372-1-4406 -אחזור מידע
Information Retrieval

סמסטר חורף, 2025-2026

Final project

Building a search engine for English Wikipedia

The project is optional, but submitting a project that meets the minimum requirements (listed below) will earn you 10 points in the final grade automatically. Moreover, the project numeric grade (56-100) will protect your exam grade (a.k.a. Magen) with a weight of 15% such that: Exam grade = max(Exam, 0.85*Exam + 0.15*Project).

Data

- Entire Wikipedia dump in a shared Google Storage bucket (same as Assignment #3).
- Pageviews for articles (you need to derive this; see code in Assignment #1 for details).
- *queries_train.json:* Queries and a ranked list of up to 100 relevant results for them. This contains the train split (30 queries+ ranked result list for each), but not the test split, which we will use for evaluation.

Code

The staff provides you with the following pieces of code (available on Moodle):

- *search_frontend.py*: Flask app for search engine frontend. It has six blank methods that you need to implement.
- *run_frontend_in_colab.ipynb*: notebook showing how to run your search engine's frontend in Colab for development purposes. The notebook also provides instructions for querying/testing the engine.
- *run_frontend_in_gcp.sh*: command-line instructions for deploying your search engine to GCP. You need to execute these commands to start a Compute Engine instance (machine), reserve a public IP, and run your engine in GCP.
- *startup_script_gcp.sh*: a shell script that sets up the Compute Engine instance. No need to modify this unless you need additional packages installed in GCP.
- *inverted_index_gcp.py*: code for reading and writing an index to GCP storage bucket.

Minimum Requirements

To get a passing grade (56) you must meet ALL of the following requirements:

1. **Functional search engine**: a search engine that can process queries and return results from the entire corpus.
2. **Testable search engine:** your search engine must be accessible through a URL you provide to us upon submission and able to answer queries during a two-day testing period, from **Tuesday, January 13, 2025 at 12:00 (noon) to Thursday, January 15, 2025 at 12:00 (noon)**. Alternative testing period may be provided for students who cannot meet this deadline due to university-approved rights and regulations (hospitalization, reserve duty, etc.).
3. **Efficiency requirement**: No query takes longer than 35 seconds to process. Caching of results is not allowed.
4. **Quality requirement**: Average Precision@10 > 0.1 on the test set.
5. **No use of external services**: Your search engine may use additional packages and models (statically linked), but it cannot use other services or dynamically call an API at query time. For example, you can use a fancy embedding model or package to create a local vector database, but you are not allowed to have your database reside on an external server or hit an external service at query time. The same applies to all external services, even if they "just" provide word synonyms.
6. **Clean and organized code repo:** a repository on Github or other publicly available source control platform with a Readme.md file that explains the code structure, organization, and functionality of each major piece in your implementation.
7. **Reporting requirements**: A report in Hebrew or English (submitted through Moodle) of up to 4 pages (you don't have to fill up the space) using font size 12 and line spacing 1.5 that contains:
    a. Student IDs and emails.
    b. A link to a Github repo with the code.
    c. A link to a Google Storage Bucket, where all indexing data you calculated resides and is publicly accessible. Instructions for making your bucket public are here.
    d. List all index files with human-readable sizes, pasted in an appendix at the end of your report. This does not count towards the page limit. You can use the command `*gsutil du -ch gs://BUCKET_NAME*` to generate this listing or `*du -ch LOCAL_DIR*` if you store index data locally.
    e. A description of key experiments you ran, how you evaluated them, and the key findings/takeaways.
    f. A graph showing the engine performance for each major "version" of your implementation.
    g. A graph showing the engine's average retrieval time for each major "version" of your implementation.
    h. Qualitative evaluation of the top 10 results for one query where your engine performed really well and one query where your engine did not perform well. Describe what worked well and what didn't work so well. What is the dominant factor behind the poor result? What can be done about it?

Full requirements

In addition to the above minimum requirements, to get the full grade you need to:

1. **Support five ranking methods (10 points)**: (a) cosine similarity using tf-idf on the body of articles, (b) binary ranking using the title of articles, (c) binary ranking using the anchor text, (d) ranking by PageRank, and (e) ranking by article page views. See search_frontend.py for implementation details. A method that takes longer than 35 seconds to return result will not get you any points.
2. **Efficiency (7 points)**. Evaluated based on the average retrieval time of the 'search' method as follows: <1 second (7 points), 1-1.5 seconds (6 points), 1.5-2 seconds (5 points), 2-2.5 seconds (4 points), 2.5-3 seconds (3 points), 3-3.5 seconds (2 points), 3.5-5 seconds (1 point), and >5 seconds will get no points.
3. **Results quality (18 Points)**: Measured using the average across test queries of the harmonic mean of Precision@5 and F1@30, relative to other submissions.
4. **Experimentation and evaluation (15 points)**. Which additional experiments and/or retrieval models were tried and how thoroughly were they evaluated?
5. **Reporting (4 Points)**. Clean code & clear explanations (2 points). Submit a short presentation (3-5 slides) summarizing your work (2 points).

Words of advice

1. Go over the slides of lecture #3 to remind yourself of the ingredients needed for efficient cosine similarity calculation.
2. Remind yourself what goes into the tf-idf calculation.
3. Read Chapter 7 of the class's textbook. The techniques there can be very helpful for efficient retrieval, especially when queries take too long.
4. Consider leaving aside some of the training set queries for your own testing.
5. Use the training set to derive a small inverted index that consists only of words that appear in the training set queries. Use this index to optimize ranking and retrieval efficiency. Only later on move on to building the full index for the entire corpus.
6. Your best search engine (implemented under the 'search' method) can use whatever method or technique we learned in class including stemming, stopword removal, word embedding, query expansion, and more. Meet with your partner and plan what you'll implement/try out during the work on the project.
7. Start small, using Colab, and build up from there. Only add one thing at a time and test it.
8. Spend time *early* to make sure that you can run the frontend without any modifications on GCP (following the instructions in run_frontend_in_gcp.sh), and make sure that you can query it using the public IP it generated.
9. Leave enough time (2-3 days) to optimize the engine on GCP.
10. Block off regular time between now the submission date when you and your partner can meet to work together on the project, ideally while both of you are physically located in the same space.

Good luck!
Your IR Staff