under tag "club career" for a question asking relevant information.

- Named Entity Detection and Extraction
  Input: tokenized sentences. Output: Extracted Named Entity with IOB tag and NE type tag. We use NLTK's NE chunk function combined with tokenization and Part-of-Speech tagging to detect and extract Name Entity. We store extracted NE in python tuple. This step can be useful for both question generation and answering, for example, to generate or answer a "who" question for an extracted person NE.

- Extending NE Extraction Task: Relation Extraction
  Input: tree-structured sentence and detected NE. Output: a 3-element python tuple that specifies the relation between any two NEs. This step can be used in question generation and question answering step for generating or answering a question that can be described as a logical form with 2 NEs.

2.2 Question Generation

- Who, Where, When, What Questions
  For 4W-type (who, where, when, what) questions, we begin with extracted relation tuple. For any detected NES and their extracted relation (the 3-element tuple), we generate a question by removing any element of the relation tuple. When the removed element is the relation, we further rephrase the question by substituting it with its synonym (using WordNet). Taking tuple ('Dempsey','married to',Bethany Keegan') as a simple example, we remove the relation "married to" and generate the following candidate questions:1. "Who is Dempsey married to?"; 2. "Who is Dempsey's spouse?".

- Yes/No questions
  To generate a Yes/No question, we move the modal verb to the front of the sentence and add a question mark at the end. This can be achieved via transducer template solutions. For example, with the sentence "Johnson will go to the theater tomorrow.", we can generate the question "Will Johnson go to the theater tomorrow?".

2.3 Question Answering

2.3.1 Question Preprossing

Since our system is based on a relatively small set of documents, the question preprocessing is important.We have read some papers about the this module and figured out what methods we can use.
1.Question formulation: we will simply use WordNet to expand the query term. In addition, we also apply some hand-written rule to rephrase the question to make it like a declarative sentences. All the result sentence will be passed to the IR system.
2.Question classification: Ideally, we would like to classify questions into different categories, which improves the efficiency in retrieving paragraphs in a document, in particular for structured documents (e.g., wikipedia pages) that follow certain pattern. For instance, the information about marriage, family members, and religious are usually given in "Personal life" part. If we could identify that a question regarding "religious" or "son's name" belongs to the category "Personal life", then we would quickly locate corresponding paragraphs (by reading html tag). When evaluating a candidate answer, we could also give more weight (score) to a candidate appearing in related paragraphs. In addition, The question classification will also help us use the related set of templates to extract the answer from the IR result. We are going to build a small question type table. And use simple regular expression rule to classify the question.
Difficulty: for documents in each set, what is the best way of summarizing question categories?

2.3.2 Candidate Sentence Retrieval

Currently we have implemented the candidate answer sentence selection by information retrieval. We use apache solr and its python bindings to index the document by sentences and return the candidate sentences based on statistical retrieval models for question queries. The results are satisfying. The right answer sentence basically appears in the result sentences.

2.3.3 Answer Processing

Our IR system will return a set of candidate sentences from the documents. The Answer Processing will help us to extract the exact answer for the question. Basically, we will apply the answer-type pattern extraction. We use a Learing Surface Text Pattern Algorithms(Ravichandran 2002) to build our templates. Every question type will have its own templates set, using the template, we can easily get the exact answer. The pattern learning algorithm has its own limitation. For the condition which it cannot be applied, we plan to use N-gram tiling. These are the algorithms we have learned so far. We may still have some improvement in the future.

3 Problems

3.1 Parametric Retrieval

It is hard to find an optimal parameter set for the retrieval model. The return precision is somehow not very stable because the shallow statistics are based on short text. We may need to train based on the corpus to find a candidate parameter set.