Data mining project

Watson

Hincu Alice-Ramona

Herlea Stefan-Alexandru

Gherghel Vlad-Zeno

1. Introduction

In this project, we created a system that was modeled after IBM Watson's approach to answering to Jeopardy questions. The system's main goal was to build an extensive database of information using Wikipedia data. Our system focuses on parsing Wikipedia pages, indexing them in Whoosh, and using natural language processing methods for query handling. Using metrics such as Mean Reciprocal Rank (MRR), the system assesses its performance on Jeopardy-style clues with the goal of identifying the most accurate responses.

1.1. How to run project

First of all, install python (it works both with 3.8, 3.9 and 3.11).

Second of all, you will need to clone the project using the following command (make sure to check out to the branch master after the project is done cloning):

git clone https://github.com/AliceHincu/IBM-Watson

git checkout master

Then, you have to install the packages that the program needs to run it (if you don't already have them) using this command in the terminal:

```
pip install Whoosh nltk
```

Now you can run main by either using an IDE like PyCharm or with the command

```
python main.py
```

2. Indexing and Retrieval

The Indexing phase of the system involves parsing Wikipedia pages to extract relevant data, such as titles, content, and categories. This information is then organized into a structured format suitable for indexing. Utilizing Whoosh, a Python indexing and search library, the system creates an efficient search index that allows for rapid retrieval of information.

During the Retrieval phase, the system processes user queries, typically Jeopardy-style questions, by analyzing and interpreting the natural language of the query to identify key search terms. It then searches the Whoosh index for relevant Wikipedia pages that could contain the answer. The system employs natural language processing techniques, including tokenization and lemmatization, to enhance the search's effectiveness and accuracy. The goal is to retrieve the most relevant documents that answer the query, demonstrating a practical application of information retrieval principles inspired by IBM Watson's capabilities.

2.1 Description of code

The code is grouped in 5 directories:

- index – this is the directory that contains every file related to the creation of the whoosh index. It contains the following files:
    - **index_creation.py** - creates a search index from Wikipedia pages. Contains a function that defines the schema for the index with title, content, and category fields. It then reads through all files in the given directory, parses each file to extract page data, and adds this data to the search index. The function begins by clearing any existing index in the specified path and then creates a new index. For each Wikipedia page, it extracts the title, content, and categories and adds a document to the index. If a page is a redirect, it adds an entry for the redirect target as well.
    - **utils.py** - contains helpful methods like reading all files from a directory and preprocessing file content (removing tags).

    - **WikipediaPageParser.py** - A parser class to extract information about wiki pages from a text file. Reads the file at the initialized file path and processes its content into WikipediaPage objects.
    - **WikipediaPage.py** - A class representing a Wikipedia page. It contains the title, the category and the content.
    - **ParseResult.py** - A class to encapsulate the results of parsing Wikipedia page data. It contains a set of WikipediaPage objects that have been processed and a dictionary mapping titles of pages that are redirects to their target titles.
- performace – this directory contains the script for evaluating the implemented system.
    - **compute_mrr.py** - computes the Mean Reciprocal Rank
- query – this directory contains the logic for running the questions using our program.
    - **query_runner.py** - executes a search query against the index using a given category and clue, comparing the results against a specified answer. Here are also methods that use synonyms for the construction of the query, but using them gave a lower score.
    - **question_reader.py** - a class that reads the questions from the given file.
- wikipedia_pages – directory that contains the input data for the index
- indexdir7 – directory where the index was saved (after the seventh try the index worked)
- **main.py** - point of entry for running the program. You can either create the index, answer the questions, or exit.

2.2 Answering questions from requirements

1.Describe how you prepared the terms for indexing (stemming, lemmatization, stopwords, etc.).

```
schema = Schema(
    title=TEXT(stored=True, analyzer=custom_analyzer),
    content=TEXT(analyzer=custom_analyzer),
    category=KEYWORD(commas=True, scorable=True, stored=True)
)
```

We have used a custom analyzer created with the following filters:

- **RegexTokenizer:** Splits text into tokens (words) based on a regular expression, typically separating words by spaces or punctuation.
- **LowercaseFilter:** Converts all tokens to lowercase to ensure case-insensitive matching.
- **StopFilter:** Removes common 'stop words' (like 'the', 'is', etc.) from the tokens, as specified by the ENGLISH_STOP_WORDS list, to reduce index size and improve search focus.

