Course: CSC 483/583
Instructor: Dr. Mihai Surdeanu
Student: Yang Hong
Date: May 3rd, 2017

Watson Project Guide and Error Analysis

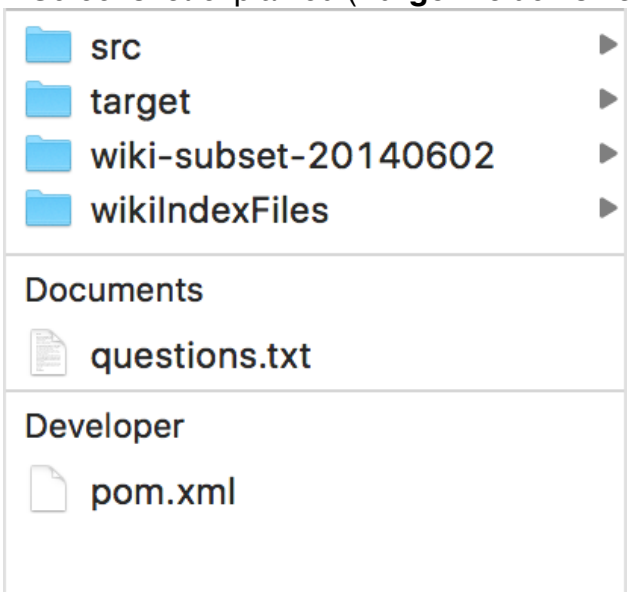
Part One: Read Me

Make sure $JAVA_HOME is pointing to the location where java is installed.
Make sure maven is correctly exported.

OPTIONAL - To set up maven, run following commands:
export JAVA_HOME=$(/usr/libexec/java_home)
export PATH=$JAVA_HOME/jre/bin:$PATH
export PATH=~/ReplaceThisWithMavenFolderLocation/bin:$PATH

Note:
- The program is tested on personal computer with maven 3.3.9 and on lectura with maven 2.2.1.
- This program uses Lucene for language processing (dependencies already added to pom.xml).

IMPORTANT:
- Make sure "**pom.xml**" file and "**src**" folder are in the same directory
- Make sure "**questions.txt**" are in the same directory
- Please include the "**wiki-subset-20140602**" folder in the same directory (Considering the size of the folder, I choose not to upload it with the project, but it is needed)
- Screenshot explained ("**target**" folder is not required, mvn will generate it for you)



Indexing Performance Optimization

- Indexing about 280,000 pages is quite an amount of work, even for a computer
- It originally took quite a few minutes to initialize the program each time
- I'm impatient and don't want to wait
- So idea is that instead of indexing pages each time I run the program, why not just save the indexed documents to hard drive so I don't have to go through the entire indexing process each time
- The solution is simple, use FSDirectory instead of RAMDirectory
- **Important notes**
  - The first run of the program will still take about 3 minutes to index all the pages into documents, but they are saved to hard drive instead of RAM
  - The first run of the program will generate a folder called "**wikiIndexFiles**" with full of indexed documents ready to be used in the future
  - After the first run of the program, you may (want to) disable the indexing function, so future runs become instantly fast, otherwise 3 minutes wait every time, your choice
  - **How To Make It Instantly Fast (Important)**
    - Make sure you have run the program at least one time
    - Then simply comment out the #63 line as following

Comment out line #63 **indexingWikiPages(writer);** in main method:

```
56    public static void main(String[] args) throws IOException, ParseException {
57        StandardAnalyzer analyzer = new StandardAnalyzer();
58        Directory wikiDictionary = FSDirectory.open(Paths.get("./wikiIndexFiles"));
59        IndexWriterConfig config = new IndexWriterConfig(analyzer);
60        IndexWriter writer = new IndexWriter(wikiDictionary, config);
61
62
63        indexingWikiPages(writer);
```

Expected result:

```
56    public static void main(String[] args) throws IOException, ParseException {
57        StandardAnalyzer analyzer = new StandardAnalyzer();
58        Directory wikiDictionary = FSDirectory.open(Paths.get("./wikiIndexFiles"));
59        IndexWriterConfig config = new IndexWriterConfig(analyzer);
60        IndexWriter writer = new IndexWriter(wikiDictionary, config);
61
62
63        // indexingWikiPages(writer);
```

Note: you need to re-compile the program before executing.

And you're all set, enjoy.

