The beginning of Watson

Farcas Alex

Frunza Andrei

Ghinea Mihai Emilian

Orz Nichita

Pasc Daria Nicola

Project description

- Default Project:

Implementing a Question and Answer system based on Information Retrieval with the help of Woosh for indexing and retrieving relevant pages from Wikipedia based on the clues from Jeopardy. The evaluation and improvement of the system is done trough metrics, error analysis and suggesting improvement methods for performance and accuracy.

- How does it work?

We are indexing the content of the Wikipedia pages, each page appears as a separate document in the index. We prepared the terms for indexing by applying tokenisation and stemming on them. After this process is done, we will search for the title of the document based on the clue that we get and we will return it as a result to the question. This way we measure precision at 1. We also return a list of the first 10 results and check if the answer appears in the list, this way we can check if we were close to the answer and it is another relevant metric.

- Measuring performance:

We will use two metrics, P@1 because we want to check how precise the algorithm is and also First10, to check if the answer appears in the first 10 results. The last one indicates how close we were to the answer and we can also determine why other titles came before the correct one.

- Error analysis:

We will analyse the errors for each performance, focusing on the best performance, and we will categorize the errors.

- Improving retrieval:

We will use AND/OR operators between tokens, take into consideration the category of the question and integrate Open AI API.

Code Analysis

Structure

- The code in main.py file has a menu structure when you run it, from which you have 4 consecutive options

1) create_index_value = Y ==> create_index() method is called from methods.py file. Inside this method the Wikipedia file is decomposed in documents, with the title and the content. The content is processed and tokenised and the stemming is applied, eliminating stop-words. The resulted documents are inserted into an index file.

2) category_value (Y/N) ==> Decide if you want to use the category of the question for the search query

3) AND_OR_value (AND/OR) ==> Decide if you want to use AND or OR logic operators between tokens. The AND option was very inefficient, accuracy was 0%-5% with all the improvements, so we eliminated it and only kept the OR option between tokens, this value is set by default.

4) chatGPT_value (Y/N) ==> Decide if you want to use ChatGPT API in your query for a retrieval improvement. If Y, the method call_chatGPT() is called. Take into consideration that you need to set a valid key in the api_key variable if you select this option.

- For a more in depth analysis and documentation of the code, open the main.py and methods.py files, where the methods are described and the code is commented.

Code Analysis

Performance Measure

- We use P@1 for measuring the performance of the system because it provides a quick and early assessment for the system's performance and has a strong influence on the user perception, due to first impression.
- Since P@1 might not capture the overall effectiveness of the system, we are also using the P@10 metric. Together, they provide a more comprehensive evaluation of the system. You can determine how often the result is the first one on the list and also how often it appears in the first 10 recommendations.

Error Analysis

- Success factors: Best case scenario we obtained P@1 = 0.22 and P@10 = 0.38. The best case scenario results from the fact that the documents, the questions and the categories were tokenised, taken trough the processes of stemming and eliminating stop words. We used the OR operator and the category. The OR operator means that not all the tokens have to match, it is enough for some of the words from the question to match some words from the document.
- Error categories:

1) Semantic and synonyms variations – the system fails in understanding context.
2) Specificity and uniqueness – Lack of distinctive keywords leads to missed matches

Results

AND_OR_value = AND, category_value = N, chatGPT_Value = N => P@1 =0, P@10 = 0.02

AND_OR_value = OR, category_value = N, chatGPT_Value = N => P@1 =0.15, P@10 = 0.37

AND_OR_value = AND, category_value = Y, chatGPT_Value = N => P@1 =0.03, P@10 = 0.05

AND_OR_value = OR, category_value = Y, chatGPT_Value = N => P@1 =0.22, P@10 = 0.38

Note: We noticed that for AND operator the results are too small, so we set OR operator as default and we focused on improving it using the ChatGPT API.

AND_OR_value = OR, category_value = N, chatGPT_Value = Y => P@1 =0.19, P@10 = 0.39

AND_OR_value = OR, category_value = Y, chatGPT_Value = Y => P@1 =0.25, P@10 = 0.40

We noticed that the precision increased when using ChatGPT, the most significant one being for the OR operator being 25% precision if we also have the category taken into consideration.