Programming Assignment - Part 2: Instructions

Help Center

Submit Now

Programming Assignment 2: Search Engine Competition

This assignment is a search engine competition where you will freely experiment with different retrieval methods to improve your score on the leaderboard. You will not receive a numerical grade on the competition, but the Programming Competition Leader Badge will mention your rank on the leaderboard. Throughout the competition, if you want to know more about a certain class or function, you can use the search toolbar in MeTA's Documentation, which provides a brief explanation of the different modules. You might also find some of MeTA's tutorials useful when experimenting with the analyzer and writing new functions. If you have questions about the programming assignment, use the **Programming Assignments Forum**. This is a great place to ask questions and also help your fellow classmates.

Downloading the Assignment

Before downloading this assignment, make sure you have installed MeTA and run the **Setup.sh** script as detailed in Programming Assignment -- Part 1. In this assignment's instructions, we assume that MeTA is installed in **~/Desktop/**. If you installed it in a different directory, you will have to change **~/Desktop/** in the commands to the parent directory where you placed the folder **meta**.

1. Download Assignment_2.tar.gz and extract it into the parent directory of meta. If meta is in ~/Desktop/, then you should extract the assignment in ~/Desktop/

2. In the terminal, change the directory to **Assignment_2** and run the bash script **Setup2.sh**. This can be done using the following two commands:

```
cd ~/Desktop/Assignment_2/
./Setup2.sh
```

3. Recompile MeTA:

```
cd ~/Desktop/meta/build/
make
```

Relevance Judgements (20 pts added to Assignment 1)

In the previous assignment, you came up with your own queries and identified the corresponding relevant documents, which led to the creation of a large pool of queries and relevance judgments. These judgements will be used to compare the effectiveness of the search engines in the competition. However, assessors might not always agree on the same set of relevant documents for a given query.

In order to make the judgements more reliable, you will be asked to identify the relevant documents for a randomly selected query written by another student. At the end of the competition, we will combine the relevance judgments collected from both assignments to calculate the final leaderboard rankings.

In the terminal, execute:

```
cd ~/Desktop/meta/build/
./new-judgements config.toml
```

The program will first ask you whether you agree to *anonymously* share your relevance judgments with researchers at the University of Illinois at Urbana-Champaign. The automated grader will *not* keep any information related to your identity if you are willing to make your relevance judgments available to researchers. Answer by "yes" or "no" without quotations. Then, the program will randomly select a query and will return the top 20 corresponding documents. You should enter the numbers of the relevant documents separated by spaces. If you feel that you did not understand the query, run the program again to get another query. Some of the students' queries contain typos. If the query you are asked to judge contains a typo, you should complete the relevance judgments as long as you can understand the query.

Finally, submit the output using the submission script:

```
cd ~/Desktop/meta/build/Assignment2/
python submit.py
```

If you encounter any problems with the submission script, upload the file **new-judgements.txt**, which is located in **meta/build/Assignment2/**, to the assignment's submission page.

Competition

The competition involves optimizing your own search engine to search the MOOC's dataset. We have provided you with 100 training queries accompanied by relevance judgments to quantify the effectiveness of your search engine. After optimizing your search engine based on the training queries, it will be evaluated by the automated grader based on another set of 538 testing queries, which you do not have relevance judgments for. Your rank on the leaderboard will be based on the Mean Average Precision achieved by your search engine *on the testing queries*. If your search engine is robust enough, you should expect the MAP value you get on the training queries to be close to the MAP on the testing queries. You can have a look at the training and testing queries in **meta/data/moocs/moocs-queries.txt**, but do not modify the file.

We have created a program for this competition called **competition.cpp** located in **meta/src/index/tools/**. Open **competition.cpp**, read the code within the **main** function along with the comments, and try to understand what is being performed. You should focus on the two while loops within **main**. The first loop passes over the 100 training queries and prints the precision at 10 documents and the MAP. The second while loop passes over the 538 testing queries and writes the IDs of the top 50 documents corresponding to each query to the output file **output.txt**, which is located in **meta/build/Assignment2/**.

You are free to try or implement any concept that you feel can lead to better retrieval in the MOOC's dataset, even if it was not discussed in class. We provide some pointers below that will help you in achieving higher retrieval performance. The main techniques are programming-based and require writing new functions, but we also provide pointers to some techniques that do not require programming. You are highly encouraged to experiment with the programming-based techniques first.

Programming-Based:

1. Implement pseudo feedback through Rocchio's method. One simple way to do this is to write a new function in **competition.cpp** that implements Rocchio feedback. The function should take a query and a set of positive feedback documents as arguments and return a modified query. Call the Rocchio function for each query and its top 10 initially retrieved documents. Then, call the scoring function again on the modified query that was output by Rocchio. This can be done as follows:
   Replace:

   ```
   auto ranking = ranker->score(*idx, query, 50);
   ```

   with:

   ```
   auto ranking = ranker->score(*idx, query, 10);
   auto new_query = Rocchio(query, ranking); // You should implement this function
   ranking = ranker->score(*idx, new_query, 50);
   ```

   You should perform pseudo feedback on both the training and testing queries, which means that you should replace "auto ranking = ranker->score(*idx, query, 50);" in the two while loops. Also, note that this is only a suggestion. You can implement the feedback in other ways and using a different number of positive feedback documents.