Command
-Instruction

mvn compile
-To compile the source code and generate a "target" folder

mvn exec:java
-To run the program

Describe how you prepared the terms for indexing.

I use the default indexing function embedded in Lucene to index wiki pages. Basically, it processes the text by tokenizing, stemming, lemmatizing, and generalizing then stores it into the TextField of each document.

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

1. **Title of each page in specific format [[Title]].** There are lots of varieties among different pages such as categories, histories, timelines. Some pages contain fields that others don't. However, they all have one thing in common, which is the title of each pages is in following format [[Title]]. Thus whenever I encounter such format, I know it is a title for this page. I simply trim off the "[[" and "]]" but did not tokenize the title in any way. I then store the title into StringField of each document.
2. **Inconsistency of the page content.** I discovered that not 2 pages are identical. In fact they may contain different fields. It would be hard to actually store each field into the document considering the varieties of the fields. Thus I decided to store everything except the title into TextField of each document.

3. **Duplicated format [[Title]].** I discovered that there are titles existing even in the body part of each page, so if I just look for the specific format, it might ends up having duplicated titles in document directory. The solution is to read every line each time, if there is a single line which matches the title format and nothing else, then it would become the title of a new document.

Describe how you built the query from the clue.

Version 1.0: I took the entire clue as the query. I first did some language processing on the query before passing it to the search engine, otherwise the result would be useless.

Optimization: For this part, I used 2 directories to make the result more precise. I first did some filtering by returning top 200 relevant documents out of 280,000 set, then I searched the desired result from the smaller directory. The directory which holds the document collection is specified as FSDirectory for performance consideration. The smaller directory is constructed as RAMDirectory since the amount is relatively small and doesn't require that much of a computational power. When filtering, I took the entire clue as my query with language processing done of course. For searching, to make the result even more accurate, I choose to eliminate common terms and stop words such as what, why, how, when, this, that, for, etc.

Are you using the category of the question?

At beginning, I used category of the question as part of my query. I found out that it actually **decreases** the accuracy. I believe the reason is that the category is used to

categorize the questions, not the documents. So it doesn't really make any sense to use a question category as a query for searching documents, so I abandoned it. However, I did use the category field of the documents as one vital factor to determine the relevance of each document when filtering.

Part Two: Measuring Performance

When considering which performance measurement method to choose, I first ruled out the Mean Reciprocal Rank (MMR) because although it is useful when you care about the position of the top correct answers, for the measurement of this project I don't find it very useful since I only want the most relevant answers to appear at higher positions, to be more extreme, I only care about whether the top 1 answer is the correct answer or not.

I was hesitated to choose between Precision at Top 10 (P@10) and Normalized Discounted Cumulative Gain (NDCG). They are both good metric measurements. I ended up choosing NDCG because P@10 requires the measurer to keep track of a lot of different data such as number of documents that are true-positives, false-positives, true-negatives and false-negatives.

I choose NDCG based on following reasons:

1. It is designed to measure the score/ability of an information retrieval algorithm to retrieve non-binary documents that are relevant to the queries.

2. The score has a quite high reference value when determining the ability of an algorithm on web retrieval, which is exactly what the Watson project is about: retrieving correct answers from wikipedia pages.
3. It has an alternative formula for DCG which places stronger emphasis on retrieving relevant documents, and this alternative formula is commonly used in major web search companies and data science competition platforms. I used this formula and I'll show it later.
4. It rewards relevant results that appear higher in the result set, which is what I want since I expect the correct answer appear at 1st.
5. It normalizes the final score. The result of the ranking should be irrelevant of the query preformed, so for instance queries "Michael Jackson" and "Jackson, Michael" shouldn't affect the final score of the algorithm.

NDCG formula break down:

Cumulative Gain (CG) formula

$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

Discounted Cumulative Gain (DCG) formula

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

Discounted Cumulative Gain (DCG) alternative formula (the one I used)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Ideal Discounted Cumulative Gain (IDCG) formula

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Normalized Discounted Cumulative Gain (NDCG) formula

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p},$$

Performance Measurement:

NDCG score is a floating point number between 0 - 1, the higher the score the better the retrieval algorithm. Theoretically there should be an NDCG score for each query, but when measuring the performance of answering 100 questions, there will be 100 NDCG scores which don't really make sense. So I calculated an additional Average Normalized Discounted Cumulative Gain (ANDCG100) which is the average of 100 individual NDCG scores.