- **Lemmanizer:** Provide lemmatization capabilities for tokens during indexing or querying. It initializes with a WordNet lemmatizer and processes each token, applying lemmatization to the token's text.
- **CharsetFilter:** Applies character normalization, like removing accents using the accent_map, to improve matching tokens with diacritics or special characters to their plain text equivalents.

This set of filters is designed to improve the efficacy of searches by normalizing text data and increasing the probability that queries will match documents with different word forms, case, and character encoding.

As for the schema, the title and content are altered, while category is not. Usually, the title and content fields are altered through various techniques such as tokenization, case normalization, stop word removal, and stemming. This is so because the language used in these fields is usually very diverse and directly related to the searches that users conduct. By processing them in this manner, the noise in search results is decreased and various linguistic forms of a word are matched.

2. What issues specific to Wikipedia content did you discover, and how did you address them?

The first problem that we encountered was when were initially parsing the Wikipedia files, the regex that we were using was incorrect. At first, we tried to parse the files by sections (title and content), using a regex pattern that was trying to match “[[Title]]”, but not only the title was put in between the square brackets, but also “[[file:…]]”, “[[File:…]]”, “[[image:…]]”, “[[Image:…]]”. After we realized that, we tried using a regex that uses only the left side square brackets, but again, there were cases of “[[file:…” and “[[File:…” that had no right closing square brackets. In the end, we stopped using regex for page sectioning. We then observed that the Wikipedia pages were split between each other by 3 spaces in a file.

Another problem we encountered was the existence of certain tags inside the content of Wikipedia pages that were of no help to us. These tags are the following: “[tpl]…[/tpl]”, “[ref]…[/ref]”, and URL links.

The last thing we noticed was that certain pages only had redirects to other pages (“#REDIRECT”), so to solve this we decided to create a dictionary which maps page titles to their redirect targets.

3.Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset? If the latter, what is the best algorithm for selecting the subset of words from the clue? Are you using the category of the question?

We have noticed that certain questions from the “questions.txt” file had extra text after the category, which was specified between brackets, for example: “STATE OF THE ART MUSEUM (Alex: We'll give you the museum. You give us the state.)”. This was solved by removing the text surrounded by brackets.

The clue is preprocessed with the same analyzer as the content from the Wikipedia pages so we are not using all the words, while the category is left untouched. Lemmatization is preferred over stemming for this project because it involves reducing words to their base or dictionary form in a way that considers the context, leading to more meaningful word forms. Unlike stemming, which often chops off word endings to achieve a base form, lemmatization understands vocabulary and morphological analysis, producing linguistically accurate lemmas. This accuracy is crucial for interpreting Jeopardy clues, which often contain complex language that requires understanding the specific meaning of words in context. Lemmatization helps in matching queries with relevant content more effectively by using these accurate, contextual forms, enhancing the system's ability to retrieve the most appropriate answers.

We tried to add synonyms to widen the search's scope. However, this approach led to an unexpected outcome: the performance did not improve as anticipated. The most likely explanation is that the query may have been oversimplified when complex queries were transformed into a simpler version by synonym substitution. This procedure may have removed specific information or subtle context that was necessary to correctly match the search query.

Using both the clue and the category in queries increases the search precision by taking advantage the additional context provided by the category. This approach helps to narrow down the search results to more relevant documents, especially in a vast dataset like Wikipedia, where clues alone might return a broad range of topics. The category basically acts as a filter.

The final form of the query looks like this:

CLUE OR CATEGORY

Even though the query is simple, it obtains good results, which will be detailed in the next chapter.

3. Measuring performance

For evaluating our Jeopardy system, Mean Reciprocal Rank (MRR) is a suitable metric because it's designed for systems where the goal is to retrieve a list of items and the relevance is binary (relevant/not relevant). MRR is especially relevant when you are interested in the rank of the first correct answer in the list of returned answers. It's calculated as the average of the reciprocal ranks of the first correct answer for each query. This metric effectively measures the system's

ability to return relevant results as close to the top position as possible, which aligns well with the Jeopardy system's objective where typically only one answer is correct.

