Building (a part of) Watson

Radu Andrei – Ioan (SDI - 254)

1. Code

To implement this project, I decided to use Java with Lucene. The program consists of 6 classes which I will explain below:

- **TextProcessingOption**: an enum that contains different text processing options. It is used to specify what kind of processing to be applied to the text found in the documents. It has the following options:
    - **NONE**: the text is not modified.
    - **STOP_WORDS**: stop words are removed from the text.
    - **STEMMING**: stemming is applied to the text.
    - **LEMMATIZATION**: lemmatization is applied to the text.
    - **STOP_WORDS_STEMMING**: stop words are removed from the text and stemming is applied.
    - **STOP_WORDS_LEMMATIZATION**: stop words are removed from the text and lemmatization is applied.
- **Utils**: a class where the paths to the data and folders are saved. Also it contains implementations for the functions that perform the text processing.

- **WikipediaIndexer**: a class that builds the index for the Wikipedia pages. It parses every file and splits it into Wikipedia pages so each page will appear as a separate document in the index. The splitting is done based on these rules:

    - The current line starts with “[[“ and end with “]]” -> a new Wikipedia page is starting and between the square brackets is its title.

    - The current line starts with “CATEGORIES: “ -> the line contains the categories.

    - The current line starts with “=” and ends with "=” -> the line contains a header.

    - Otherwise -> the line contains the text of the Wikipedia page.

    After splitting a Wikipedia page into title, categories, and content (contains the title, categories and all the information of the Wikipedia page), a text processing function is applied to the content, based on the specified option from the TextProcessingOption. In the end, the page is added to the index using Lucene.

- **JeopardyQueryResult**: a class that holds the information about a document from the index and its score after executing a query.

- **JeopardyQuery**: a class where the Jeopardy questions are parsed, and a query is created for each clue. After searching using Lucene, the first result is checked with the correct answer to calculate the P@1 metric and display the information in the console. Also, here is done the reranking of the first 2 results with bert.

- **Main**: this is the class from where the program starts. It is used to create the indexes if they are not present. Also, it runs a query on the specified index for every Jeopardy question and measures the performance of the program.

2. Indexing and retrieval

2.1. Indexing

To build the index I used the provided text files. Each file contains multiple Wikipedia pages. As required, I indexed each Wikipedia page as a separate document in the index. To do this, I parsed the files and extracted each Wikipedia page following these rules:

- The current line starts with "[[" and end with "]]": a new Wikipedia page is starting and between the square brackets is its title.

- The current line starts with "CATEGORIES: ": the line contains the categories.

- The current line starts with "=" and ends with "=": the line contains a header.

- Otherwise: the line contains the text of the Wikipedia page.

For each Wikipedia page, I decided to index a document containing the following data: title, categories, and content. The content consists of title, categories, and the Wikipedia page text.

Before adding the document to the index, I applied text processing on the content. I ended up having 5 different indexes: no text processing, stop-words removed, text stemming, stop-words removed and text stemming, text lemmatization. I did not create an index with stop-words removed and text lemmatization because the lemmatization took a lot of time. To apply this text processing techniques, I tokenized the text using Stanford CoreNLP and did the following:

- Stop-words removal: removed the words that were included in the EnglishAnalyzer stop words set, found in Lucene.
- Text stemming: used PorterStemmer found in Lucene.

- Text lemmatization: used Stanford CoreNLP.

After that, the document was added to the index and continued to process a new Wikipedia page.

2.2. Retrieval

I used the Jeopardy questions provided, which consists of category, clue, and answer. To create the query, I decided to use both the category and the clue and apply some processing. As mentioned earlier, there are 5 indexes, and each has a different content processing type, so for the query I applied this rule: Apply the same processing option on the query as the one applied when building the index. As an example, for the index where I removed the stop-words, I also removed the stop-words from the query; for the index where I applied stemming, I also applied stemming on the query, and so on. To do this, I used the same techniques presented earlier.

To search in the index, I used the IndexSearcher provided by Lucene, where I could set what similarity strategy to be used. I tested different similarity strategies like: TF-IDF, Boolean, BM25, Jelinek-Mercer and Dirichlet. The search returns a list of top documents, where the first document is the first answer.