Since NDCG score requires human feedback, a way to do it is to prompt user if each returned document is relevant, but the problem is there will be a huge amount of work (100 queries hence 100 * n documents where n is top n documents returned). Therefore I write a piece of code which is able to calculate the scores automatically, the way I solve the relevant/irrelevant problem is that I first check out the first 20 queries with returned documents, I found out that generally if the scoring of the document is in the range of 50% of the top ranked document score, then this document is relevant. So the program will basically assign a int value of 1 to a relevant document, and 0 to irrelevant. The relevant benchmark of scoring is 0.5, the higher the benchmark, the more accurate the result but the lower the NDCG score; the lower the benchmark, the less accurate the result but the higher the NDCG score.

Note:

- I use vector-space scoring function (tf-idf weighting) to retrieve documents;
- Returned document number is set to 10;
- benchmark (relevance baseline) is set to 0.5;
- Average NDCG = each NDCG / 100 (100 queries hence 100 individual NDCG).

The result (part) for each NDCG is as follows:

```
.
Clue: The dominant paper in our nation's capital, it's among the top 10 U.S. papers in circulation
tf-idf Similarity:
The Washington Post                                     0.142408
Watergate complex                                       0.055676
Martin's Tavern                                         0.031816
Harry Thomas, Sr.                                       0.019806
Washingtonpost                                          0.017473
Washington Wave                                         0.014294
Virginia Scholastic Rowing Association                  0.011189
Washington Square (Bellevue, WA)                        0.004915
Ken Harvey (American football)                          0.004690
Joe Mickles                                             0.004213
 DCG=1.0
IDCG=4.543559338088345
NDCG=0.2200917662980802

Answer: The Washington Post
Answer found: true

------------------------------
Clue: The practice of pre-authorizing presidential use of force dates to a 1955 resolution re: this island near mainland China
tf-idf Similarity:
China, Republic of                                      0.003632
Taiwanese geography                                     0.002905
Taiwanese history                                       0.002905
Festivals in Taiwan                                     0.002905
Taiwan/Transnational issues                             0.002905
Holidays in Taiwan                                      0.002905
Roman Catholicism in the Republic of China              0.002905
Taiwan Province (Republic of China)                     0.002905
List of mountains of Taiwan                             0.002905
Passport of Taiwan                                      0.002905
 DCG=4.543559338088345
IDCG=4.543559338088345
NDCG=1.0

Answer: Taiwan
Answer found: false
```

```
Clue: In 2010: As Sherlock Holmes on film
tf-idf Similarity:
Robert Downey, Jr.                                      0.324790
Robert Jr Lockwood                                      0.155940
Hail Caesar (1994 film)                                 0.151546
Institute for Contemporary Studies                      0.136448
Robert e cooper jr                                      0.136448
Morton Downey, Jr.                                      0.066325
Morton James                                            0.028912
Downey, Ca                                              0.028912
Downey, CA                                              0.021684
Downey, ID                                              0.021684
 DCG=1.0
IDCG=4.543559338088345
NDCG=0.2200917662980802

Answer: Robert Downey, Jr.
Answer found: true

------------------------------
Result: 82 out of 100.
------------------------------
Average NDCG: 0.44968374241545683
------------------------------
Search completed.
```

I noticed that although vector-space model works well on returning the correct answer, its Average NDCG score is not as high as I expected. There are following reasons:

- Number of documents returned affects NDCG score, smaller number of documents returned results in higher NDCG score, because more relevant documents / total documents returned;
- Relevance baseline affects NDCG score, smaller benchmark value results in higher NDCG score, because smaller benchmark value assigns more relevant documents;
- The document collection affects NDCG score, for instance, a well-designed IR model is always able to return the desired document out of a collection of 50,000 documents,

but each time there is only 5 document in the collection which is determined relevant, thus the NDCG score is still going to be low.

- An IR algorithm with a low NDCG score doesn't mean it is unable to find the document you want. There are many reasons which could affect the score. It simply means this algorithm lacks the ability of finding a consecutive list of most relevant documents, or the quality of the document collection is bad enough.

Part Three: Changing the Scoring Function

To make the average NDCG score comparable, I didn't change any variables during NDCG score computation except the scoring function.

