Rohan Sinha

Mihai Surdeanu

CSC 483

May 1, 2019

Building (a part of) Watson

How to run the code:

To run the code in command line:

1) pull/clone the code from: github.com/rohansinha16/Watson
2) download the index folder from (it includes 6 indices so it's 3.76 GB): https://drive.google.com/file/d/1a6zP2Id01Ow9iGngDItK6XVy2mZHJ3kp/view?usp=sharing
3) put index folder in the pulled/cloned folder
4) navigate to the location of the folder from cmd line/terminal
5) type the following:

```
$ mvn compile
$ mvn test
```

If done correctly, for each index in the index folder (a total of six), a test will run that calculates the precision at 1 (P@1) and mean reciprocal rank (MMR) using four scoring methods:

Vector Space Model and tf/idf (default)
Boolean Model
BM25 Model
Jelinek Mercer Model

The Code:

There are four classes in this project. They are all documented fairly thouroghly.

Watson.java – includes main function, generally calls QueryEngine or IndexCreator. Has several settings as variables that will be passed to QueryEngine/IndexCreator for easy use of the program.

File settings:

String indexPath - index location name, used by both
String queries – queries filename, used by QueryEngine
String input_dir – wiki files folder. used by IndexCreator

Settings:

boolean useLemma – should index/queries use lemmas or not, used by both.
boolean useStem – should index/queries use stems or not, used by both.
boolean removeTpl – should tpl tags be removed, used by IndexCreator

Functionality Settings:

boolean createIndex – should an index be created by IndexCreator
boolean runQueries – should queries be tested by QueryEngine

QueryEngine.java – requires String indexPath, boolean useLemma, boolean useStem. Optional Similarity s (used for scoring). Important functions:

```
/* runQueries
 * @param String filename - name of the file with queries
 * @return String result - string including the P@1 and MRR scores
 *
 * runs all the queries in the file filename with queries in the form of:
 * |============|
 * |CATEGORY      |
 * |query             |
 * |answer           |
 * |============|
 *
 * Calculates P@1 and MMR as well
 */
public String runQueries(String filename)
```

```
/* runQuery runs the query on the index at indexPath, returns top 10 results, and scores
 * using the Optional<Similarity> s
 * @param String query - the query to test
 * @return List<ResultClass> - top 10 hits
 */
public List<ResultClass> runQuery(String query)
```

IndexCreator.java – requires String indexPath, boolean useLemma, boolean useStem, boolean removeTPL. Important functions:

```
/* Reads all files in a given directory and indexes them
 * @param String directory - directory of files to parse
 */
public void parseFiles(String directory)
```

```
/* Reads a file and creates an index
 * @param String filename - the file to index
 * @param IndexWriter w - the index writer
 */
public void parseFile(String filename, IndexWriter w)
```

```
/* tplRemove, used to remove tpl tags
```

```
 * @param String line - the current line being processed
 * @param Stack<Boolean> tpl - stack to keep track of open and closing tpl tags
 * @param StringBuilder result - parts of line not inside tpl tags will be added to result
 */
private void tplRemove(String line, Stack<Boolean> tpl, StringBuilder result)

/* numAtEnd
 * @param String src - the string to check
 * @param String token - the token to look for at the end
 * @return int - the number of times token appears at the end of String
 * Used for tpl tags at ends of lines as split doesn't do the trick
 */
private int numAtEnd(String src, String token)

/* addDoc
 * @param IndexWriter w - the index writer
 * @param String title - title of doc
 * @param String categories - categories of doc
 * @param String text - content of doc
 * Creates new document for the parameters and adds it to index. Also takes care of
 * stemming/lemmatization based on the booleans passed to constructor
 */
private void addDoc(IndexWriter w, String title, String categories, String text)
```

ResultClass.java – simple class to store documents and scores. Same as from Project 3.

Indexing and retrieval

To prepare the terms for indexing I tried nothing, stemming, and lemmatization. When parsing each page, the title, categories, and content are separated. Upon indexing, the title and categories get their own field but are also included in the text field along with the rest of the content. During queries, I use the category and clue (all words) from the query combined to search within the text field.

One issue specific to Wikipedia content I found was tpl tags. To address the tags, I created an option in my program to remove the tags and their enclosed contents upon index creation. The methodology essentially required splitting lines by the tags and using a stack to keep track of the tags due to some tags being nested.

