Wmjg

NLP Senior Seminar

Final Project

May 8, 2023

FirstToSay

The idea came from the "UNIQUE" database constraint -- what if you could ONLY post Unique posts into a database. How would that work? What counts as unique? I sought to answer that question using similarity metrics. And in the process, if I got to build and launch a website too, that's just a bonus.

So, what did I do: Built a flask web app with sqlite database handled with sqlalchemy and an elasticsearch backend to query user texts, analyze them and fetch similar text to see if that text has ever been entered into the db. The basic user instruction: "Try to say something original."

Next, the how:

- FLASK -- the views/models/authentication blueprints and login handling to serve Users
- Sqlite3 -- to handle the tables of Users, Quotes, and Likes
- Normalization -- (casing, stripping, utf8 encoding) to get proper readable text from users
- Space language detection -- as a shorthand for grammaticality of the inputs
- Elasticsearch -- index to QUICKLY find similar texts in the data with custom analyzer
- Difflib -- cosine similarity of ES results for the final score similarity ranking
- RESTFUL – Flask handling all the redirecting and backend-frontend communication
- HTML/Javascript/Bootstrap – displaying and beautifying
- Started with 1/2 million quotes from kaggle to prefill the db
    - https://www.kaggle.com/datasets/manann/quotes-500k
    - https://www.kaggle.com/datasets/iampunitkmryh/funny-quotes
    - https://www.kaggle.com/datasets/faseehurrehman/popular-quotes
    - https://www.kaggle.com/datasets/abhishekvermasg1/goodreads-quotes
- Pandas -- efficiently organize the data, removing duplicates, then sql inject into the db
- Docker -- made it launchable and accessible with gunicorn and nginx

And launch we did in class, handling 6-7 concurrent users and about 30 inputs, all user inputs that day are still in the live production database at **firsttosay.com:8080**, including some which saw errors on the day, but made it into the backend.

As it stands; firsttosay handles:

- Users: registering, login, logout
- User entered text on the home page
    - Similarity lookup, querying based on normalized text
    - Generating Quotes from User entered text

- Similar quotes results page
    - Index lookup similar texts
    - Similarity scoring index results
- Quotes displaying:
    - User entered text results with similarity scores
    - User profiles displaying all Quotes (accessible via click or url)
- Likes, you can like quotes, which is tracked under User table

A big goal which didn't make it into the process, I wanted to implement a more fun Profile page, to see what other authors you're similar to and then we could do stats with topic modelling or just a simple word cloud. But those were stretch goals, and I got stuck learning about server hosting and infrastructure -- took up a lot of time.

Problems arose due to scope, because my eyes were bigger than my stomach, there were stretch goals:

- "Followers": track other User authors you're getting similar entries to, display in profile (WIP)
- Topic Model: display in profile your most common topics
- cookies and tracking: can't see your entry result more than once; refresh or back loses that similarity results page
- Gamification: 3 strikes and you're out, track longest streak of user inputs (WIP)

These weren't major features, but they were disappointments to leave on the cutting room floor. However I did learn a lot about Flask and Docker as compensation.

I worked alone. (though I did have help from a flask tutorial youtube.com/watch?v=GQcM8wdduLI)

In order to replicate the site that's running right now at firsttosay.com:8080, you should use the docker-compose that's on github. Here are the steps:

- Git the repository at https://github.com/wmjg-alt/FirstToSay (master branch, not prototype)
- Download the 4 kaggle files into a data/ directory
- Rename one of the "quotes.csv" files to "quotes2.csv"
- Have Docker Desktop running in the background
- Then from cmd line in the top level of the directory with app.py run:
    - docker-compose up –build

This will build you a local flask server you can access from localhost in a browser after it builds the elasticsearch index, nginx workers, sqlite database (and then fills it with 1/2 million quotes). Your database will not contain the user-inputs from the in-class launch.

Some major adjustability and honing comes in the form of the thresholds I used to define certain steps in the process. These came from a lot of testing, including gibberish inputs, repeat inputs, inputs similar in characters to existing inputs, and inputs similar in content to existing inputs.

The goal was for a user to be able to input 2 texts with the same meaning and successfully generate a quote as long as they were distinct in vocabulary. Minimal changes like punctuation shouldn't count.

- "I love dogs." vs "I love dogs" -> too similar, fail to generate a Quote
- "I love dogs" vs "I am fond of dogs" -> dissimilar, generate Quote

Rock bottom assumptions this is all built on:

- Spacy at least 50% confident your text is English functions similarly to grammaticality
- A cosine similarity score is a better text of character similarity than es analyzer
- A cosine similarity score 95% or more indicates a near-copy that's too similar
    - Fooled by longer text inputs
- No one will notice the 256-character limit for one "sentence" input