Final Project, Deliverable 2

Submission. For your submission, each group will upload their presentation, a link to their GitHub repository containing their Python project (as a comment in Canvas), and any other supporting materials if those were not already represented in the project itself. I should be able to reproduce all the results that you describe in your presentation.

The culminating activity in this course is the final project. In groups of about three to four students, you will design a project that establishes a research question and executes some type of experimental procedure around it. This work involves two key components: a software component and a presentation component.

The **software component** will be a GitHub repository containing the Python package your team has created for your project. Your software should follow Python best practices, leveraging the ideas we've been developing throughout the semester. In particular, your software should at least:

- follow modern project structuring described in our Python Packaging lab—that is, use the `pyproject.toml` format
- use object-oriented programming when appropriate (remember, part of expertise is knowing when to apply what you know and when not to)
- feature a carefully-designed testing suite that tests the code you've written **using `pytest`**
- **support logging using the `logging` module at sensible places**
- allow the end-user to access and use the package with an easy-to-understand API (in other words, it should be simple for me to reproduce your experiments)
- discuss the high-level aim of the package in the README, including a demonstration of how to use key features and a graphic of a diagram illustrating how the pieces of your package connect to one another. In addition, you must use draw.io or some other diagram software to create an image showing your program architecture. The point is, someone who has never looked at your code should be able to easily ground themselves in the high-level structure of what you've done by reading your documentation. **In the README, put the full names of each team member.**
- include all other project-structuring code, such as an `environment.yml` file (which will likely differ from our class's configuration if you use additional libraries)
- **Data should not be pushed to GitHub. Instead, you should use GitHub LFS for large data files, or provide a link to the cloud (e.g., Google Drive), where the data can be downloaded using your code. Then, you can preprocess the data, or store preprocessed data in the cloud instead (e.g., if preprocessing takes a long time).**
- **Your team is expected to collaborate on GitHub. One team member should create a Git repository for the project, and the other team members should pull remote versions of that repository for their own local copies. Each member is expected to make contributions, which will be reflected in the commit history of the repository. Gaining some comfort with this process is important for what real team-based software development looks like.**
- every function, method, and class should have a docstring
- use a linting software like `pylint` or an opinionated, PEP 8 formatter like black or ruff to clean your code. It is possible to have such software run every time you save changes; for example, you can install a formatter or linter as a VS Code plugin.
- Use GitHub Actions to run standard CI steps
- use issue templates and track progress through a robust Git collaborative workflow, as described in Assignment 2
- follow a microservices design pattern, where individual components of your work are separate Docker containers, connected via Docker Compose. As described in Assignment 3, this means making good design decisions about the software you'll use for a user-facing API, backend database logic, analytics, and anything else relevant to the functioning of your application
- As you did in Assignment 2, discuss runtime, memory usage (including saying how you measure memory usage), conduct an informal error analysis (i.e., look at a few of the predicted document clusters, trace back from the document IDs to the raw documents, and discuss whether you see any mistakes), **plus anything else you think is important (remember: you're performing scientific experiments and you're asking questions)**.
- Include all of your analysis in a PDF in your repository called `discussion.pdf`. The `discussion.pdf` should be no longer than 4 pages, excluding references, and it will include your comments, graphics, and overall analysis. Your general structure might be something like:
  - exploratory data analysis over your dataset (e.g., collection size, collection characteristics, …)
  - details of engineering decisions such as preprocessing (e.g., discussing how you preprocess documents, did you use techniques like deduplication or other forms of preprocessing, …)
  - details of your search algorithms, possibly including improvements (e.g., you don't need to elaborate in detail on the models you use, but you should make sure you provide baselines and talk about improvements you've made, just as we did in Assignment 2)
  - visualizations, results, and overall analysis (e.g., the survey papers provide examples of ways to visualize results and in class we learned about evaluating retrieval systems. If you choose to focus on the engineering of the problem (bigger dataset without an evaluation), then you will discuss how that looked for your team).
  - challenges you encountered and conclusions
- **The upshot is that the final deliverable will be a Dockerized pipeline enabling search over a large collection as in this demo. Your team may evaluate your approach (on CRISISFacts or NeuCLIR), or you might use a very large collection instead. Your focus will be on good software design, engineering carefully by taking into consideration the dataset size, and running experiments to see how the improvements you make affect some aspects of your system performance (e.g., retrieval quality, memory usage, latency, or others). You are not expected to come up with a new search approach, but your team might take a current one (like ColBERT) and read some of the surveys to understand what improvements are possible and implement some of them.**

- **The expectation is that this will involve aspects of Assignment 2 (where you started with an algorithm and improved on it thoughtfully) and Assignment 3 (where you fit pieces of your pipeline into Docker).**
- be thoughtfully-designed and organized

Your **team's presentation** should follow the best practices we will outline when we think about effective presentations, in addition to whatever flare your team brings during showtime. Your goal, of course, is to tell a memorable story—have fun with it. Your presentation should be no less than fifteen minutes and no more than twenty minutes. In addition to a slide deck, you may want to provide a live demo.