Watson Project

1. How to run the program:

- Clone git repository https://github.com/mauherrerag/Watson.git
- Download and unzip index files from https://arizona.box.com/s/mqtkkho9myv5lbz40ei6mku3fiawrcmx
- Place index directory inside *src/main/resources*
- If you plan to build the indices, download the wiki-subset of articles from https://arizona.box.com/s/0s9ybt3zo2vn42il9cl7kkawlb1x824o
- Place wiki-subset directory inside *src/main/resources*
- Cd into the Watson directory and run *mvn verify*

2. Building the index

For this project, I decided to create 3 different indices. One of them does no stemming or lemmatization. Another only does stemming, and the final one only does lemmatization. I chose not to remove stop words.

The Wikipedia articles were fairly well organized. It was easy to distinguish when one article ended and another one began with the use of brackets to identify article titles. I believe I would have gotten better results had I cleaned up the contents of the articles, since a lot of them contained sections that were not relevant. Some of them were references, some others were inline tags that resembled HTML. Due to the variety of characters that identified this irrelevant information, I chose to leave it as is.

I chose to create three fields per document going into the index: the title of the article (non-tokenized), the categories for the article, and the contents concatenated with the categories. The last two were tokenized using Stanford's CoreNLP library.

The queries were built using the contents of the question, as well as the category by concatenating them into a single string. Then it was given the same name as the TextField stored in the index.

3. Measuring performance

For this project I chose to use precision at 1 and mean reciprocal rank. I think P@1 is the best metric for this project because it reflects the same number of questions answered correctly.

Index \ Similarity Function	BM25	TFIDF	Boolean	DFIS	DFRS	Dirichlet	Jelinek Mercer
No stemming, no lemma	0.21	0.01	0.12	0.29	0.15	0.33	0.22
Only stemming	0.2	0	0.13	0.31	0.18	0.34	0.23
Only lemmatization	0.19	0	0.11	0.29	0.17	0.34	0.22

As we can see, the best results were obtained from the indices using either stemming or lemmatization and the Dirichlet similarity function.

4. Error analysis

My best system was using an index with either stemming or lemmatization and the Dirichlet similarity function. I think that there is a better chance of getting a hit with lemmas or stems of words because it reduces the number of unique words to be matched. However, I think I would have gotten a higher score if I had cleaned up the Wikipedia articles better. The only cleanup I did was removing the lines including sub headers for the references or see also sections.

5. The code

This program includes four Java classes:

1. LuceneIndex: This class builds the index or indices indicated.

   a. buildIndex(): This method uses the static variables holding the paths where the indices will be stored to iterate through them and create the indices. It iterartes through the files in the wiki-subset directory and passes each one to parseDocument().
   b. parseDocument(): This method takes in the path for the document to be parsed, as well as the IndexWriter object where it will be written to. It goes through the document line by line, identifying if a line contains the title, categories, or article content. Then, once all three variables are set, they are passed to addDocToIndex().
   c. addDocToIndex(): This method takes in the title, categories, contents of the article, and the IndexWriter object and it lemmatizes/stems the strings accordingly and adds them to the Document object. Finally, it writes the document into the index.
   d. stemTerm(): This method uses a PorterStemmer to stem a given string and returns it. This method is static in order to use it in the QueryEngine class.
2. QueryEngine: This class takes care of querying an index using a questions file containing n number of queries. The constructor takes in the indexPath, and flags to either lemmatize or stem.
   a. setSimilarityFunction(): This method assigns a given similarity function to the query engine. This method takes in an instance of Similarity.
   b. playJeopardy(): This method reads in the queries file and parses the different parts into variables. Then it passes those values (category and query) to runQuery(). This method also keeps track of the P@1 and MMR values and total correct answers.
   c. runQuery(): This method queries the index with the query passed after it has been lemmatized or stemmed when necessary. This returns a list of ResultClass objects ranked by score.
   d. parseQuery(): This method lemmatizes/stems a query when necessary.
3. ResultClass: This class acts as a container for the results in order to put together a document and its score.

