Jeopardy System Report

Students: Iulia Păpureanu, Alexandru-Cristian Mitrofan

Contents

1. Indexing and Retrieval

For implementing the solution, we used Python and the Whoosh library.

1.1 Custom Analyzer

For the indexing part, we used a custom analyzer that incorporates both stemming and stop words removal. The stemming process reduces words to their root form, used for matching different variations of the same term. Simultaneously, stop words, common language terms that contribute little to information retrieval, were excluded to enhance the relevance of indexed terms.

```
stop_words = set(stopwords.words("english"))

# Create a custom analyzer with stemming and stop words removal
custom_analyzer = StemmingAnalyzer() | StopFilter(stoplist=stop_words)

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

1.2 Index Creation

The Wikipedia pages are distributed across 80 files. Each file contains several thousand pages, with titles enclosed in double square brackets (e.g., [[BBC]]). The following code demonstrates the creation of the Whoosh index, using the custom schema we mentioned previously:

First, we created the index:

```
# Create an index in the 'index' directory with the defined schema
index = create_in("index", schema)
```

Then, we extracted the pages into documents, as title – content pairs:

```
def extract_documents(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        content = file.read()
        # Split content into documents based on the delimiter "[["
        documents = content.split("[[")
        for document in documents:
            # Skip empty strings
            if not document.strip():
                continue
            # Try to split each document into title and text using "]]"
            parts = document.split("]]", 1)
            # Check if there are enough parts
            if len(parts) == 2:
                title, text = parts
                writer.add_document(title=title.strip(), content=text.strip())
```

1.3 Query Construction

The retrieval component aims to match Jeopardy clues with the most relevant Wikipedia pages. A key consideration is how to build the query from the given clue. In our approach, we use a QueryParser to parse the clue and create a query for the search. We used all the words in the clue, not a subset and we didn't use the category of the question.

```
def retrieve_wikipedia_page(question, searcher):
    query_parser = QueryParser("content", searcher.schema)
    query = query_parser.parse(question)
    results = searcher.search(query, limit=1)
    if len(results) > 0:
        return results[0]["title"]
    else:
        return "No Wikipedia page found"
```

2. Measuring Performance

We chose P@1 as our performance metric because it fits with our task of identifying the correct Wikipedia page title for Jeopardy-style questions. With a focus on precision at the top result, P@1 reflects our system's ability to provide the correct answer. This choice simplifies interpretation and directly addresses the requirement of delivering a single, accurate result in the top position.

For checking the answers, we parse all the questions and check if the retrieved result is correct:

```
def check_answers(file_path, searcher):
    correct_answers = 0
    total_questions = 0
    with open(file_path, "r") as file:
        lines = file.readlines()
        i = 0
        while i < len(lines):
            category = lines[i].strip()
            clue = lines[i + 1].strip()
            correct_answer = lines[i + 2].strip()
            retrieved_answer = retrieve_wikipedia_page(clue, searcher)
            print("RESULT:")
            print(retrieved_answer)
            print(correct_answer)
            if (retrieved_answer.lower() in correct_answer.lower()) or (
                    correct_answer.lower() in retrieved_answer.lower()):
                correct_answers += 1
            # Move to the next set of category, clue, and correct answer
            i += 4  # Skip the blank line
            total_questions += 1
    return correct_answers, total_questions
```

The performance of our system:

```
Number of correctly answered questions: 21
Total nr of questions: 100
Precision at 1:  0.21
```

3. Error Analysis

The system manages to answer questions related to well-known entities and specific categories. It identifies key terms, contributing to accurate retrievals.

Main problems:

- Insufficient Context Handling: There are challenges with questions requiring specific interpretation or contextual understanding due to the system's reliance on individual terms.
- Ambiguity in Clues: Ambiguous clues with multiple interpretations present challenges, leading to occasional incorrect retrievals.

Examples of issues met in results:

1. Clue: The dominant paper in our nation's capital, it's among the top 10 U.S. papers in circulation

   Retrieved Answer: Confederate States of America

   Issue: The system misinterpreted the clue, failing to recognize it as a reference to a newspaper rather than a political entity.

2. Clue: Song that says, "you make me smile with my heart; your looks are laughable, unphotographable"

   Retrieved Answer: No Wikipedia page found

   Issue: The system struggled with the cultural reference and lyric-based clue, unable to link it to the correct title "My Funny Valentine."

3. Clue: 1988: "Father Figure"

   Retrieved Answer: Ruhollah Khomeini

   Issue: The system failed to grasp the pop culture reference, providing an unrelated answer instead of the correct one, "George Michael."

4. Clue: In the 400s B.C. this Chinese philosopher went into exile for 12 years

   Retrieved Answer: Roman Empire

   Issue: The system misinterpreted the historical clue, associating it with a different context (Roman Empire) instead of identifying the philosopher (Confucius).

4.Improving Retrieval

We tried to use ChatGPT to rerank the top 10 pages produced by the retrieval system. We tried to connect to the OpenAI API, but we had issues with exceeding the quota for our plan.

In the end, we manually selected the first 10 pages provided by our system and asked ChatGPT to pick the best one for the provided clue: