Data mining - Building a part of Watson

Munteanu Bianca Ștefania
Munteanu Tudor Constantin
Negrea Gabriela Veronica
Nichitean Mărioara
Popa Monica

Github repository link

- `https://github.com/BiancaM30/Data_Mining.git`

Compiling and Running - Instructions

- **Setting Up the Environment:**
  - **Install PyCharm:** If you haven't already, download and install PyCharm from the JetBrains website.
  - **Clone the Project:** Open PyCharm, go to VCS - Checkout from Version Control - Git. Enter the specified Git link to clone the project. This provides a dedicated space for your code and associated files.
- **Install Required Libraries:**
  - **Open the Terminal in PyCharm:** You can find it at the bottom of the PyCharm window.
  - **Install Libraries:** The code uses several libraries (nltk, whoosh, etc.).
    To install them, use the pip command in the terminal:

```
pip install nltk whoosh openapi
```

- **Download or build the index**
  - You can obtain the index by downloading it from here: `https://drive.google.com/drive/folders/1dapWGLerbHQdf1fvQVcSS_OVfJjqsYW8`. Once downloaded, simply place it within your project directory.

- If the index is not already present within your project, it will be generated during the first run of the program. However, please note that this indexing process may take several hours to complete. Therefore, we highly recommend opting for the first option.

- **Paths verification** Ensure that the paths to the dataset directory, index directory and questions file are correctly set in the main file.

- **Choose an Option:**

  - The program will prompt: *Choose an option:*
  - Enter **1**, **2**, **3** or **4** corresponding to your choice and press Enter.

  Option 1 - Run Tests from Questions File:

  - Selecting **1** will automatically process a set of questions from "questions.txt" file.
  - The script will display each query and the search results.
  - After processing all questions, it will display the number of correct answers, performance metrics, list of questions with correct answers and the list with all anwers.

  Option 2 - Enter a Custom Query:

  - If you choose **2**, you'll be prompted to enter a category and a clue for your query.
  - After entering each, the script constructs and runs your query against the index.
  - Results, if any, will be displayed in the console.

  Option 3: Rerank titles from questions file with ChatGPT

  - Selecting **3** will automatically process a set of questions and answers from "chatGpt_input.txt" file.
  - The system make calls to ChatGPT 3.5 to either perform reranking on titles returned by the Standard IR system, or generate new results.
  - Performance metrics will be displayed in the console and the new answers will be stored in "chatGpt_output.txt"

  Exiting the Program:

  - To exit, enter **4**. The script will display *"Exiting program."* and terminate.

  Invalid Choice:

  - If you enter a choice other than **1**, **2**, or **3**, the script will display *"Invalid choice, please try again."* You will then be prompted to make another choice.

Provide a description of the code

The code is grouped into 7 files and below we will describe the main functionalities:

- **main.py:** In this file, the environment is set up by importing necessary modules, establishing custom text analyzers, and defining a schema for the search index. The script handles index management by either creating or opening a Whoosh search index and indexes Wikipedia pages as required. For query processing, it incorporates functions that transform questions into search queries and conducts searches on the index to retrieve relevant document titles. The system's performance is evaluated by running tests on a set of questions, assessing precision and mean reciprocal rank. User interaction is facilitated, allowing inputs of custom queries or reranking questions using ChatGPT. Lastly, the main loop of the script provides an interactive menu, offering options for testing, querying, reranking with ChatGPT, or exiting the program.

- **Lemmatizer.py:** The Lemmatizer class is a custom text processing filter for Whoosh's search indexing system. It uses NLTK's WordNetLemmatizer to reduce words in the text to their base or dictionary forms.

- **SynonymFilter.py:** The SynonymFilter class is a custom filter designed for enhancing text processing in Whoosh's search engine framework by incorporating synonyms into the analysis pipeline. It utilizes the wordnet corpus from NLTK to find synonyms for each token in the text, thus expanding the scope of search and indexing to include various forms and expressions of a word.

- **data_parser.py:** Includes functions for parsing and extracting data from Wikipedia pages and question files.

- **performance.py:** Contains two key functions for evaluating the question-answering system. precision_at_1, calculates the Precision at 1 (P@1) score, assessing if the top result in search results is the correct answer. mean_reciprocal_rank computes the Mean Reciprocal Rank (MRR), a statistic that averages the reciprocal ranks of results for a set of queries.

- **utilities.py:** It includes functions for synonym generation, category retrieval, document information extraction, and checking category existence in a search index.

