CSC 483/583: Programming Project (200 pts)

Building (a part of) Watson

Author: Sixiang Zhang

1. Description of the command line to run the python code, with an example :

Run indexManager for creating indexes

Run TextSearch for answer questions from the answer.txt

1. Description of the code. You don’t have to describe every function implemented. But you should describe the main part of the code and indicate where each question is addressed.

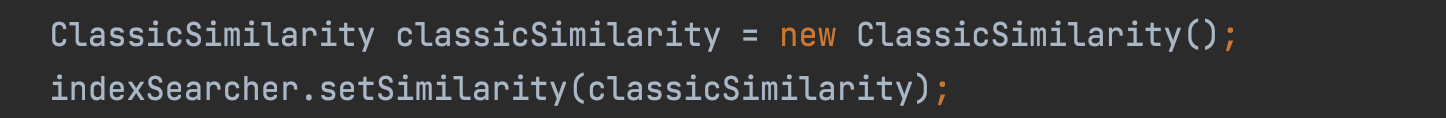
Question 1 (Different analyzer and stop words list)Text

Description automatically generated

Question 2 (P@1 function from scratch)Text

Description automatically generated

Question 3 (Score function)



Question 4 (Stem/lemmatization)

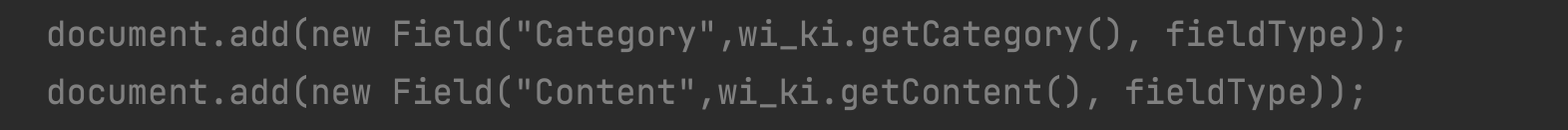
Text

Description automatically generated

Question 5(positional index)

Graphical user interface, text

Description automatically generated



1. ResultsGraphical user interface, application, table, Excel

   Description automatically generated
2. Answers for all questions that do not require programming.
3. (100 pts) Indexing and retrieval: Index the Wikipedia collection with a state of the art Information Retrieval (IR) system such as Lucene (http://lucene.apache. org/) or Terrier (http://terrier.org/). Make sure that each Wikipedia page appears as a separate document in the index (rather than creating a document from each of the 80 files).
4. Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.).

Each document represents each record from the wiki-dataset. And when we make the indexes for the document, we can set different field for different columns. For example, the StringField and Text field both for the String type date, however, the former does not be tokenized. And the propose for the tokenization is for the search. In this project, we do not need search or query by the answers, thus, I decided not to tokenize the answer column.

Stemming, lemmatization, stop words, etc.:

Then, there are many built-in Analyzers from Lucene. What is more, Lucene's analyzers are consist of tokenizers and filters. Different analyzers have different combinations of tokenizers and filters. I tried three different analyzers (StandardAnalyzer, SimpleAnalyzer, and EngligshAnalyzer) and one customer analyzer. The difference between the SimpleAnalyzer and StandardAnalyzer is that StandardAnalyzer has a stopwords filter, SimpleAnalyzer does not. Moreover, EnglishAnalyzer has stemming and lemmatization, but the other two do not. My customer analyzer is for the later adjusting the stemming and lemmatization effect. One more thing is that, the default stopwords is that: "a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "such", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with". However, after this stopwords list, I saw that the indexes have more stopwords. I added "from","which","also","ha","on","he","it" into the stop words list.

Graphical user interface, application, table

Description automatically generated

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

There are many reference links, file paths, or picture paths that we need to remove from Wikipedia content. Most of them have similar format, for example, they start with </ref> or [ref] or [/ref]. Moreover, They are some strange characters, which are not English; I remove them from the content. Furthermore, we need to remove” \_” from the words, which sometimes creates a huge “term” that we may not search. Moreover, there is a GUI called luke, which shows how Lucene stores the indexes. For example, as the picture shows (from luke) below. I need to remove the [#redicrect] or [#Redicrect] or #REDICRECT from content. Last but not least, always check the blank line. The Wikipedia content contains a lot of blank lines needed to be removed.