BM25 NDCG score:

```
Clue: Post-it notes
BM25 Similarity:
3M                                              2.583040
Feudal (game)                                   0.086350
Samuel Smith (chemist)                          0.083251
3m tape                                         0.078361
Minnesota mining and manufacturing              0.078361
3m                                              0.077284
Patsy O'Connell Sherman                         0.074680
Christopher Cross (album)                       0.070429
Kent Ferguson                                   0.068321
Mark Bradshaw (diver)                           0.068321
 DCG=1.0
IDCG=4.543559338088345
NDCG=0.2200917662980802

Answer: 3M
Answer found: true

------------------------------
Clue: In 2010: As Sherlock Holmes on film
BM25 Similarity:
Robert Downey, Jr.                              10.895964
Hail Caesar (1994 film)                         4.590226
Morton Downey, Jr.                              3.455816
Robert Jr Lockwood                              2.057201
Institute for Contemporary Studies              2.055333
Robert e cooper jr                              2.055333
Morton James                                    0.646216
Downey, Ca                                      0.646216
Downey, CA                                      0.644726
Downey, ID                                      0.644726
 DCG=1.0
IDCG=4.543559338088345
NDCG=0.2200917662980802

Answer: Robert Downey, Jr.
Answer found: true

------------------------------
Result: 78 out of 100.
------------------------------
Average NDCG: 0.4747026953525826
------------------------------
Search completed.
```

Language Model with Mercer Smoothing (weighting factor set to 0.5):

```
Clue: Post-it notes
LM Similarity with Mercer Smoothing:
3m tape                                          2.682507
Minnesota mining and manufacturing               2.682507
3M                                               2.081332
3m                                               1.482837
Samuel Smith (chemist)                           0.493792
Feudal (game)                                    0.426638
Kent Ferguson                                    0.348658
Mark Bradshaw (diver)                            0.348658
Christopher Cross (album)                        0.240932
Patsy O'Connell Sherman                          0.235901
 DCG=2.56160631164485
IDCG=4.543559338088345
NDCG=0.5637884576902256

Answer: 3M
Answer found: true

------------------------------
Clue: In 2010: As Sherlock Holmes on film
LM Similarity with Mercer Smoothing:
Robert Downey, Jr.                               6.966279
Institute for Contemporary Studies               6.451178
Robert e cooper jr                               6.451178
Robert Jr Lockwood                               4.332721
Hail Caesar (1994 film)                          3.131760
Morton James                                     2.313990
Downey, Ca                                       2.313990
Downey, CA                                       1.812709
Downey, ID                                       1.812709
Morton Downey, Jr.                               1.634883
 DCG=2.56160631164485
IDCG=4.543559338088345
NDCG=0.5637884576902256

Answer: Robert Downey, Jr.
Answer found: true

------------------------------
Result: 77 out of 100.
------------------------------
Average NDCG: 0.6025132290849726
------------------------------
Search completed.
```

Language Model with Dirichlet Smoothing (weighting factor set to 0.5):

```
Clue: Post-it notes
LM Similarity with Dirichlet Smoothing:
3m tape                                                 2.447493
Minnesota mining and manufacturing                      2.447493
3m                                                      1.191933
3M                                                      0.288201
Samuel Smith (chemist)                                  0.000000
Feudal (game)                                           0.000000
Kent Ferguson                                           0.000000
Mark Bradshaw (diver)                                   0.000000
Patsy O'Connell Sherman                                 0.000000
Christopher Cross (album)                               0.000000
 DCG=1.6309297535714573
IDCG=4.543559338088345
NDCG=0.3589542101716347

Answer: 3M
Answer found: true

------------------------------
Clue: In 2010: As Sherlock Holmes on film
LM Similarity with Dirichlet Smoothing:
Robert Jr Lockwood                                      6.675280
Institute for Contemporary Studies                      6.193927
Robert e cooper jr                                      6.193927
Morton James                                            2.105727
Downey, Ca                                              2.105727
Downey, CA                                              1.580195
Downey, ID                                              1.580195
Hail Caesar (1994 film)                                 1.341659
Robert Downey, Jr.                                      0.586326
Morton Downey, Jr.                                      0.000000
 DCG=2.1309297535714573
IDCG=4.543559338088345
NDCG=0.4690000933206748

Answer: Robert Downey, Jr.
Answer found: true

------------------------------
Result: 69 out of 100.
------------------------------
Average NDCG: 0.4607517413410253
------------------------------
Search completed.
```

In this particular document collection with 100 specific queries, I found out that language model with Mercer smoothing (weighting factor 0.5) acquires the highest score of 0.60, the 2nd is BM25 with a score of 0.47, the 3rd is LM with Dirichlet smoothing (weighting factor 0.5) with a score of 0.46, the last place is vector-space model (tf-idf weighting) which earns a score of 0.45.