- **chatGpt_integration.py:** Contain functions which integrate OpenAI's GPT-3.5-turbo model for reranking or generating Wikipedia page titles, and it also evaluates the performance of these reranked results.

Important text files in the project:

- **questions.txt**: Used for testing the system; contains the given list of questions, with corresponding categories and correct answers.

- **Top 10 responses from the system.txt**: Displays the top 10 answers provided by our Standard IR system, offering insights into initial results.
- **chatGpt_input.txt**: Contains questions with a list of titles from the IR system. If the correct title is within the top 10, it is included; otherwise, the list remains empty.
- **chatGpt_output.txt**: Contains questions and their top 10 answers after reranking by ChatGPT, along with new titles generated by ChatGPT, if needed, for questions without conclusive initial titles.

Answers for - Indexing and Retrieval -

- **Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.)**

  In preparing terms for indexing in our project, we employed a multi-step approach using a custom analyzer, which can be seen in Figure 1

```
custom_analyzer = RegexTokenizer() | LowercaseFilter() | SynonymFilter() | Lemmatizer() | StopFilter(
    stoplist=STOP_WORDS)
```

Figure 1: Custom analyzer code snippet

Here's a conceptual explanation of each step and our argument for choosing them:

1. RegexTokenizer: This breaks the text into individual terms or tokens based on regular expressions. We chose this for its precision and flexibility in defining how tokens are identified from the text.

2. LowercaseFilter: It converts all tokens to lowercase. This standardization is important because it ensures that variations in capitalization don't affect the indexing process, treating words like "Newspaper" and "newspaper" as identical.

3. SynonymFilter: This adds synonyms to the token stream. We incorporated this because we observed that in many cases, synonyms of words in queries were more likely to appear in the Wikipedia pages containing the relevant answers. This step aims to broaden the search capability by recognizing semantically similar terms.

4. Lemmatizer: Instead of using a stemmer, we opted for a lemmatizer. While stemmers truncate words to their roots, often leading to a loss of meaning and an increase in false positives, lemmatizers reduce words to their base or dictionary form in a context-sensitive manner. This approach provides more accurate and meaningful indexing.

5. StopFilter with a Custom Stop Words List: This step removes common 'stop words' (like 'the', 'is', 'in') from the token stream. We customized the stop words list (STOP_WORDS) to suit our dataset, ensuring irrelevant words don't clutter the index and affect search relevance.

Our indexing process, while comprehensive, has a drawback. The inclusion of the SynonymFilter significantly increased the size of our index. This expansion, while beneficial in enhancing the scope of our search capabilities, also presents challenges in terms of storage and search efficiency. Larger indices can require more storage space and potentially slow down search queries. We're considering this trade-off carefully as we continue to refine our approach.

In our project, we utilized a specific schema for indexing our dataset, as seen in 2. This schema includes:

1. **title_original**: Stored as TEXT and kept unaltered. We chose to store the original title because it's essential to return the title exactly as it is in the dataset, without any modifications from the analyzer. This helps in accurately presenting search results.

2. **title**: Also stored as TEXT, but this field is processed through our custom analyzer. This processed title aids in enhancing the searchability of the content, taking advantage of the various filters and tokenization we've implemented.

3. **content**: This is the main body of the text, indexed using the same custom analyzer. It's not stored to conserve space.

4. **category**: Like the title, this field is indexed using our custom analyzer and is stored. Storing the category information is important for facilitating efficient retrieval based on categorical searches, enhancing the user's ability to find relevant information quickly.

```
schema = Schema(
    title_original=TEXT(stored=True),
    title=TEXT(analyzer=custom_analyzer, stored=True),
    content=TEXT(analyzer=custom_analyzer),
    category=TEXT(analyzer=custom_analyzer, stored=True)
)
```

Figure 2: Indexing schema code snippet

- **What issues specific to Wikipedia content did you discover, and how did you address them?**

  In our project, we've designed our parsing approach around identifying titles within double brackets [[title]] and categories indicated by the CATEGORIES keyword in the Wikipedia dataset. We considered the content as the part of the page found before another title occurs. Despite this, we've encountered some challenges. Particularly, a notable number of titles don't have associated categories, which can be observed in Figure 3. We discovered this issue when we displayed titles along with their indexed categories, after the indexation process was done.