In addition, I noticed that some attachments and other info would be in the same form of the titles “[[title]]” which wasn’t a great design decision. So, I added a check to make sure the brackets were at the beginning and end of the line. There were also ref tags that I wasn’t sure how to deal with.

As I created six indices, 3 with and without tpl tags for nothing, stemming, and lemmatization they are put inside the index folder with the following names.

==none==
Comments: No stemming/lemma, including tpl tags.

==noneNoTPL==
Comments: No stemming/lemma, removing tpl tags.

==stem==
Comments: stemming, including tpl tags.

==stemNoTPL==
Comments: stemming, removing tpl tags.

==lemma==
Comments: lemmatization, including tpl tags.

==lemmaNoTPL==
Comments: lemmatization, removing tpl tags.

Measuring Performance:

I used the precision at 1 (P@1) and mean reciprocal rank (MRR) scores. I included P@1 because in the context of the question it's the proportion of answers that were correct. However, I think the MMR is a better metric because it's perfect for when there's only one correct answer.

Changing the Scoring Function:

I tried using 4 scoring methods, the default scoring function (vector space model with tf/idf weighting), the Boolean model, the probabilistic BM25, and Jelinek-Mercer scoring. The Boolean model consistently performed worse than other models with the chosen metrics (P@1 and MMR) from "Measuring Performance", as expected. BM25 on the other hand had the exact same scores as the default scoring. Lastly, Jelinek Mercer performed consistently better than all other scoring functions for the given metrics.

Results:

Configuration	TPL Tags Included	Metric	Default (VSM/tfidf)	Boolean	BM25	Jelinek Mercer
None	w/ TPL	P@1	0.21	0.08	0.21	0.28
		MMR	0.26	0.14	0.26	0.34
	w/out TPL	P@1	0.19	0.09	0.19	0.29
		MMR	0.25	0.14	0.25	0.35

Stemming	w/ TPL	P@1	0.2	0.14	0.2	0.26
		MMR	0.26	0.17	0.26	0.31
	w/out TPL	P@1	0.22	0.12	0.22	0.27
		MMR	0.28	0.17	0.28	0.34
Lemmatization	w/ TPL	P@1	0.2	0.12	0.2	0.26
		MMR	0.25	0.16	0.25	0.30
	w/out TPL	P@1	0.21	0.11	0.21	0.29
		MMR	0.27	0.16	0.27	0.34

Error Analysis

The best version in terms of correct/incorrect answers was using Jelinek-Mercer scoring with either no stemming/lemmatization or lemmatization, both with tpl tags removed and getting 29/100 queries correct. It is important to mention that scores within the Jelinek-Mercer scoring with different indices produced results within 3 correct queries of each other, so not a huge difference, but also not a huge number of tests. In terms of MMR, none w/out tpl tags had the highest with a score of 0.35, but none w/tpl, stemming w/out tpl, and lemmatization w/out tpl all had similar MMRs with 0.34.

I think the questions can be answered by such a simple system because the format of the queries and because simple systems often work well due to being well designed and a lack of noise. The format of the queries including a category and the Wikipedia pages also having categories is also helpful and an easy opportunity to get relevant documents. Then the clue can be used to home in on results.

Error Classes:
Short clues: Shorter clues often resulted in wrong answers. For example: NAME THE PARENT COMPANY, Crest toothpaste or '80s NO.1 HITMAKERS, 1989: "Miss You Much".

Common words: Clues or categories with fairly common words seemed to get more wrong answers. For example: GOLDEN GLOBE WINNERS, In 2001: The president of the United States on television. This query returned several United States constitution amendments. For example, CAPITAL CITY CHURCHES, Matthias Church, or Matyas Templom, where Franz Joseph was crowned in 1867. This query returned several churches.

Shared clues: Some clues could refer to or have some words that refer to multiple pages. COMPLETE DOM-INATION, This sacred structure dates from the late 600's A.D. This could refer to many things. The church example from the last class could also fit here.

