MIM QA System: Deliverable 2

Antariksh Bothale
abothale@uw.edu

Julian Chan
jchan3@uw.edu

Yi-shu Wei
yishuwei@uw.edu

Abstract

We introduce the preliminary work done on MIM, our Question Answering system.

1 Introduction

This document reports the creation and performance of an end-to-end Question Answering system. We have a basic setup in place for the overall task and present current baseline results.

2 System Overview

Our system architecture can be broken down into three major classes - MainFacilitator, TaskExecutor, and Session. Our information retrieval system processes queries and retrieves answers in 3 steps, namely Document Retrieval, Passage Retrieval, and Answer Processing. Each of these steps are encapsulated in their own class and derives from the TaskExecutor task. The Session object is used to pass information between these tasks. The implementation of each TaskExecutor is completely transparent to the MainFacilitator, whose job is to simply run the TaskExecutor(s) in the correct order and pass a Session object between them. The QuestionProcessor object is embedded in the Session object and contains methods that allows the TaskExecutor(s) to access the user query.

During start-up, MainFacilitator instantiates the three TaskExecutor(s) and put them in an array. When there is a new query and the AnswerQuestion() method is called, MainFacilitator creates a new Session object and calls Execute() on each of the TaskExecutor(s) sequentially. Barring any execution error, the answer will be written to the Session object.

The SystemEvaluator reads in all the question from the data set and calls MainFacilitator's AnswerQuestion() function. It then reads the answer from the returned Session object and writes to the output file.

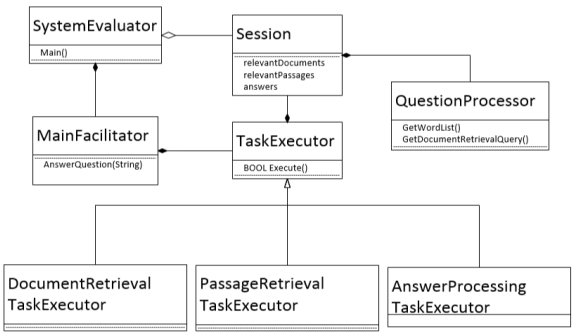

Fig 1: class diagram of the MIM QA system

3 Approach

3.1 Question Processing

Our current question processing module only removes the *wh-* words in the question and feed the rest of the question text as a bag of words to the document retrieval modeule. In the next deliverable we would like to do some machine learning-based question classification so that in the downstream we will be able to verify whther the answer is suitable for the specific question type.

3.2 Document Retrieval

The Document Retrieval Subsystem takes a document retrieval query from the session and populates the session with a list of hits from the indexing and retrieval system, as described below. This list of documents is then used for Passage Retrieval and Answer Processing.

For the purpose of indexing and searching the documents, we use Whoosh [2], which is pure-Python based text indexing, search, and spell checking library. The choice of a Python based library above alternatives such as Lucene and Indri/Lemur was motivated by its ease of integration and use with our Python code-base, and its functionality being comparable to that of its Java counterparts.

3.2.1 Indexing

The Beautiful Soup XML parser [1] was used to parse all the documents and extract the Document ID, Headline and Body Text. Whoosh's in-built indexing engine was used to index all the documents present in the corpus. The current system uses Whoosh's default indexing mechanism, which does not process the text, we plan to improve on this by incorporating tokenization and stemming. The indexing schema was designed to store the Document ID and Headline with the Index, while the Body Text was indexed but not stored as it would have caused unnecessary duplication of data. Instead, the text is retrieved from the document using the Document ID whenever needed. This index is currently stored on Patas.

3.3 Passage Retrival and Answer Extration

In this deliverable we combine passage retrieval and answer extraction into one step. This step reads the files reported as relevant by the document retrieval module, extracting all passages in the documents that contain the keywords (the words in the question). A "passage" in our current system is simply defined as any chunk within the document with no more than 250 characters. These passages are then scored based on the density of keywords in the passage. The passage extraction and scoring function are taken from Whoosh, which are originally used for returning snippets for search results. The final answers of our current system are the top three passages.

In the future version of our system, we would like to split out an separate answer extraction module so that it will generate short answers from the passages and score them based on whether they are valid answers to the question type and their relevance. We would also like to improve the passage retrieval part by implementing our own passage scoring function.

4 Results

Under the baseline system, we obtain an aggregate score of 0.0466 under the Strict evaluation scheme, and an aggregate score of 0.0595 under the Lenient evaluation scheme. One reason we got such low scores is that we did not notice there is a `text` attribute in the parent node of the questions (the `target` node) which contains the main subject of the questions. Our search was based on the question text only, and thus missed the most important information.

5 Discussion

The baseline system has been put in place and works end to end, but since it uses fairly naive and rudimentary processing methods at each step of the chain, the results are far from satisfactory. The next few weeks on the project would now be devoted to improving each of the segments.

Acknowledgments

References

[1] *Beautiful Soup*. URL: `www.crummy.com/software/BeautifulSoup/`.

[2] Matt Chaput. *Whoosh*. URL: `https://pypi.python.org/pypi/Whoosh/`.