4. Watson: This class acts as the “view” in this model. It can create all indices or run queries on existing indices.
    a. playJeopardy(): This method iterates through the indices and calculates the scores for all the different types of similarity functions.
    b. buildIndices(): This method builds all three indices. It takes a few hours, especially when building the index with lemmas.

6. Code results

==========STARTING WATSON==========

========CALCULATING INDEX src/main/resources/index/none========

--------Similarity Function: NONE

P@1: 21/100 = 0.21

MMR: 0.21

--------Similarity Function: org.apache.lucene.search.similarities.BM25Similarity

P@1: 21/100 = 0.21

MMR: 0.21

--------Similarity Function: org.apache.lucene.search.similarities.ClassicSimilarity

P@1: 1/100 = 0.01

MMR: 0.01

--------Similarity Function: org.apache.lucene.search.similarities.BooleanSimilarity

P@1: 12/100 = 0.12

MMR: 0.12

--------Similarity Function: org.apache.lucene.search.similarities.DFISimilarity

P@1: 29/100 = 0.29

MMR: 0.29

--------Similarity Function: org.apache.lucene.search.similarities.DFRSimilarity

P@1: 15/100 = 0.15

MMR: 0.15

--------Similarity Function: org.apache.lucene.search.similarities.LMDirichletSimilarity

P@1: 33/100 = 0.33

MMR: 0.33

--------Similarity Function: org.apache.lucene.search.similarities.LMJelinekMercerSimilarity

P@1: 22/100 = 0.22

MMR: 0.22

========CALCULATING INDEX src/main/resources/index/stem========

--------Similarity Function: NONE

P@1: 20/100 = 0.2

MMR: 0.2

--------Similarity Function: org.apache.lucene.search.similarities.BM25Similarity

P@1: 20/100 = 0.2

MMR: 0.2

--------Similarity Function: org.apache.lucene.search.similarities.ClassicSimilarity

P@1: 0/100 = 0.0

MMR: 0.0

--------Similarity Function: org.apache.lucene.search.similarities.BooleanSimilarity

P@1: 13/100 = 0.13

MMR: 0.13

--------Similarity Function:
org.apache.lucene.search.similarities.DFISimilarity
P@1: 31/100 = 0.31
MMR: 0.31
--------Similarity Function:
org.apache.lucene.search.similarities.DFRSimilarity
P@1: 18/100 = 0.18
MMR: 0.18
--------Similarity Function:
org.apache.lucene.search.similarities.LMDirichletSimilarity
P@1: 34/100 = 0.34
MMR: 0.34
--------Similarity Function:
org.apache.lucene.search.similarities.LMJelinekMercerSimilarity
P@1: 23/100 = 0.23
MMR: 0.23

========CALCULATING INDEX src/main/resources/index/lemma========
--------Similarity Function: NONE
[main] INFO edu.stanford.nlp.tagger.maxent.MaxentTagger - Loading POS tagger from edu/stanford/nlp/models/pos-tagger/english-left3words-distsim.tagger ... done [0.5 sec].
P@1: 19/100 = 0.19
MMR: 0.19
--------Similarity Function:
org.apache.lucene.search.similarities.BM25Similarity
P@1: 19/100 = 0.19
MMR: 0.19
--------Similarity Function:
org.apache.lucene.search.similarities.ClassicSimilarity
P@1: 0/100 = 0.0
MMR: 0.0

--------Similarity Function:
org.apache.lucene.search.similarities.BooleanSimilarity

P@1: 11/100 = 0.11

MMR: 0.11

--------Similarity Function:
org.apache.lucene.search.similarities.DFISimilarity

P@1: 29/100 = 0.29

MMR: 0.29

--------Similarity Function:
org.apache.lucene.search.similarities.DFRSimilarity

P@1: 17/100 = 0.17

MMR: 0.17

--------Similarity Function:
org.apache.lucene.search.similarities.LMDirichletSimilarity

P@1: 34/100 = 0.34

MMR: 0.34

--------Similarity Function:
org.apache.lucene.search.similarities.LMJelinekMercerSimilarity

P@1: 22/100 = 0.22

MMR: 0.22

==========TERMINATED==========