2.3. Answers

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

The documents were organized well. It was easy to distinguish when a Wikipedia page was starting because of the title that was between square brackets. There were a lot of headers that are the same for multiple Wikipedia pages, like: “See also”, “References”, “Further reading”, “External links”, etc. I tried to remove from the content of the page these headers, but the performance of the system did not change.

Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset?

As described earlier, the query consists of the category and the clue. The type of text processing used for the index is applied also on the query.

Are you using the category of the question?

Yes, I am using the category of question. When I create the index, I add a field for “categories” and one for “content”. The content also contains the categories. At first I tried to search separately, searching with the category in the “categories” field and with the clue in the “content” field, but I got bad results. Than I decided to create the query concatenating the category and the clue, and search with the query only in the “content” field, and I got better results.

3. Measuring performance

For the performance measuring I used Precision at 1(P@1). This metric measures the accuracy of the top-ranked result. It is relevant because Jeopardy is a question-answering scenario where only the first answer matters and it will reflect the precision of the model. It is calculated with the following formula:

P@1 = Number of correct top-ranked answers / Total number of predictions.

Below is a table where I tested different similarity functions on all the indexes in order to find which has the best performance.

Index type \ Similarity function	TF-IDF	Boolean	BM25	Jelinek-Mercer	Dirichlet
None	0	0.8	0.2	0.23	0.29
Remove stop-words	0	0.12	0.2	0.22	0.31
Stemming	0.01	0.13	0.23	0.2	0.29
Remove stop-words & stemming	0	0.13	0.23	0.22	0.31
Lemmatization	0	0.09	0.21	0.21	0.3

The results show that my best systems are using Dirichlet Similarity and are removing-stop words from the index and the query. I will choose the one where I only remove the stop-words without applying stemming since the results are the same.

4. Error analysis

I chose to perform error analysis on the system that uses Dirichlet similarity and the index where the stop-words were removed. I decided to group the questions by category and count the number of correct and incorrect answers for each category.

Category	**Correct Answers**	**Incorrect Answers**	**Accuracy**
GREEK FOOD & DRINK	3	0	100.0%
HE PLAYED A GUY NAMED JACK RYAN IN...	3	0	100.0%
RANKS & TITLES	2	0	100.0%

UCLA CELEBRITY ALUMNI	4	1	80.0%
THE RESIDENTS	2	1	66.7%
SERVICE ORGANIZATIONS	2	1	66.7%
"TIN" MEN	2	2	50.0%
CEMETERIES	1	1	50.0%
AFRICAN-AMERICAN WOMEN	2	2	50.0%
NEWSPAPERS	2	3	40.0%
GOLDEN GLOBE WINNERS	2	3	40.0%
NOTES FROM THE CAMPAIGN TRAIL	1	2	33.4%
CAPITAL CITY CHURCHES	1	2	33.4%
POETS & POETRY	1	2	33.4%
HISTORICAL HODGEPODGE	1	3	25.0%
AFRICAN CITIES	1	3	25.0%
1920s NEWS FLASH!	1	3	25.0%
BROADWAY LYRICS	0	1	0.0%
THAT 20-AUGHTS SHOW	0	1	0.0%
CONSERVATION	0	3	0.0%
HISTORICAL QUOTES	0	3	0.0%
'80s NO.1 HITMAKERS	0	5	0.0%
COMPLETE DOM-INATION	0	3	0.0%
THE QUOTABLE KEATS	0	2	0.0%
NAME THE PARENT COMPANY	0	5	0.0%
POTPOURRI	0	3	0.0%
OLD YEAR'S RESOLUTIONS	0	4	0.0%
CAMBODIAN HISTORY & CULTURE	0	3	0.0%
I'M BURNIN' FOR YOU	0	2	0.0%
STATE OF THE ART MUSEUM	0	5	0.0%

There are 31 correct answers and 69 incorrect. The mean accuracy for all the categories is 30.6%. There are only 3 categories where all the answers were correct. On the other hand, there are a lot of categories that have 0 correct answers, or the precision is below 50%.

I think that such a simple system can answer questions correctly because of the use of common keywords or phrases in both the Wikipedia pages and the questions. Also, having a consistent structure and a low ambiguity can help identify the correct answer.