The Mean Reciprocal Rank (MRR) achieved for the top 10 search results is **0.688**. When expanding the search to the top 100 results, the MRR slightly decreases to 0.648, maintaining the same number of correct answers. This indicates a strong performance in retrieving relevant answers within the first few results, with a modest decline in effectiveness as more results are considered. Top 300 000 results leads to a MRR score of 0.512.

Right now, the limit in the code is set at 10. For those that are not in the top 10 search results, we add a 0 to the MRR score. The table outlines the distribution of correct answers based on their rank for each run, indicating the frequency at which answers appeared at different rank positions:

Rank	Nr of answers
1	24
2	4
3	1
4	3
5	1
6	0
7	1
8	2
9	4
10	1
N/A	59

4. Error analysis

1.How many questions were answered correctly/incorrectly?

The system answered 24 questions correctly and 76 questions incorrectly

2.Why do you think the correct questions can be answered by such a simple system?

The success of such a simple system in answering Jeopardy questions can be attributed to multiple factors:

- the rich and structured nature of Wikipedia content, which provides comprehensive coverage of topics that Jeopardy questions often touch upon.

- employing techniques like lemmatization and the use of categories helps, which refine search queries to match relevant articles effectively.
- the strategic implementation of the Whoosh index, by carefully designing the schema to include fields like title, content, and category.

3.What problems do you observe for the questions answered incorrectly?

The incorrect answers can be categorized into these main groups:

- **Similar but Incorrect**: These are cases where the system retrieves answers that are related to the query but not precisely what the question asks. This might occur due to the broad matching of terms without capturing the question's specific context or nuance.
    - For the clue "In this Finnish city the Lutheran Cathedral also known as Tuomiokirkko", Finnish Orthodox Church is the second answer returned, but the fifth answer, Helsinki is actually the one that is correct. his Finnish city query illustrates how related but incorrect answers can surface due to the system's inability to discern the specific significance of "Lutheran Cathedral" and "Tuomiokirkko" directly pointing to Helsinki. Instead, it retrieves "Finnish Orthodox Church" as a closely related term.
- **Loss of Crucial Information Due to Preprocessing**: Certain preprocessing steps, like over-simplification or incorrect lemmatization, can strip away essential context or details from the clue. For instance, reducing "US" to "U" alters the meaning entirely, leading the search to focus on irrelevant documents. This category highlights the limitations of the preprocessing strategy in retaining the semantic integrity of the queries. Example:
    - The first clue from the file, "The dominant paper in our nation's capital it's among the top 10 US papers in circulation", was converted to " dominant paper nation capital among top 10 u paper circulation". In the top 10 results, we can find foreign newspapers like Svenska Dagbladet, which is a Swedish newspaper.
- **Contextual Misinterpretation**: This category involves cases where the system's interpretation of the query does not accurately capture the essence or specificity required, often due to a lack of contextual understanding.
    - For example, in searching for "The Georgia O'Keeffe Museum," the system might miss because it does not connect the specific museum's name with its geographical location, New Mexico, despite related terms being present in the index.

To address the identified categories of errors:

1. Similar but Incorrect Answers: We should implement more sophisticated natural language processing (NLP) techniques, such as named entity recognition (NER) and context-aware models like BERT, to better understand the specific context and nuances of each query.

2. Loss of Crucial Information Due to Preprocessing: We should fine-tune the preprocessing steps to retain essential information. For specific known issues (e.g., "US" reduced to "U"), we need to create exception rules or use more advanced parsing techniques that recognize and preserve meaningful acronyms and initialisms.

3. Contextual Misinterpretation: We should enhance the index with metadata or context-rich annotations that provide additional information about each document, enabling the system to make more informed matches based on the query's context. Incorporating machine learning models trained on similar types of queries and documents can also improve contextual understanding and relevance of search results.

5. Improving retrieval

1.What is the performance of your system after this improvement?