This is quite an unexpected result since I assumed that vector-space model would get the highest score, but again, the NDCG score is not a definite value to determine the performance of a scoring function, because a NDCG score is affected by many factors such as relevance baseline, number of returned documents and the quality of the document collection.

As a result, vector-space model performs the best by returning the most correct answers (82 out of 100), but acquires the smallest average NDCG score (0.45); language model with Mercer smoothing performs the worst on returning correct answers (69 out of 100) but acquires the greatest average NDCG score (0.60).

Part Four: Error Analysis

How many questions were answered correctly / incorrectly?

Scoring Function	Accuracy
Vector-Space Model (tf-idf weighting)	82 / 100
BM25	78 / 100
LM with Mercer smoothing	77 / 100
LM with Dirichlet smoothing	69 / 100
Multiple Similarity	91 / 100

Why do you think the correct questions can be answered by such a simple system?

In simple terms, the magic of information retrieval.

Most of the questions are in fact detailed descriptions of the fact of the correct answer, in other words, the question itself has already given out the answer. All we did is just to find the answer that is linked to that question, thus information retrieval.

If there are logics such as mathematical calculations contained in those questions, I doubt a simple system like this is capable of giving the right answers, for example:

```
Please enter a query:
what is the result of two to the power of four?

Multi-Similarity:
Powers of two                                    41.302841
Power gain                                       39.128113
Social balance theory                            31.367363
Social conflict theory                           30.560818
Everway                                          30.327557
```

I simply want to know the result of 2 to the power of 4, the search engine gives me a bunch of documents instead of the correct answer 16.

In conclusion, this type of search engine is only able to answer the questions if and only if the answer is present in its database, otherwise the returned results are useless.

What problems do you observe for the questions answered incorrectly? Try to group the errors into a few classes and discuss them.

The weighting of each term affects the result. In my program, I did conduct language processing on query and get rid of stop words / common terms, but I didn't differentiate terms, meaning there is no weighting for each term, so essentially all the terms left are equally valuable, but it should not be. Take this for an example:

```
Clue: Don Knotts took over from Norman Fell as the resident landlord on this sitcom
BM25 Similarity:
Don Knotts

tf-idf Similarity:
Don Knotts

LM Similarity with Mercer Smoothing:
Don Knotts

LM Similarity with Dirichlet Smoothing:
Don Knotts

Multi-Similarity:
Don Knotts

Answer: Three's Company
Answer found: false
```

All five scoring functions return Don Knotts instead of Three's Company, therefore the correct answer is not found. If we look at this more closely, we discover that in general, a person's name does have the greatest weighting among other terms, which means Lucene actually did a very good job on weighting terms. However, in this case, the word "sitcom" should have the greatest weighting over the others. As I discussed earlier, a simple information retrieval algorithm is not capable of answering logical questions. This question actually has some logic behind it which this search engine is not able to comprehend, thus causing it to simply return the highest scoring document which is the profile of the person named Don Knotts.

The quality of the document collection affects the result. As I discussed earlier, a simple information retrieval algorithm is heavily depended on a document collection with a decent quality. My search engine is able to achieve a accuracy of 91 out 100 not because the algorithm is super smart, but because the document collection contains all the correct answers. The credits belong to Wikipedia. Imagine a document collection with full of mathematic equations, the same set of 100 questions is passed into the search engine, it will eventually return the highest scoring document as the result, but it will never be the correct answer.

The valuable information contained in each query affects the result. The effectiveness of a simple information retrieval algorithm is somewhat depended on the quality of the query as well. I first try to search for the correct answer by just entering a few crucial terms, it doesn't work.

```
Please enter a query:
singer american beat it

Multi-Similarity:
Mario's fifth studio album                    53.725128
Rudolph Mangual                               47.155655
Zig Zag (2002 film)                           43.115540
Michelle Lewis                                41.784599
Skipping a Beat                               40.816116
```

The problem is that although we as people immediately know the answer for the question, the program can't do that. The program simply searches documents where each term is present, then return the highest scoring document, but what we are expecting is some logic process behind the query that a more advanced search engine like Google is capable of.