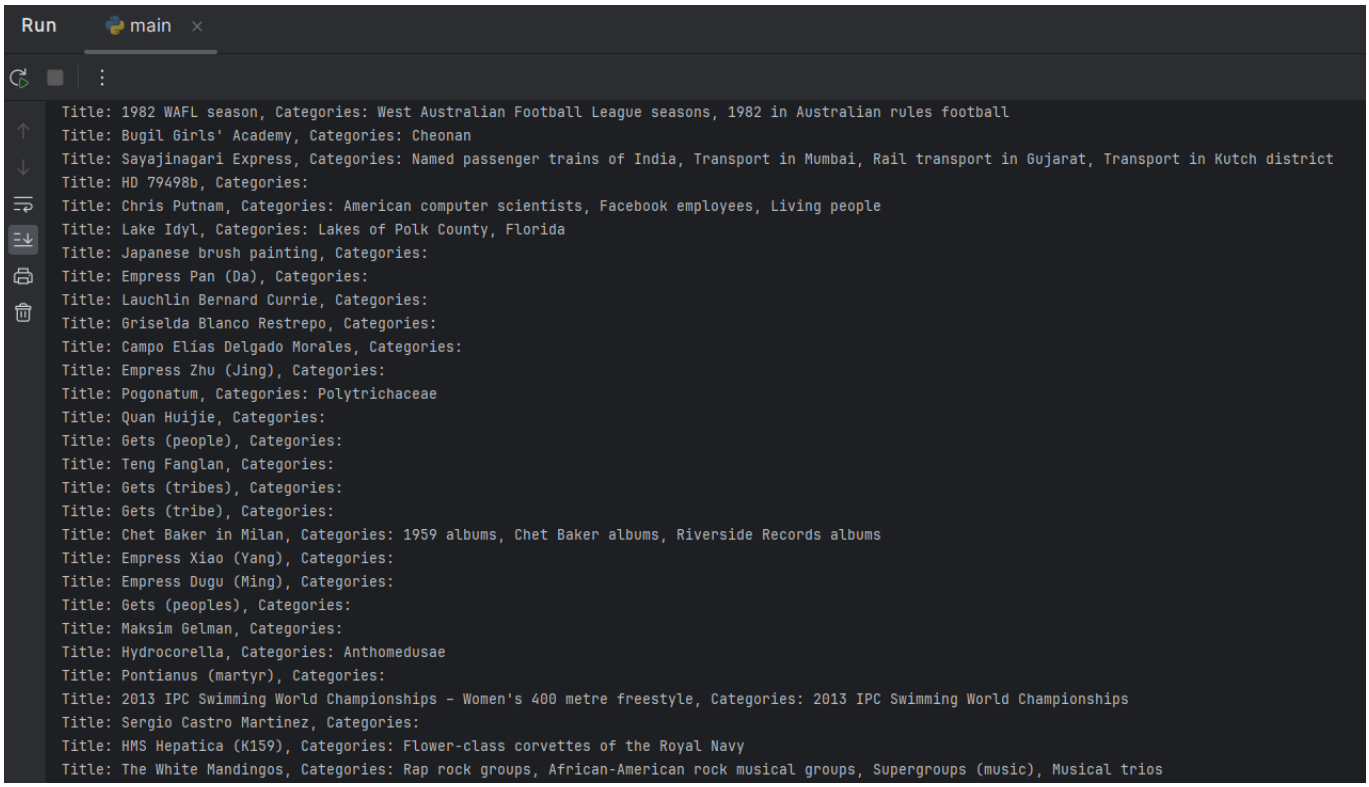

Figure 3: Wikipedia issue: Missing categories for titles

Upon further investigation, we found that this issue is often due to redirect pages, as illustrated in Figure 4. These pages typically don't contain category information, posing a significant challenge to our categorization process. We also noticed that the content of these pages includes numerous links, adding complexity to our parsing task.

[[Cryptography/Hashfunction]]

#REDIRECT Hash function

[[Cryptography/Key]]

#REDIRECT Key (cryptography)

Figure 4: Wikipedia issue: Redirect problem

As a team, we are planning to address these issues in future iterations of our project. We aim to refine our parsing logic to handle redirects effectively and to develop a method to

infer categories for titles without explicit category data. These improvements are expected to enhance the accuracy and completeness of our dataset, making it a more reliable source for information retrieval and analysis.

- **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?**

  For the clues in our project, we don't use all the words. We select a subset after preprocessing with a custom analyzer, which includes tokenization, converting to lowercase, and lemmatization, while removing stop words. We have developed an algorithm to fetch the most relevant synonyms from WordNet to expand our search terms, using only the first two synonyms to avoid overwhelming the query.

