Data Mining – Jeopardy Project

1. Introduction

In the captivating realm of game shows, few have captured our attention quite like Jeopardy!, where contestants' general knowledge is rigorously tested through a series of challenging questions. We've all been there, attempting to summon our inner trivia expert, yet often finding ourselves falling short. But what if there was a way for a program to outsmart us in this intellectual arena?

The secret is found in a simple tactic: to use the vast collection of human knowledge stored inside the Wikipedia pages. This idea suggests that, in most cases, the answer to a Jeopardy! question can be found within the title of a corresponding Wikipedia page. The premise is enticing – by constructing a broad index of all Wikipedia entries and adeptly searching through it, we may discover a methodical strategy to win the Jeopardy! Game.

2. Indexing and retrieval

- **Preparing the terms for indexing:** We used ***CoreNLP*** for the pre-processing of the terms, namely for lemmatization. In the context of our Jeopardy project, lemmatization is preferred over stemming as it ensures that the base forms generated are valid words, aligning more closely with the linguistic nuances inherent in Jeopardy questions. The precision offered by lemmatization, considering the grammatical context and part of speech, contributes to a more accurate and contextually aware question answering system.
    - ***What issues specific to Wikipedia content did you discover, and how did you address them?***

      The files are not consistent. We have noticed that not only the titles are marked between "[[...]]", but also the publishers. In order to identify the titles, we checked that a line starts with "[[" and ends with "]]".

      e.g. : publisher=[[The Press-Enterprise (California)|The Press-Enterprise]]

      Another issue we had related to the Wikipedia files, was that some documents were empty. We solved this issue by skipping these pages, since they were messing up our processing.

- **Indexing terms:** We used ***Apache Lucene*** for the indexing of the terms. Lucene uses an inverted index, by mapping a term to the titles of the documents that contain it. We use Lucene's ***Analyzer*** to clean and filter the text data, by removing common stop words and making all words lowercase. After cleaning the data, the Analyzer converts the text into tokens that are added to the index. The inverted index then uses these tokens as lookup keys to find the documents which contain the searched tokens.

- **Retrieval:** For query searching, we used the ***QueryParser*** and ***Searcher*** from Lucene. For query building, we concatenate the clues to the question. We then lemmatize all the words from the content using ***CoreNLP***. Originally, we were not applying the lemmatization step on the query, and the performance was quite bad, which is understandable since the searcher was searching for raw words in

an indexed that only contained lemmatized words. Thus, most meaningful words were likely not matched and it ended up matching on shorter, less relevant words which have the same form in the lemmatized form as in the raw form. After lemmatizing the query, we noticed a significant improvement in the results.

- **Are you using all the words in the clue or a subset?**

  We are using all the words in the clue, all words being indexed as the rest of the words, after we have excluded the sequence "Alex: ", which is the name of the host.

- **Are you using the category of the question?**

  We are using the category normally, marking it similar to any other words. We did not feel the need to add any other weight, since it already works well. What we did notice is that, once it runs out of pages similar to the content of the query, the other returned pages are related to the category of the question.

  e.g.: for a question related to the Nile and Cairo, the system returned as a first answer Cairo (which is the correct answer), and then was followed by the "Geography of Egypt" and "Demographics of Egypt", and then results related to African cities category.

- **Parallelization**: An important challenge we came across was the time it took to build the index. Using a basic preprocessing that splits the text into words based on delimiters and removes stop words results in a very time efficient index building, of roughly 2-3 minutes. On the other hand, adding lemmatization is extremely time consuming. Hence, we decided to parallelize the index creation by distributing the preprocessing and the indexing of the files across threads, using Java's *streams*.

3. Measuring performance

MRR (Mean Reciprocal Rank) assesses the effectiveness of a system by calculating the reciprocal of the rank of the first correct answer in the list of retrieved answers. MRR is suitable for this Jeopardy project because the primary goal is to prioritize accuracy in quickly retrieving the correct answer. It focuses on the first correct answer, which aligns with the contest's nature, where contestants aim to respond promptly.

We are also interested in the percentage of the time that the system finds the correct answer in the top 3 results per query, as well as the top result. For these, we use the metric ***Precision at k*** twice, with k = 1, and k = 3.

- MRR = 23.18% -> on average, the system finds the correct answer in the top 5 results (1/0.23 = 4.31).
- P@1 = 17% (0.17)
- P@3 = 24% (0.24)

4. Error analysis

Originally, we had a poorer performance because the document titles in our index were processed in a more complex way than the question answers. Hence, even though the answers were correct, the system could not tell because of the different formatting. After resolving this issue, our metrics slightly improved.

An issue we noticed in our system is that clues are sometimes not interpreted according to their importance. For instance, for the question "The Naples Museum of Art", our system doesn't find the

correct result, Florida, but instead yields a list of museum or art related answers (Maryhill Museum of Art, Colin Center for the Art etc.). This can be explained by the weight of the clue in this case, which provides important additional instructions: " We'll give you the museum. You give us the state.". While we do use the words from the clue in our query, the system simply looks for matches of words and doesn't truly understand the semantics of the question. Since the word "museum" occurs in both the question and the clue, it makes sense that we get many museum results and the state is missed.

5. Improving retrieval

The approach we followed to improve the retrieval system is based on a Deep Learning approach. What we have done is that we have taken a pre-trained ***NLP model from Hugging Face*** which is specialized to provide sequence embeddings for sequences of a length up to 512 that can be later used for an information retrieval task. This process had 2 major steps.

1. Indexing the embeddings of all sequences from the Wikipedia dataset into a vector database (***Qdrant***) and save as a payload/metadata for each embedding the corresponding title.
2. Do inference with each question from the questions file and find the most similar embedding of the query from the vector database using a cosine similarity metric and retrieve the payload of the most similar embedding as an answer.

We noticed an improvement in the metrics computed above by using the new DL-powered solution.

Metric	Lucene Index Retrieval	Deep Learning
MRR	23.18%	25.84%
P@1	17%	23%
P@3	24%	28%

6. Conclusion

In summary, our information retrieval system has made commendable progress in addressing linguistic nuances, parallelizing the indexing process, and delivering results that align with the user queries. While there's room for improvement in the metrics, the positive components demonstrate a commitment to quality and accuracy, providing a strong basis for ongoing refinement and optimization. As we will continue to iterate and enhance the system, these achievements will contribute to the development of a robust and effective information retrieval solution.