Graphical user interface, table

Description automatically generated

1. 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?

For the query part, I use the BooleanQuery from Lucene. BooleanQuery can compare multiple queries to search from the index. It has settings should, must, and must not. Here, I give the two queries with should. And I used all the words in the clue. The reason is that most of the documents are very large. For example, Michal Jackson has more than 6000 words. And all the clues are relatively short, and I have to use all the words to have better hits.

1. Are you using the category of the question?

A screenshot of a computer

Description automatically generated with medium confidenceYes. If I am not using the category, the total hits at rank 1 are worse. I put the results as follows:

1. (25 pts) 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.

I choose the precision at 1(P@1) in this project as the measuring performance method. The reason is that the system I wrote is a QA system; people who use the QA system are looking for the first place answer. No one cared about the later answers.

I implement this method from scratch. Since Lucene has the functions topDocs, we can use code,topDocs.totalHits, to check the total number of hits from the index. And we can store all the results. Here, I just used the rank 0 results, the first place, and compared it with the correct answer from the question.txt.

1. (25 pts) 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?

Changing the scoring function from BM25 to tf.idf weighting worsens my system. First, It can not predict anything correctly in the first place. And I rose to the rank 100; it only hit 22, which is much worse than before.

A picture containing text, document, screenshot

Description automatically generated

1. a) (50 pts) 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.

The best results are that my QA system can answer the following questions, which are 20 questions from 100 questions. Before discussing these, I firstly wrote a function to find the document from the wiki-dataset. And for the common of most correct answers is that clue has the particular terms that help choose the correct indexes. For example, Jackie Joyner-Kersee has three terms basketball, Olympics, and UCLA from the clue and category. However, those questions lack such terms as the particular terms contained by the clue, category, or the wiki-dataset for the wrong answers. And one more thing, the documents like Michal Jackson, which has more than 6000 words. The clue is "Man in the Mirror," which shows almost the end of the document, which is why it is not ranked 1.Table

Description automatically generated with medium confidence

b) Lastly, what is 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?

a)no stemming or lemmatization

Using a simple analyzer is the same as the effect of no stemming or lemmatization.

I created a customed Analyzer by adding different filters to stemming or lemmatization.

b)stemming c)lemmatization

From the picture following, I found that stemming is the most impact on my system. But, the best configuration is with stem and lemmatization.

A screenshot of a computer

Description automatically generated with low confidence

1. (50 pts) Improving retrieval (GRAD STUDENTS ONLY): Improve the above standard IR system using natural language processing and/or machine learning. For this task you have more freedom in choosing a solution and I encourage you to use your imagination. For example, you could implement a positional index instead of a bag of words; you could use a parser to extract syntactic dependencies and index these dependencies rather than (or in addition to) words; you could use supervised learning to implement a reranking system that re-orders the top 10 (say) pages returned by the original IR system (hopefully bringing the correct answer to the top).

In this project, first, I tried to "enlarge" the queries and categories by searching every term's (filtered by the stopword\_filter) synonyms. However, the results became worse, and the reason is that I may create different queries for searching, leading to the wrong answers.

Later, I used the positional index instead of a bag of words and proximity Searches. Lucene supports the proximity Searches, which use the "~" symbol at the end of a phrase. (For example, The format is that "Jakarta apache "~10, which is to search for an "apache" and "Jakarta" within ten words of each other.) Moreover, getting a window of words around a match can be helpful in a Question Answering System. However, the default of Lucene fields does not contain the position information. You need to use the field methods and set index options to the doc&freq&position information.

The result is a little better than the bag of words. Total hits in rank 1 are 20, which achieve 20% accuracy at position 1, and the previous is 19.

A screenshot of a computer

Description automatically generated with low confidence