**CS6200 Information Retrieval Project 02**

Jiachi Liu

# Problem

Build an information retrieval system in the following five models:

1. Vector Space Model use OKAPI TF as weight.
2. Vector Space Model use OKAPI TF with IDF as weight.
3. Language Model with Laplace smoothing.
4. Language Model with Jelinek-Mercer smoothing
5. BM25

# Resources

The following file contains basic information the system needs to form the ranking result:

1. doclist.txt: a file that mapping the interval representation of documents with external representation.
2. Stem-classes.lst.txt: stemming files that has information about terms and its stemming.
3. Stoplist.txt: a list of stop terms.

# Input and Output

The system is running by the following command:

*Python run.py [model id] [database id] [output]*

To run model 0 on database 3, just input: *Python run.py 0 1 model0-d3.txt*

|  |  |
| --- | --- |
| Model id | Model |
| 1 | OKTF |
| 2 | OKTF\_IDF |
| 3 | Max likelihood with Laplace |
| 4 | Max likelihood with Jelinek-Mercer |
| 5 | BM25 |

Table 1 Mapping between model\_id and model

A query file is provided which contains 25 queries. For the given model, the system will run all the 25 queries on it and output the top 1000 ranking documents for each query to the given output file.

# Evaluation

Using “qrel.irclass10X1” and “qrels.adhoc.51-100.AP89” to evaluate the results to see average precision of each model.

# Implementation

## System Overview

The system contains several classes:

1. Resouces: contains the information about the database, document mapping, stemming terms, stop terms and queries.
2. InvertList: A bean class contains information got from Lemur web interface. Each InvertList mapping to a term.
3. InvertListParser: send request to the web interface and parse the response content to InvertList.
4. Query: Saving the terms in each query. Provided a filtering list that can be used to remove terms from query. Also provide basic statistic function to get information about a query.
5. QueryParser: Parse the given query file to Query, remove the stop words and replace words to stemming terms.
6. Application: System context class which serves as a controller to manage the initialization of models and work flows. This class will dynamically create model based on input model id.
7. OKTF, OKTFIDF, MaxLikelihood, JelinekMercer, BM25: Five models. Each of them implement an interface contains “setQuery” method and “rank” method.
8. Run.py: a script to run the system.

## Ranking

Basic steps of the system:

1. Load the resources files to memory.
2. Preprocessing the queries.
3. Ranking the queries on certain model selected by the input argument.
4. Evaluating the result output.

### Preprocessing the Query

For each query, the QueryParser will do the following:

1. Transform all character to lowercase.
2. Remove the punctuation from query such as ‘,’, ’-’, ‘ ’ ’. For ‘-’, the original word will be separated into two new words.
3. Remove the stop words.
4. Convert each term to its stemming term.
5. Remove the words that has less meaning in query like ‘documents’, ‘discuss’, ‘include’, ‘identify’, ’cite’, ‘describe’. Those words are in the filtering list which can be modified easily for experimental purpose.

### Ranking the queries

The following steps will be executed when ranking queries:

1. Initialize the ranking model based on the input model id and database id.
2. For each query in queries:
   1. Set it as the current query of model
   2. Rank the query.
   3. For the output of rank, sort it in decreasing order and get the top 1000 results.
   4. Format the results as required in problem description
   5. Add the formatted result to final output
3. Write the final output to file.

### Ranking the query

For each term in query:

1. Generate the query url.
2. Get the invert list from web interface and parse it into InvertList class.
3. Calculate the require information based on model(like query length,idf).
4. For each document in the invert list:
   1. Calculate the ranking score based on model.
   2. Saving the score in a map where key is docid and value is the score.
   3. If the doc is already in the map, add the score to the existed value. Otherwise, put the docid with the score in the map.
5. Return the result map.

### Parameters for the models

1. Space Vector Model: the avg\_query\_len is calculated by adding all the terms in preprocessed query and divided it with 25.
2. Jelinek-Mercer: λ = 0.2
3. BM25: K1 = 1.2, K2 = 100, B=0.75

## Evaluation and Analysis

### Result of DB 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DB = 3 (STOP,STEM) | | | | | | | | | |
| ID | Model Name | MAP | | R-Precision | | Precision at 10 | | Precision at 30 | |
| NIST | IRClass | NIST | IRClass | NIST | IRClass | NIST | IRClass |
| 1 | OKAPI TF | 0.1863 | 0.2538 | 0.2208 | 0.2622 | 0.3240 | 0.2800 | 0.2680 | 0.2040 |
| 2 | OKAPI TF \* IDF | 0.2570 | 0.3150 | 0.2813 | 0.3159 | 0.3880 | 0.3400 | 0.3187 | 0.2427 |
| 3 | Laplace | 0.1848 | 0.2501 | 0.2316 | 0.2358 | 0.3608 | 0.3200 | 0.2880 | 0.2493 |
| 4 | Jelinek-Mercer | 0.2181 | 0.2483 | 0.2472 | 0.2621 | 0.3200 | 0.2608 | 0.2707 | 0.1920 |
| 5 | BM25 | 0.2503 | 0.3035 | 0.2770 | 0.3025 | 0.3560 | 0.3080 | 0.3120 | 0.2333 |

Table 2 DB3 Results Table

figure 1 Recall-Precision (NIST) on DB3

figure 2 Recall-Precision (IRClass) on DB3

As the above table and figures shown, the OKAPITF\*IDF(model 2) has the highest mean average precision. BM25 has the second highest precision followed by Jelinek-Mercer.

Comparing the result between NIST and IRClass, the relative positions of these models are remaining same whereas IRClass get higher precision. It is probably because that IRClass is using similar model as us while NIST using a different way to rank documents.

The recall-precision plot shows that although BM25 has a top precision at the beginning and the end, but the average precision in the middle is lower than OKAPITF\*IDF. Therefore, the MAP of BM25 is lower than OKAPITF\*IDF. If we want to have more relevant documents, OKAPITF\*IDF should be chosen. But if we want to have more relevant documents at beginning of query, we need BM25.

Comparing the Language Model, although Laplace has a very high recall at beginning, it also drops fast with the processing. On the other hand, Jelinek-Mercer’s performance is relatively stable and at the end got a much better result than Laplace.

### Results of DB 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DB = 2 (STOP,NO STEM) | | | | | | | | | |
| ID | Model Name | MAP | | R-Precision | | Precision at 10 | | Precision at 30 | |
| NIST | IRClass | NIST | IRClass | NIST | IRClass | NIST | IRClass |
| 1 | OKAPI TF | 0.1568 | 0.1956 | 0.1806 | 0.2181 | 0.2840 | 0.2080 | 0.2147 | 0.1560 |
| 2 | OKAPI TF \* IDF | 0.1902 | 0.2273 | 0.2157 | 0.2330 | 0.3160 | 0.2520 | 0.2493 | 0.1730 |
| 3 | Laplace | 0.1486 | 0.1862 | 0.1855 | 0.1945 | 0.3240 | 0.2720 | 0.2547 | 0.1947 |
| 4 | Jelinek-Mercer | 0.1490 | 0.1695 | 0.1830 | 0.1739 | 0.2600 | 0.1880 | 0.1973 | 0.1187 |
| 5 | BM25 | 0.1889 | 0.2207 | 0.2109 | 0.2182 | 0.3000 | 0.2240 | 0.2480 | 0.1587 |

Table 3 Results for DB 2

figure 3 Recall-Precision (NIST) on DB2

figure 4 Recall-Precision (IRClass) on DB2

As the table and graph shows, the mean average precision dropped significantly comparing with DB3. The problem could be the stemming preprocessing of query. Since I manually