RECREATING WATSON

Or just a part of it

ABSTRACT

Using NLP libraries like Lucene and Stanford's Core NLP along with a few heuristics to improve retrieval, I recreated Watson's Jeopardy Answering mechanism.

Shreyas Khandekar

CSC 483

Table of Contents

Introduction

This program aims to replicate a part of IBM's Watson where it uses Information Retrieval strategies to answer Jeopardy questions.

The source code and running instructions can be found on the following GitHub:

https://github.com/ShreyasKhandekar/jeopardySolver

Indexing and Retrieval

We use a similar strategy to that of IBM's Watson of scouring Wikipedia pages and answering queries based on the top page match.

How we handle Wikipedia pages

Each Wikipedia page consists of various kinds of texts that must be processed differently. At the outset, each Wikipedia page contains at least one of the following:

1. A Title
    a. This is the title of the document we create in Lucene
    b. Titles are delimited by two square brackets on either side
2. Section Titles
    a. These titles are originally ignored but we will later see a use for them to improve retrieval.
    b. Section titles are delimited by two equal signs (==) on each side
    c. Subsection titles may have more than two equal signs delimiting them
3. Contents
    a. The contents are what our primary search is based on.
4. Links and Redirects:
    a. Links and redirects are of the format [tpl] ... [/tpl] - These seem to be for any text that is a hyperlink
        i. Most of the tpl links are citations to web pages, journals, and books, and thus contribute useful information to the content of the page. This is why I chose to include the content between these [/tpl] tags while stripping out tags themselves.
    b. #REDIRECT
        i. These links simply redirect the Wikipedia page to another Wikipedia page.
        ii. Since this means that the page contains no relevant information, it is useless to search.
        iii. Therefore, we do not index pages that are redirects.
5. Embeddings
    a. [[File:…]] and [[Image:…]]
        i. Both of these items are just links and more often than not, the information about them is not relevant to the search. Therefore we ignore these entirely.
        ii. Since these delimitations are the same as Wikipedia page titles, I had to take special care to ensure that these are not treated as Wikipedia pages.
6. Special characters
    a. There are some characters like & which just don't play nice with Lucene.

b. To deal with them we just escape them using Lucene's in-built function QueryParser.escape()

How we handle questions

Each Question is made up of multiple elements where all questions must have the following components:

1. A category
    a. This is the category of the question which dictates the genre of the question that follows. Since this is an important piece of information we prepend the actual question with the category to get richer results with more context based on the information available.
    b. Sometimes a category is accompanied by an explanation of what it is which is denoted in parentheses by "(Alex: … clue …)".
    c. If we have this clue, we parse just the clue, removing the parenthesis and "Alex:" from the clue.
2. A Question
    a. This is the actual question (or if we're talking in jeopardy terms, this would technically be the answer, but for simplicity's sake I will stick to calling this the question)
    b. This is what we are trying to answer and therefore this is added to the query

These things together make up the question that we pass to our question answering system.

Measuring Performance

Measure the performance of your Jeopardy system, using one of the metrics discussed in class, e.g., precision at 1 (P@1), normalized discounted cumulative gain (NDCG), or mean reciprocal rank (MRR). Note: not all the above metrics are relevant here! Justify your choice, and then report performance using the metric of your choice.

In the game of jeopardy, the only relevant metric is the correctness of our answer. Therefore since this is a system to answer jeopardy questions, we must only care about the correctness of our top answer since that is what we will be evaluated on.

This is known as Precision at 1 (P@1)

Explanation of Each Index

Standard Analyzer with Lemmatization with only the first 15 lines

This uses Lucene's Standard Analyzer which removes stop words and lowercases the generated tokens. Lemmatization is done using Stanford's CoreNLP library. This index only works with the first 15 lines of each document because they are usually the most relevant to the content of the document.

Whitespace Analyzer with Lemmatization with only the first 20 lines

The uses Lucene's Whitespace Analyzer which does not do anything to the contents, just splits it on whitespace and analyses that. Lemmatization is done using Stanford's CoreNLP library. This index only works with the first 20 lines of each document because they are usually the most relevant to the content of the document.

Whitespace Analyzer with Lemmatization on the whole document

This also uses Whitespace Analyzer and Lemmatization using CoreNLP but it does so for the whole document.

Whitespace Analyzer without Lemmatization on the whole document

This just uses the Whitespace Analyzer on the whole document.

Standard Analyzer without Lemmatization on the whole document

This uses the Standard Analyzer on the whole document.

English Analyzer without Lemmatization on the whole document

White the Standard Analyzer has StandardTokenizer, StandardFilter, LowercaseFilter, and StopFilter. EnglishAnalyzer rolls in an EnglishPossesiveFilter, KeywordMarkerFilter, and PorterStemFilter. So, it does stem using the Porter Stemmer as well. We do not do any lemmatization here and we include the whole document in the index.