Sample Output

The results below are summarized in the table above but include specific index directory names in the index folder.

```
$ mvn test
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< edu.arizona.cs:SinhaRohanWatson >-------------------
[INFO] Building SinhaRohanWatson 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ SinhaRohanWatson ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory
/Users/Home/Documents/TextRetrieval/FinalProject/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ SinhaRohanWatson ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
SinhaRohanWatson ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory
/Users/Home/Documents/TextRetrieval/FinalProject/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ SinhaRohanWatson -
--
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.0:test (default-test) @ SinhaRohanWatson ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running edu.arizona.cs.TestStemNoTpl
Running similarity 1.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Running similarity 2.
Running similarity 3.
Running similarity 4.

For the index at index/stemNoTpl
Using the default Vector Space Model and tf/idf:
        P@1: 22/100 = 0.22
        MMR: 0.2792142857142857
Using the Boolean Model:
        P@1: 12/100 = 0.12
        MMR: 0.16808333333333333
Using the BM25 Model:
        P@1: 22/100 = 0.22
        MMR: 0.2792142857142857
Using the Jelinek Mercer Model:
        P@1: 27/100 = 0.27
        MMR: 0.3395
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 13.822 s - in
edu.arizona.cs.TestStemNoTpl
[INFO] Running edu.arizona.cs.TestLemmaNoTpl
Running similarity 1.
Running similarity 2.
Running similarity 3.
Running similarity 4.

For the index at index/lemmaNoTpl
Using the default Vector Space Model and tf/idf:
        P@1: 21/100 = 0.21
        MMR: 0.26773412698412696
Using the Boolean Model:
        P@1: 11/100 = 0.11
        MMR: 0.16032936507936504
Using the BM25 Model:
        P@1: 21/100 = 0.21
        MMR: 0.26773412698412696
Using the Jelinek Mercer Model:
        P@1: 29/100 = 0.29
        MMR: 0.3365277777777777
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 9.177 s - in
edu.arizona.cs.TestLemmaNoTpl
[INFO] Running edu.arizona.cs.TestLemma
Running similarity 1.
Running similarity 2.
Running similarity 3.
Running similarity 4.

For the index at index/lemma
Using the default Vector Space Model and tf/idf:
        P@1: 20/100 = 0.2
        MMR: 0.2525952380952381
Using the Boolean Model:
        P@1: 12/100 = 0.12
        MMR: 0.15959523809523807
Using the BM25 Model:
        P@1: 20/100 = 0.2
        MMR: 0.2525952380952381
Using the Jelinek Mercer Model:
        P@1: 26/100 = 0.26
        MMR: 0.303361111111111
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 10.722 s - in
edu.arizona.cs.TestLemma
[INFO] Running edu.arizona.cs.TestNoneNoTpl
Running similarity 1.
Running similarity 2.
Running similarity 3.
Running similarity 4.

For the index at index/noneNoTpl
Using the default Vector Space Model and tf/idf:
        P@1: 19/100 = 0.19
        MMR: 0.25223412698412695
Using the Boolean Model:
        P@1: 9/100 = 0.09
        MMR: 0.14128968253968252
Using the BM25 Model:
        P@1: 19/100 = 0.19
        MMR: 0.25223412698412695
Using the Jelinek Mercer Model:
        P@1: 29/100 = 0.29
        MMR: 0.3518333333333332
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 9.331 s - in
edu.arizona.cs.TestNoneNoTpl
[INFO] Running edu.arizona.cs.TestStem
Running similarity 1.
Running similarity 2.
Running similarity 3.
Running similarity 4.

For the index at index/stem
Using the default Vector Space Model and tf/idf:
        P@1: 20/100 = 0.2
        MMR: 0.25665079365079363
Using the Boolean Model:
        P@1: 14/100 = 0.14
        MMR: 0.17465079365079364
Using the BM25 Model:
        P@1: 20/100 = 0.2
        MMR: 0.25665079365079363
```

```
Using the Jelinek Mercer Model:
        P@1: 26/100 = 0.26
        MMR: 0.30863888888888885
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 11.949 s - in
edu.arizona.cs.TestStem
[INFO] Running edu.arizona.cs.TestNone
Running similarity 1.
Running similarity 2.
Running similarity 3.
Running similarity 4.

For the index at index/none
Using the default Vector Space Model and tf/idf:
        P@1: 21/100 = 0.21
        MMR: 0.25991269841269843
Using the Boolean Model:
        P@1: 8/100 = 0.08
        MMR: 0.14386904761904762
Using the BM25 Model:
        P@1: 21/100 = 0.21
        MMR: 0.25991269841269843
Using the Jelinek Mercer Model:
        P@1: 28/100 = 0.28
        MMR: 0.3372738095238095
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 9.283 s - in
edu.arizona.cs.TestNone
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:07 min
[INFO] Finished at: 2019-05-01T21:42:35-07:00
[INFO] ------------------------------------------------------------------------
```