David Schulz

CS 4980 001

4/17/2022

**NLP Project Background**

Both the knowledge database and questions get tokenized and lemmatized to perform keyword extraction. Part-of-speech tagging and syntactic parsing can then be used to determine the answer type. Many words can often represent more than one type, so other words in the question need to be considered. A lexical dictionary such as WordNet can be used for understanding the context.

Once the question type has been identified, an information retrieval system is used to find a set of documents containing the correct keywords. A tagger and NP/verb group chunker can be used to verify whether the correct entities and relationships are mentioned in the found documents. Selecting documents with tf-idf tables based on unigrams and bigrams is also a possible approach. Only the relevant paragraphs are selected for ranking.

A vector space model can then be used for classifying the potential answers. Transformer models are used to convert paragraphs, sentences, and words into vectors. It will check if the answer is of the correct type as determined in the question type analysis stage. An inference technique can also be used to validate the potential answers. A score is then given to each of these according to the number of question words it contains and how close these words are to the question. The more and the closer the better. The answer is then translated into a compact and meaningful representation by parsing.

NLTK’s functions can split both the knowledge database and questions into tokens, remove stop words, and then stem all remaining tokens. nltk.word\_tokenize() can tokenize the sentences. nltk.corpus has a stopwords module that contains a list of stop words that should be removed. nltk.stem has a LancasterStemmer, PorterStemmer, and SnowballStemmer that can stem the tokens. The LancasterStemmer is the most aggressive stemmer while the SnowballStemmer is a slightly more aggressive version of the PorterStemmer. Various transformer frameworks and models, such as word2vec, BERT, GloVe, and Universal Sentence Encoder, can be used to convert sentences and tokens into vectors. Some modules, such as scipy and sklearn, have cosine similarity functions that can be used to compare the question’s vector with vectors from the knowledge database.

A paper titled “Closed Domain Question Answering System Using NLP Techniques” by Mukesh Raghuwanshi presents implementation methods and experimental results with analysis for a closed-domain QA system which handles documents related to the education system. Although Mukesh doesn’t name specific tools and frameworks he used to create his system, he describes the techniques and architecture well enough to figure out what he might have used. His system’s architecture begins with a corpus of questions and a knowledge database of domain-related documents. Both are tokenized, have stop words removed, part-of-speech tagged, and stemmed. The resulting lists of tokens from both are considered the extracted keywords. The keywords from the knowledge database are compiled into an “index term dictionary” which is a table containing two columns: word and file names. Word is an extracted keyword and file names is a list of the files which contain that keyword. He then performs document retrieval by matching the keywords from the question with the keywords in the index term dictionary. Only documents which contain all of the question’s keywords will be selected to potentially contain the answer. He then reranks the selected files using the Jaccard similarity function. To perform the final answer extraction, he uses the POS tagging from before and finds the string that matches the question’s POS tags the closest.

A paper titled “Closed Domain Question Answering for Cultural Heritage” by Bernardo Cuteri is about a novel architecture for closed-domain question answering and a possible application in the cultural heritage context. Bernardo also doesn’t name the specific tools and frameworks used, but he describes the techniques and architecture. The system is split into five tasks: question processing, template matching, query expansion and contextualization, query execution, and answer creation. First, the question is tokenized and tagged with part-of-speech (POS) tags. Then a natural language parser extracts grammatical relations (typed dependencies) from the text. Questions are then classified and transformed into formal queries by template matching. In this context, templates represent the structure of typical questions. If a certain template is matched, things can be inferred about the question type. Every question template is accompanied with a formal query in which some slots are empty and are filled with terms extracted from the question that matches the template. Sometimes, to be effective, the query must be expanded with context information and/or word semantic information. His solution is to expand the query by using synonyms, hypernyms, and other word semantic relations. In his model, the query is executed against a structured knowledge base. Query results can then be used to build a natural language answer with a mechanism like template matching, but in the inverse direction.

A paper titled “AQUA: A Closed-Domain Question Answering System” by Maria Vargas-Vera and Miltiadis Lytras describes AQUA, an experimental question answering system. The authors also don’t name the specific tools and frameworks they used, but they describe the techniques and architecture very well. AQUA combines NLP, ontologies, logic, and information retrieval technologies in a uniform framework. The architecture is split into four parts: user interaction, question processing, document processing, and answer extraction. During user interaction, the user inputs the question and validates the answer. Question processing is performed in order to understand the question asked by the user. This “understanding” of the question requires several steps such as parsing the question, representation of the question, and classification. For document processing, a set of documents are selected, and a set of paragraphs are extracted. This relies on the identification of the focus of the question. In the answer processing phase, answers are extracted from passages and given a score. The answer with the highest score is returned to the user.

For the sake of time and simplicity, my system will be based off of the first paper, “Closed Domain Question Answering System Using NLP Techniques”, the most. Given more time, I would have taken a more complex approach with an ontology like AQUA. Like the first paper’s approach, my system will tokenize and stem the knowledge database, but instead of creating an index term dictionary, it will create a tf-idf table, where the rows are all stemmed tokens that exist in any of the files and the columns are the files. The question will also be tokenized and stemmed, and then the top three files that are most likely to contain the answer will be chosen by getting all rows in the tf-idf table where their token is in the question and calculating the sum of those rows by column. The top three columns are the three files. Once the files are chosen, they will all be read and compiled into a list of sentences. Instead of matching the question with the best answer by POS tags like the first paper, those sentences and the question will be embedded into vectors using a sentence transformer. Then the question vector and each sentence vector will be compared using a cosine similarity function. The sentence with the highest cosine similarity will be chosen as the answer.