For improvement, we have decided to let ChatGPT 3.5 reorder the ranking of the results returned by the query if the answer is not found on the first position of results. Because we have used the web interface of ChatGPT, we have extracted only a few questions/statements and measured the rate of improvement on those.

The prompt used in talking with ChatGPT is the following:

“For the following statement:

<question/statement>

Please rearrange the following list of possible answers such that on the first position is the true answer. Take as many breaks as you need. Thank you.

<list of top 10 findings from our query>”

For the statement “Phnom Penh's notorious gridlock is circumvented by the nimble tuk-tuk, a motorized taxi that's also known as an auto this, a similar Asian conveyance.”, which has as correct answer “Rickshaw”, the initial list of answers that our application returns are:

- Olympic Stadium (Phnom Penh)
- Transport in Cambodia

- Norodom Buppha Devi
- History of Cambodia
- Economy of Cambodia
- Cycle rickshaw
- Mekong
- Telecommunications in Cambodia
- Rickshaw
- Khmer language

After asking ChatGPT to rerank the list such that the first position contains the answer, the rearranged list was:

- Rickshaw
- Cycle rickshaw
- Transport in Cambodia
- Economy of Cambodia
- History of Cambodia
- Telecommunications in Cambodia
- Khmer language
- Olympic Stadium (Phnom Penh)
- Norodom Buppha Devi
- Mekong

As seen from the above result, the list was correctly rearranged, and the first position contains the correct answer.

For statement “Not to be confused with karma, krama is a popular accessory sold in cambodia; the word means "scarf" in this national language of Cambodia”, with correct answer “Khmer language”, the initial list is:

- Politics of Cambodia
- History of Cambodia
- Economy of Cambodia
- Khmer language
- Siem Reap
- Cambodia National and Provincial Resources Data Bank
- Norodom Buppha Devi
- Party of Democratic Kampuchea
- Olympic Stadium (Phnom Penh)
- American Assistance for Cambodia

The rearranged list by ChatGPT is:

- Khmer language
- Politics of Cambodia
- History of Cambodia
- Economy of Cambodia
- Siem Reap
- Cambodia National and Provincial Resources Data Bank
- Norodom Buppha Devi
- Party of Democratic Kampuchea
- Olympic Stadium (Phnom Penh)
- American Assistance for Cambodia

For statement “It's the end of an empire! This empire, in fact! After 600 years, it's goodbye, this, hello, Turkish Republic!”, which has the answer “Ottoman Empire”, the initial list is:

- 1920s
- Enosis
- 600
- Van, Turkey
- Kilkis (regional unit)
- Ottoman Empire
- İsmet İnönü
- Aftermath of World War I
- 340
- Simeon of Moscow

The ChatGPT rearranged list is:

- Ottoman Empire
- 1920s
- Enosis
- 600
- Van, Turkey
- Kilkis (regional unit)
- İsmet İnönü
- Aftermath of World War I

- 340
- Simeon of Moscow

For question “Matthias Church or Matyas Templom where Franz Joseph was crowned in 1867”, with answer “Budapest”, the initial list is:

- Matthias Corvinus
- Votive Church, Vienna
- Huedin
- Church of St. Martin de Porres (Poughkeepsie, New York)
- Franz Xaver Witt
- 1867
- Basilica of Our Lady of Perpetual Help, Brooklyn
- Budapest
- Mayerling
- Holíč

The rearranged ChatGPT list is:

- Budapest
- 1867
- Matthias Corvinus
- Votive Church, Vienna
- Huedin
- Church of St. Martin de Porres (Poughkeepsie, New York)
- Franz Xaver Witt
- Basilica of Our Lady of Perpetual Help, Brooklyn
- Mayerling
- Holíč

As can be seen from the 4 examples above, ChatGPT reordering of ranking would improve the results and statistics by a large margin, as it can select the correct answer and put it at the first position each time. But, given the behavior of ChatGPT, the results of the prompt are nondeterministic, so if ran multiple times, the resulting reordering can be wrong as well. When we calculated the new score manually, we obtained multiple MRR scores because of this behavior. The score improved by ~0.10 (average of the 5 trials that we did).