  The category of the question is indeed utilized. It's processed first to set the context for the search and is included in the final query. This method ensures that our search is focused and relevant to the category in question while being comprehensive enough to account.

  Let's take the first example from our test file, for a better understanding of the next 2 steps.

  We have the query: **The dominant paper in our nation's capital, it's among the top 10 U.S. papers in circulation**

  And the category: **NEWSPAPERS**

  In phase one, after preprocessing the clue with a custom analyzer, we select keywords and their synonyms, organizing them in a boolean structure with OR operators to capture various lexical forms. We then connect these groups with AND operators to construct a complex query that requires all conditions to be met, as seen in Figure 5

```
(newspaper OR newsprint OR newspaper publisher) AND (dominant OR prevailing OR prevalent)
AND (paper OR report OR theme)
AND (our)
AND (nation OR commonwealth OR body politic)
AND (capital OR uppercase OR great)
AND (among)
AND (top OR clear OR peak)
AND (10 OR ten OR decade)
AND (u.s)
AND (paper OR report OR theme)
AND (circulation)
```

Figure 5: Query - phase 1

Phase two involves transforming the structured query to target specific fields in our dataset—namely the 'category,' 'title,' and 'content' fields. We extracted the category and put it in the first set of parentheses, and after that, we searched in the title and content after the rest of the tokens among with their synonyms, as seen in Figure 6.

```
((category:newspaper) AND (content:nation OR title:nation OR content:commonwealth OR title:commonwealth
OR content:body politic OR title:body politic)

AND (content:dominant OR title:dominant OR content:prevailing OR title:prevailing OR content:prevalent OR
title:prevalent)

AND (content:capital OR title:capital OR content:uppercase OR title:uppercase OR content:great OR title:great)

AND (content:paper OR title:paper OR content:report OR title:report OR content:theme OR title:theme)

AND (content:paper OR title:paper OR content:report OR title:report OR content:theme OR title:theme)

AND (content:top OR title:top OR content:clear OR title:clear OR content:peak OR title:peak)

AND (content:10 OR title:10 OR content:ten OR title:ten OR content:decade OR title:decade)

AND (content:circulation OR title:circulation)

AND (content:among OR title:among)

AND (content:our OR title:our)

AND (content:u.s OR title:u.s))
```

Figure 6: Query - phase 2

Our search strategy includes a heuristic that dynamically modifies queries to enhance retrieval effectiveness. After constructing the initial query with categories and clues processed through our custom analyzer, we iteratively adjust this query by selectively removing the least significant parts—those enclosed in parentheses with the fewest characters. This method systematically broadens the search scope, increasing the chances of matching relevant documents.

Answers for - Measuring performance -

- Measure the performance of your Jeopardy system, using at least one of the metrics discussed in class - justify your choice.

  Using metrics to evaluate our Jeopardy system is critical for several reasons. Metrics provide a quantitative measure of how well the system performs, so we can compare our system's performance against other similar systems or future versions of our own system. Also, they can reveal specific areas where our system excels or needs improvement, guiding future development efforts.

  In evaluating our question-answering system, we have selected *Precision at 1 (P@1)* and *Mean Reciprocal Rank (MRR)* as our metrics, as they are particularly well-suited for assessing the performance of such a system. P@1 is a critical metric in this context because, in Jeopardy, the correctness of the first response is very important. However, relying solely on P@1 is not sufficient without also considering MRR. P@1 exclusively evaluates whether the first answer given is correct. While this is important for a format like Jeopardy, P@1 does not provide any insight into the overall ranking quality of the system. On the other hand, MRR provides a broader view of the system's performance. It measures how well the system ranks the correct answer within its list of responses, offering insight into the system's overall retrieval and ranking capabilities. While the first response's accuracy is crucial, understanding how the system prioritizes correct answers is also valuable, especially in cases where multiple plausible answers might be generated. Together, P@1 and MRR offer a comprehensive evaluation of our system's ability to not only deliver accurate first responses but also to rank all potential answers effectively.

  Precision at 1 (P@1) is a metric that measures whether the top answer provided by your system is correct. It's a binary metric (either 0 or 1) for a single query. A high P@1 score means your system is highly effective at providing the correct answer on the first try. Traditional precision assesses the overall quality of the set of retrieved items, rather than focusing solely on the top result. P@1 only evaluates the top result, while traditional precision evaluates the entire set of retrieved results.

$$P@1 = \frac{\text{Number of correct first responses}}{\text{Total number of queries}}$$

  Mean Reciprocal Rank (MRR) is useful in evaluating how well the system ranks correct answers, even if the correct answer is not the first one.