Precision at 1

For the example dataset of 100 questions, I used a wide variety of different techniques to gauge which yields the best results.

- Standard Analyzer with Lemmatization with only the first 15 lines: 20%
- Whitespace Analyzer with Lemmatization with only the first 20 lines: 22%
- Whitespace Analyzer with Lemmatization on the whole document: 25%
- Whitespace Analyzer without Lemmatization on the whole document: 11%
- Standard Analyzer without Lemmatization on the whole document: 29%
- English Analyzer without Lemmatization on the whole document: 30%

So clearly our English analyzer is the best system which is answering 30% of the questions correctly.

Changing the Scoring function

Replace the scoring function in your
system with another. For example, by default Lucene uses a probabilistic scoring
function (BM25). You can replace this default choice with cosine similarity based on tf.idf weighting. How does this change impact the performance of your system?

If we change our scoring function from BM25 to tf.idf, the performance of our system changes as follows:

- Standard Analyzer with Lemmatization with only the first 15 lines: 5%
- Whitespace Analyzer with Lemmatization with only the first 20 lines: 5%
- Whitespace Analyzer with Lemmatization on the whole document: 2%
- Whitespace Analyzer without Lemmatization on the whole document: 0%
- Standard Analyzer without Lemmatization on the whole document: 3%
- English Analyzer without Lemmatization on the whole document: 2%

Which is horrible compared to BM25.

This shows that BM25 can capture the semantics of the questions and answers much better in its similarity than the Classic Similarity which is just a basic tf.idf rating.

Improving Retrieval

To attempt to improve the retrieval of this system I will attempt to use the section titles because they also contain important information about the Wikipedia Document.

The way I am approaching this is to make section titles into a tier and since Lucene can support tiers by just adding it as another searchable field within the document, we can add another tier to our search.

How the tiered index works

Option1:

Search content first, if you get a score above a certain threshold then return that document as the answer. If not, then go ahead and search the section titles of the page.

For each index, I tried to optimize this threshold by testing different values for the score which would yield the best result.

The results for the different indices are as follows, the threshold for each type of index is in paratheses:

- Standard Analyzer with Lemmatization with only the first 15 lines (20): 16%
- Whitespace Analyzer with Lemmatization with only the first 20 lines (17): 21%
- Whitespace Analyzer with Lemmatization on the whole document (10): 24%
- Whitespace Analyzer without Lemmatization on the whole document (5): 11%
- Standard Analyzer without Lemmatization on the whole document (15): 28%
- English Analyzer without Lemmatization on the whole document (15): 28%

We can see that this attempt at improving retrieval lowered our performance here.

Option 2:

You can search both tiers simultaneously and combine the two scores for a total score for the document.

By combining the two scores, I mean summing them up. And the sorting of the documents based on the top max score.

- Standard Analyzer with Lemmatization with only the first 15 lines: 17%
- Whitespace Analyzer with Lemmatization with only the first 20 lines: 20%
- Whitespace Analyzer with Lemmatization on the whole document: 20%
- Whitespace Analyzer without Lemmatization on the whole document: 9%
- Standard Analyzer without Lemmatization on the whole document: 22%
- English Analyzer without Lemmatization on the whole document: 22%

We can see that this has also lowered our P@1.

Based on my observations here I can say that the section titles are not a good tier to be searching to get the right answer. We should ignore them altogether.

Error Analysis

Perform an error analysis of your best system. How many questions were answered correctly/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? Try to group the errors into a few classes and discuss them.
Lastly, what are the impact of stemming and lemmatization on your system? That is, what is your best configuration: (a) no stemming or lemmatization; (b) stemming, or (c) lemmatization? Why?

Since across all systems, the English Analyzer works best, I will focus on the error in this method.

My Best system answered 30 questions correctly out of 100.
Here are the results for the questions and their subsequent answers returned by the system, for reference I am also including the solution (Gold).
Some questions have more than one acceptable answer therefore the solution is a list of acceptable answers.

We have already seen the impact of stemming and lemmatization by the results of the 5 different indices above.

Output

See Output.md

Observing the Errors

On Analyzing the output by sight and looking at the questions answered correctly, we can see that when questions list specific achievements or details about a certain topic like if they won a certain award in a certain year or were involved in a certain effort during a certain time, then the system can answer correctly.

This is likely because that combination of year and event is unique to the right answer so much so that the system is easily able to figure out the right document.

As for the incorrect ones, there is a semantic disconnect between the answer and the question. For example, for questions where the task is to name the parent company, the system always answers with the name of the original company because it does not understand that it is supposed to name the parent company. Another example would be to name the state in which an art museum is located.

Understanding the Errors

The questions that we missed follow a defined pattern in this case. This is because for the cases described above the question is very short (name of a company or a museum) and the answer is completely textually disjoint from the question. Therefore, BM25 is simply unequipped to handle such a question. What we would need in this case would be a transformer that can bridge that semantic gap.