For example, the questions for the category “GREEK FOOD & DRINKS”, have distinctive and specific keywords related to Greek food, like: “skewer”, “grilled”, “sheep’s milk”. Those keywords can be indicators for the system to make associations with specific Greek dishes. Also, the questions provide descriptive clues that can point to the correct answer, like the mention of “marinated lamb, skewered & grilled”. The questions have low ambiguity because it provides a lot of information and uses simple vocabulary.

GREEK FOOD & DRINK (100% accuracy):

Questions that were answered correctly:

- The name of this dish of marinated lamb, skewered & grilled, comes from the Greek for "skewer" & also starts with "s"
- Because it's cured & stored in brine, this crumbly white cheese made from sheep's milk is often referred to as pickled cheese
- This clear Greek liqueur is quite potent, so it's usually mixed with water, which turns it white & cloudy

For the category “UCLA CELEBRITY ALUMNI”, the correct answered questions have specific and unique details and keywords about the celebrities, including their role in TV shows, academic background, and notable achievements. Also, there are specific time periods, which increases the chance to find the correct Wikipedia page. The incorrect answer question does not have such specific details or time references. Furthermore, there could be more athletes that have the same achievements that are described in the question. I think that’s why the question was answered wrong.

UCLA CELEBRITY ALUMNI (80% accuracy):

Questions that were answered correctly:

- Neurobiologist Amy Farrah Fowler on "The Big Bang Theory", in real life she has a Ph.D. in neuroscience from UCLA
- Attending UCLA in the '60s, he was no "Meathead", he just played one later on television
- This blonde beauty who reprised her role as Amanda on the new "Melrose Place" was a psychology major
- For the brief time he attended, he was a rebel with a cause, even landing a lead role in a 1950 stage production

Questions that were answered incorrectly:

- This woman who won consecutive heptathlons at the Olympics went to UCLA on a basketball scholarship

For the category ““TIN” MEN”, the questions are well structured and detailed. The wrong answers were “Politics of Russia” instead of “Vladimir Putin|Putin” and “Fifth Beatle” instead of “George Martin”. I think that the fact that I did not clear very well the Wikipedia pages before indexing might influence these answers, because the system might find references to other Wikipedia pages and lead to a wrong answer.

“TIN” MEN (50% accuracy):

Questions that were answered correctly:

- This Italian painter depicted the "Adoration of the Golden Calf"

- You can't mention this shortstop without mentioning his double-play associates Evers & Chance

Questions that were answered incorrectly:

- He served in the KGB before becoming president & then prime minister of Russia
- He earned the "fifth Beatle" nickname by producing all of the Beatles' albums

For the category “'80s NO.1 HITMAKERS”, there were no correct answers. All the questions were answered with “Music of Zimbabwe”. The clue is short and does not have specific details other than the song title, combining this with the fact that the category is also used in the query, I think that the keywords from the category, like: “80s” and “HITMAKERS”, might mislead the search.

'80s NO.1 HITMAKERS (0% accuracy):

Questions that were answered incorrectly:

- "Father Figure"
- "Beat It"
- "Man In The Mirror"
- "Miss You Much"
- "Rock With You"

5. Improving retrieval

At first, I tried to use Stanford Named Entity Recognition (NER) since I saw that a lot of questions and answers include places, celebrities, or companies. I decided to compare the most frequent label used in the query and the most frequent label used in the top 10 documents returned by Lucene after the search. If the most frequent label used in the query matches with one from the documents, then I would use that document as the final answer. The problem was that the queries are short and most of them did not contain any word that was categorized by the Stanford NER, so the results were very bad.

The second try was to enhance the query using a synonym after each noun that was present. I used the WordNet Library to search for the first synonym when a noun was encountered in the query. Doing the search with the enhanced query was not helpful either, because the synonyms did not have the same context meaning in most of the cases. With this method I also had very bad results.

After analyzing the top 5 documents titles retrieved by Lucene for every query, I found out that when the correct answer wasn’t the first result it was present just twice, as the second result and as the fourth result. Also, I noticed, that in most of the cases the document scores assigned by Lucene were very high compared to the others for the first result. But there were some cases in which the document scores were similar for all the results. When the difference in document score between the first and second results was smaller than 1, I generated Bert embeddings using a pre-trained bert model. I generated the embeddings for the query and for the content of the first and second result. Then I compared each content embeddings with the one for the query using cosine similarity, and used the result which had the higher similarity score. After doing this, I only got one more correct answer, P@1 being 0.32.