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\text{rank}_i}$$

In this formula, Q represents the total number of queries, and rank_i is the position of the first correct answer in the list of responses for the i-th query. Essentially, for each query, the reciprocal of the rank of the first correct answer is calculated, and then these values are averaged over all queries.

Evaluation results Out of the total queries posed to the system, it provided the correct answer in **19** instances. The system correctly answered the following questions, as numbered in the dataset [2, 7, 13, 18, 38, 40, 41, 44, 46, 48, 56, 62, 65, 71, 75, 80, 86, 97, 99]. The system achieved a **P@1** score of **0.13**. This metric indicates that in 11% of the cases, the system's first response was the correct answer. The **MRR** achieved was **0.146**.

Breakdown of MRR Results:

- Rank 1: 13 correct answers
- Rank 2: 1 correct answers
- Rank 3: 1 correct answers
- Rank 4: 1 correct answer
- Rank 5: 1 correct answer
- Rank 6: 2 correct answers
- Ranks 7, 8, 9, 10: No correct answers

Answers for - Error analysis -

- **How many questions were answered correctly/incorrectly?**
  - **Correct Answers**: The system answered 19 questions correctly.
  - **Incorrect Answers**: The system answered 81 questions incorrectly.
- **Why do you think the correct questions can be answered by such a simple system? What problems do you observe for the questions answered incorrectly? Aim to group the errors into a few classes and discuss them.**

  The correct answers being identified by a relatively simple system indicates that it is effective in handling queries with clear, direct associations with their answers. This success can be attributed to:

  - **Direct Matching**: The system likely excelled in scenarios where the keywords of the category, question or the added synonyms directly matched the title or content of a Wikipedia page. This indicates a strong correlation between the more straightforward query and indexed content. For instance, specific names (like ”Daniel Hertzberg”), or unique events (like winning a Pulitzer Prize) may easily be associated with the titles of their pages. This direct matching bypasses the need for complex contextual understanding or semantic processing.
  - **Distinctiveness of Keywords**: The correct answers were often associated with queries containing distinctive keywords that uniquely identified the subject. Such keywords are less ambiguous and have a high likelihood of being directly represented in the relevant Wikipedia page. For example, ”In 2009: Joker on film”, and its category ”GOLDEN GLOBE WINNERS” have only a few specific words (after removing the stop words), making it easier to search and point to ”Heath Ledger”.

  In summary, the correct answers were largely due to the system’s effectiveness in handling queries with direct, unambiguous keywords and distinctive words. Thus, the system may have limitations in processing more complex or nuanced queries.

  So, the incorrect answers can be grouped to identify patterns in the system’s shortcomings. Here are some observed error categories:

  - **Broad or Ambiguous Queries/Clues**: For example, categories like ”POTPOURRI,” ”OLD YEAR’S RESOLUTIONS,” and ”STATE OF THE ART MUSEUM (Alex: We’ll give you the museum. You give us the state.)” are quite broad or ambiguous, possibly leading to a wide range of possible answers that the system failed to narrow down correctly. Also, such queries might lead the system to retrieve Wikipedia pages that are only loosely related to the query, as the system might struggle to discern the most relevant page among many potential matches.

- **Specificity and Uniqueness**: As this was previously described as an advantage, it also can be regarded as a drawback for queries that lack these properties. For example, the questions "Song that says, "you make me smile with my heart; your looks are laughable, unphotographable" along with its clue "BROADWAY LYRICS" is challenging since direct matching cannot be performed against the correct page "My Funny Valentine" because it contains no lyrics.
- **Inadequate Contextual Understanding**: The system might struggle with synonyms or semantic variations, failing to recognize when different wording or expressions refer to the same concept or page. To not overload the system, as a compromise, it uses only the first two synonyms for each word, which may not be enough to cover all the nuances of the queries, resulting in missing out on the correct answers.

All in all, to improve the system, it would be beneficial to enhance its ability to understand query context (e.g. cultural references), refine the indexing process to capture a wider range of relevant content, and improve its handling of semantic variations. Also, the incorporation of specialized knowledge bases (NLP techniques) could enhance the accuracy and versatility.

Answers for - Improving retrieval -

- Improve the above standard IR system using natural language processing and/or machine learning. What is the performance of your system after this improvement?

  To refine the results generated by the previously described IR system, we integrated OpenAI's ChatGPT to rerank the obtained Wikipedia page titles.