2. Write a tuning function, similar to the one you saw in Assignment 1, to optimize your ranking function over a suitable set of parameters.
3. Write a new scoring function, similar to the PL2 that you implemented in Assignment 1, that returns a weighted combination of the scores of different retrieval formulas. For example, you can implement something like:

   $\text{Score}(Q,D) = \alpha\,\text{BM25}(Q,D) + (1-\alpha)\text{DirichletPrior}(Q,D)$ where $0 < \alpha < 1$

   We have already provided you with a scaffold where you can implement your new function. At the top of **competition.cpp** you can find a commented class called **new_ranker** along with some functions. Uncomment the code and modify **score_one** to return the weighted combination you choose. The code assumes that the ranking function has two parameters named "param1" and "param2" and that the ranker can be called from **config.toml** using the name "newranker.". If your ranking function requires a different number of parameters, you should modify the code accordingly. Also, feel free to change the names of the variables and parameters; the provided code is there just to guide you through writing your new function. After implementing the function, uncomment the first line in **main**:

   ```
   index::register_ranker<new_ranker>();
   ```

   Don't forget to point **config.toml** to your new ranker. For more information on how to implement your new function, see the last section in MeTA's Search Tutorial and inspect the code of **ranking-experiment.cpp** from Assignment 1.

   If you want to further improve your ranking functions, you might want to have a look at section 6 of this paper, which introduces several empirically-driven enhancements to some of the well-known retrieval functions.

4. Write a function that expands queries with synonyms. Given a query, your function will use a thesaurus to augment the query words with their synonyms. It is a good idea to give the original query words more weight since the synonyms might cause a topic drift in some cases. One crude way to do this is to duplicate the original query words several times and have each synonym appear only once in the modified query. In **competition.cpp**, you should replace:

   ```
   query.content(content);
   ```

   with:

   ```
   query.content(content);
   query = expand_query(query);  // You should write this function
   ```

   Make sure to expand the queries in both while loops.

Non-Programming Based:

1. Try different ranking functions and different parameters (ideally, you should tune the parameters). MeTA has several built-in rankers other than BM25. For example, you can use the Jelinek-Mercer smoothing ranking function by setting the ranker in **config.toml** to **jelinek-mercer** and tuning its parameter **lambda**. You can check the different built-in rankers in **meta/src/index/ranker/**. When you open the cpp file of the ranker, you should find a constant string called **id** whose value you can use in **config.toml**, in addition to the parameters of the ranking function which are defined in the constructor (see **okapi_bm25.cpp** and **jelinek_mercer.cpp** to gain more insight).
2. Index the MOOCs dataset again while experimenting with different tokenization parameters under the analyzers tag in **config.toml**. You can try different combinations of text filters and select the one that gives the best performance on the training queries. See MeTA's Analyzers and Filters Tutorial for instructions on how to modify the default filtering behavior.
3. Modify MeTA's default stopwords list, which is located in **meta/data/lemur-stopwords.txt**. You can extend or shrink the list and check if you can achieve higher performance. After modifying the list, make sure to index the dataset again.

After you perform the optimizations, compile MeTA again and run the **competition** program. You can do so by executing:

```
cd  ~/Desktop/meta/build/Assignment2/
make
./competition config.toml
```

The program will first ask you for the nickname that you want to use if your submission appears on the leaderboard. If you make multiple submissions, the nickname you choose in the last submission will be the one that appears on the leaderboard. The results of the training queries, the precision at 10 documents, and the MAP will be printed. When you are satisfied with your results, submit the output file:

```
cd  ~/Desktop/meta/build/Assignment2/
python submit.py
```

If you are having trouble with the submission script, upload **output.txt** through the assignment's submission page.

Note: Always check the feedback from the automated grader on the submissions page after you submit your output. If the nickname you chose is already in use, the grader will ask you to submit your output again using a different nickname. You can access the leaderboard from here. If you scored in the top 30% of the class, the nickname you chose will appear on the leaderboard along with the MAP value. In case of ties, the student who submitted earlier will have the higher position on the leaderboard. If you score below the top 30%, you can privately check your ranking through the submissions page by clicking on **View** in the feedback column.

Description of your Methods (Optional)

If you are willing to share the methods you explored in the competition, you can write a small text document that does not exceed 100KB. You can describe the different techniques you tried, even if they did not lead to an improvement. Retrieval is a very empirically-driven task, and sometimes good methods might not lead to the aspired results. The document can also include a link to your personal page if you want to elaborate more on what you did. We might publicly share the methods implemented by the top students on the leaderboard after the competition ends. Name your file "description.txt" and place it in **meta/build/Assignment2/**. Use the submission script or the submissions page to upload it.

Submit Now