Part Five: IR Model Optimization (Grad Student Only)

Narrowing document collection

Since I only care about the correctness of the answer, in other words, the 1st document returned, I believe the result would be more accurate if I filter the document collection before actually searching for answer. For example, there is a greater chance of finding the exact document from 280 document collection than from 280,000 document collection.

The idea is simple, I pass the clue as the query into my search engine and get a set of n documents where n is significantly smaller than 280,000. Then I create a new RAMDirectory storing these documents as my new document collection, after that process the clue to get rid of commonly used terms such as be, why, how, who. Then pass the left rare terms as query into the search engine and get the 1st result as the answer. It did perform better by returning more correct answers.

Using multiple scoring functions

As I discovered earlier, using only one scoring function may or may not do well on giving correct answers, so why not combine them together?

I embedded all four scoring functions I've used so far (vector-space model, BM25, LM with Mercer smoothing, LM with Dirichlet smoothing) into a new scoring function called Multiple Scoring, turns out it performs the best by returning the most correct answers and acquiring the highest average NDCG score.

Multiple Scoring function performance Note:

- Scoring functions embedded:
  - vector-space model (tf-idf weighting)
  - BM25
  - LM with Mercer Smoothing (weighting factor set to 0.5)
  - LM with Dirichlet Smoothing (weighting factor set to 0.5)
- Returned document number is set to 5 (Detailed Mode);
- Returned document number is set to 1 (Simple Mode);
- benchmark (relevance baseline) is set to 0.5;
- Average NDCG = each NDCG / 100 (100 queries hence 100 individual NDCG);
- Document collection (filter directory) size set to 200.

Multiple Scoring function average NDCG score result:

(Simple Mode)

A NDCG score example:

```
Clue: Post-it notes
BM25 Similarity:
3M
 DCG=1.0
IDCG=1.0
NDCG=1.0

tf-idf Similarity:
3M
 DCG=1.0
IDCG=1.0
NDCG=1.0

LM Similarity with Mercer Smoothing:
3m tape
 DCG=1.0
IDCG=1.0
NDCG=1.0

LM Similarity with Dirichlet Smoothing:
3m tape
 DCG=1.0
IDCG=1.0
NDCG=1.0

Multi-Similarity:
3m tape
 DCG=1.0
IDCG=1.0
NDCG=1.0

Answer: 3M
Answer found: true
```

Final result:

```
------------------------------
Result: 91 out of 100.
------------------------------
Average NDCG: 0.97
------------------------------
Search completed.
```

(Detailed Mode)

A NDCG score example:

```
Clue: Post-it notes
BM25 Similarity:
3M                                          2.158955
Samuel Smith (chemist)                      0.159146
3m tape                                     0.146706
Minnesota mining and manufacturing          0.146706
3m                                          0.145098
 DCG=1.2973967099940698
IDCG=2.948459118879392
NDCG=0.440025334482903

tf-idf Similarity:
3M                                          0.145698
3m tape                                     0.066521
Minnesota mining and manufacturing          0.066521
3m                                          0.033261
Samuel Smith (chemist)                      0.014402
 DCG=1.2973967099940698
IDCG=2.948459118879392
NDCG=0.440025334482903

LM Similarity with Mercer Smoothing:
3m tape                                     2.431647
Minnesota mining and manufacturing          2.431647
3M                                          1.551476
3m                                          1.279377
Samuel Smith (chemist)                      0.396391
 DCG=2.56160631164485
IDCG=2.948459118879392
NDCG=0.8687949224876582

LM Similarity with Dirichlet Smoothing:
3m tape                                     2.179888
Minnesota mining and manufacturing          2.179888
3m                                          0.924328
3M                                          0.000000
Samuel Smith (chemist)                      0.000000
 DCG=2.1159628965150477
IDCG=2.948459118879392
NDCG=0.717650410333399

Multi-Similarity:
3m tape                                     5.383241
Minnesota mining and manufacturing          5.383241
3M                                          4.166733
3m                                          2.661303
Samuel Smith (chemist)                      0.690853
 DCG=2.7646612515120825
IDCG=2.948459118879392
NDCG=0.9376630775748505

Answer: 3M
Answer found: true
```

Final Result:

```
------------------------------
Result: 91 out of 100.
------------------------------
Average NDCG: 0.8030649838730138
------------------------------
Search completed.
```

Note that the result (91 out of 100) remains unchanged because the top 1 result remains unchanged, however, the average NDCG score is changed because the number of returned document is changed.

End of Report