  Implementation

  Input Processing: The system begins by reading a structured input file "chatGpt_input.txt" containing questions alongside their respective top 10 Wikipedia page titles. These titles are either provided by the existing IR system or are left empty for fresh generation. In cases where this list is absent or left blank, it implies that the initial IR system did not yield a conclusive set of titles containing the correct answer. Under such circumstances, the system is programmed to leverage ChatGPT's capabilities to independently generate a new list of the top 10 most relevant Wikipedia page titles for each question scratch.

  ChatGPT Integration:

  1. **API Setup**: The system uses OpenAI's GPT-3.5-turbo model, accessed via an API key. This setup ensures a connection to a powerful language model capable of understanding and processing complex queries.
  2. **Prompt Construction:** Each interaction with ChatGPT begins with an initial system message that sets the context. This message instructs ChatGPT to act as a "helpful assistant," which frames its subsequent responses. The actual query or question is then added to the messages. This step is crucial as it provides the specific context for ChatGPT's response.

     If existing Wikipedia titles are provided, they are appended to the messages individually. This signals to ChatGPT that these titles are related to the question and should be considered in its response. If no titles are provided, a message is added to prompt ChatGPT to generate relevant Wikipedia titles based on the question.

```
if wiki_titles:
    messages.append({"role": "user", "content": "Wikipedia Titles:"})
    for title in wiki_titles:
        messages.append({"role": "user", "content": title})
    messages.append({"role": "user", "content": "Rerank these titles in order of relevance to the question."})
else:
    messages.append({"role": "user", "content": "What are the top 10 Wikipedia page titles that are relevant to this question?"})
```

Figure 7: ChatGPT prompt

  3. **Response Processing**: ChatGPT's output is parsed to extract reranked or newly generated Wikipedia titles. This is achieved by splitting the response into lines and then extracting the titles from these lines.

4. **Request Handling**: The request to the OpenAI API is enclosed within a try-except block inside a for loop. This setup is crucial for handling any errors that might occur during the API request, such as network issues or server errors. If an error is caught, indicating a failure in the API request, the function prints an error message along with the attempt number. It then waits for a specified delay (5 seconds) before retrying the request. The retries parameter (defaulting to 3) determines how many times the function will attempt to resend the request after an initial failure. We implemented this because we have encountered issues when we tried to send 100 requests in a short period of time, to test the performance on the given 100 questions.

Output and Metrics:

The reranked results are saved to an output file 'chatGpt_output.txt', maintaining a format similar to the input file for consistency. Performance metrics such as Precision at 1 and Mean Reciprocal Rank (MRR) are calculated to evaluate the system's effectiveness.

1. **Top 10 Answers Coverage:** The number of correct answers found within the top 10 results has risen to 45. This indicates a significant improvement in the IR system's ability to identify relevant pages, demonstrating ChatGPT's effectiveness in refining search results.
2. **Precision at 1 (P@1):** P@1 has increased to 0.19. This metric measures the proportion of queries for which the first result returned is the correct one.
3. **Mean Reciprocal Rank (MRR):** The MRR has improved to approximately 0.267, so the correct answers appear higher in the list of search results. The partial rankings also increased to (19, 8, 5, 1, 3, 3, 1, 2, 1, 2).

Comparative Analysis

Comparing these metrics to the pre-ChatGPT implementation stats, we observe:

1. **Increase in Top 10 Coverage:** The number of correct answers in the top 10 has more than doubled, indicating a significant improvement in the overall relevance of the search results.
2. **Higher Precision at 1:** The increase in P@1 signifies that users are more likely to find the correct answer without looking beyond the first result.
3. **Improved MRR:** The rise in MRR reflects that not only are more correct answers appearing in the top 10, but they are also ranked closer to the top on average.

Challenges and Limitations

- **Title Mismatch**: Some of ChatGPT's responses included elaborated titles that, while contextually accurate, did not precisely match the Wikipedia page titles. For example,

it returned the title ”James Dean - A Hollywood actor known for his roles in rebellious characters” instead of simply ”James Dean”. This resulted in these titles not being recognized as correct in the performance evaluation.

– **Response Format Issue**: In a few instances, ChatGPT’s output format led to the inclusion of an introductory sentence at the top of the results list, pushing down the actual titles. For example, we have observed the phrase ”Here are the top 10 Wikipedia page titles that are relevant to the question you asked:” in 4 out of 100 responses. This formatting anomaly affected the accuracy of the first result.