MIM QA System: Deliverable 2

Antariksh Bothale
abothale@uw.edu

Julian Chan
jchan3@uw.edu

Yi-shu Wei
yishuwei@uw.edu

Abstract

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.

1 Introduction

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.

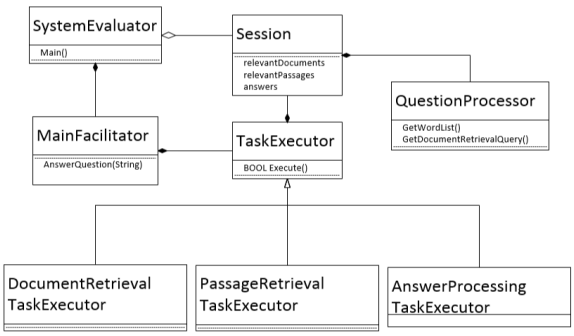

Fig 1: class diagram of the MIM QA system

3 Approach

3.1 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[**whoosh**], 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.

3.1.1 Indexing

The Beautiful Soup XML parser[**bsoup**] 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.

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.

5 Discussion

Long papers may consist of up to 9 pages of content, plus two extra pages for references. Short papers may consist of up to 5 pages of content, plus two extra pages for references. Papers that do not conform to the specified length and formatting requirements may be rejected without review.

Acknowledgments

The acknowledgments should go immediately before the references. Do not number the acknowledgments section. Do not include this section when submitting your paper for review.