IR Project

Blog Search

* Lakshya Boob S20200010109
* Ashutosh Rawat S20200010025

Preface

# Dataset

[Blog Authorship Corpus | Kaggle](https://www.kaggle.com/datasets/rtatman/blog-authorship-corpus)

Although this consists of 600,000 posts from 2004, we have just considered top 10000 due to some memory related issues of our system.

# Caution

While searching please keep in mind that the data is from 2004 and please search accordingly.

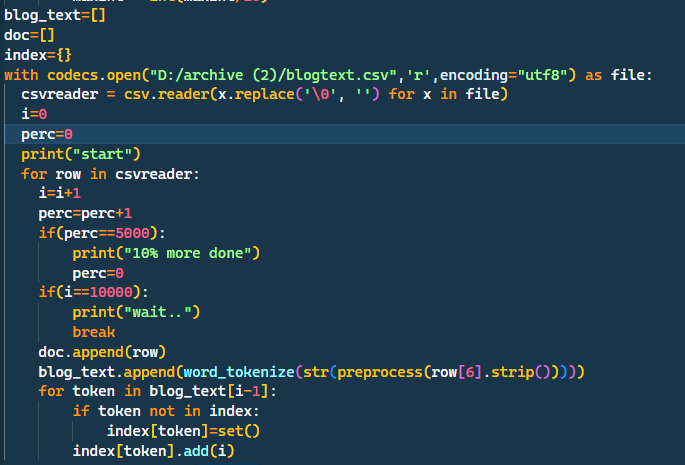
Indexing Components

This is offline process.

The data structure used for indexing is dictionary in python (kind of like HashMap). It will take the words as indices or keys and the values will be their respective posting lists which is again taken as a set in python.

The time complexity of lookups in dictionary is O(1) making the retrieval of posting lists extremely fast. Also since the posting list is in form of set so, merging or taking intersection of posting lists is efficient (O(mlogn) ;m and n being the sizes of posting lists ).

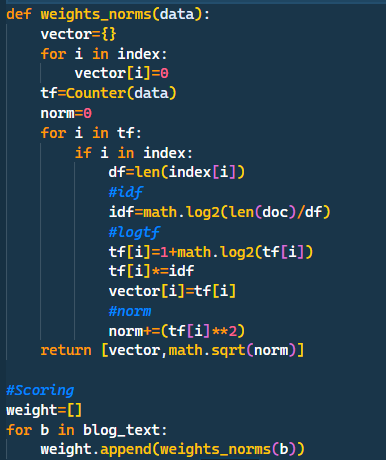
In the posting lists docIDs of all documents are stored in which the word appears.



Searching Components

# Ranking

Ranking is done based on cosine similarity. Weight is calculated as product of Inverse document frequency and log term frequency.

In this step, document vectors and their norms are calculated offline. 

In weights\_norms function, Counter is used which returns the no. of times a particular word has occurred in a document which is further used to calculate log term frequency. Idf is calculated according to formula.

Below the weight array stores all the documents vector along with their norms.

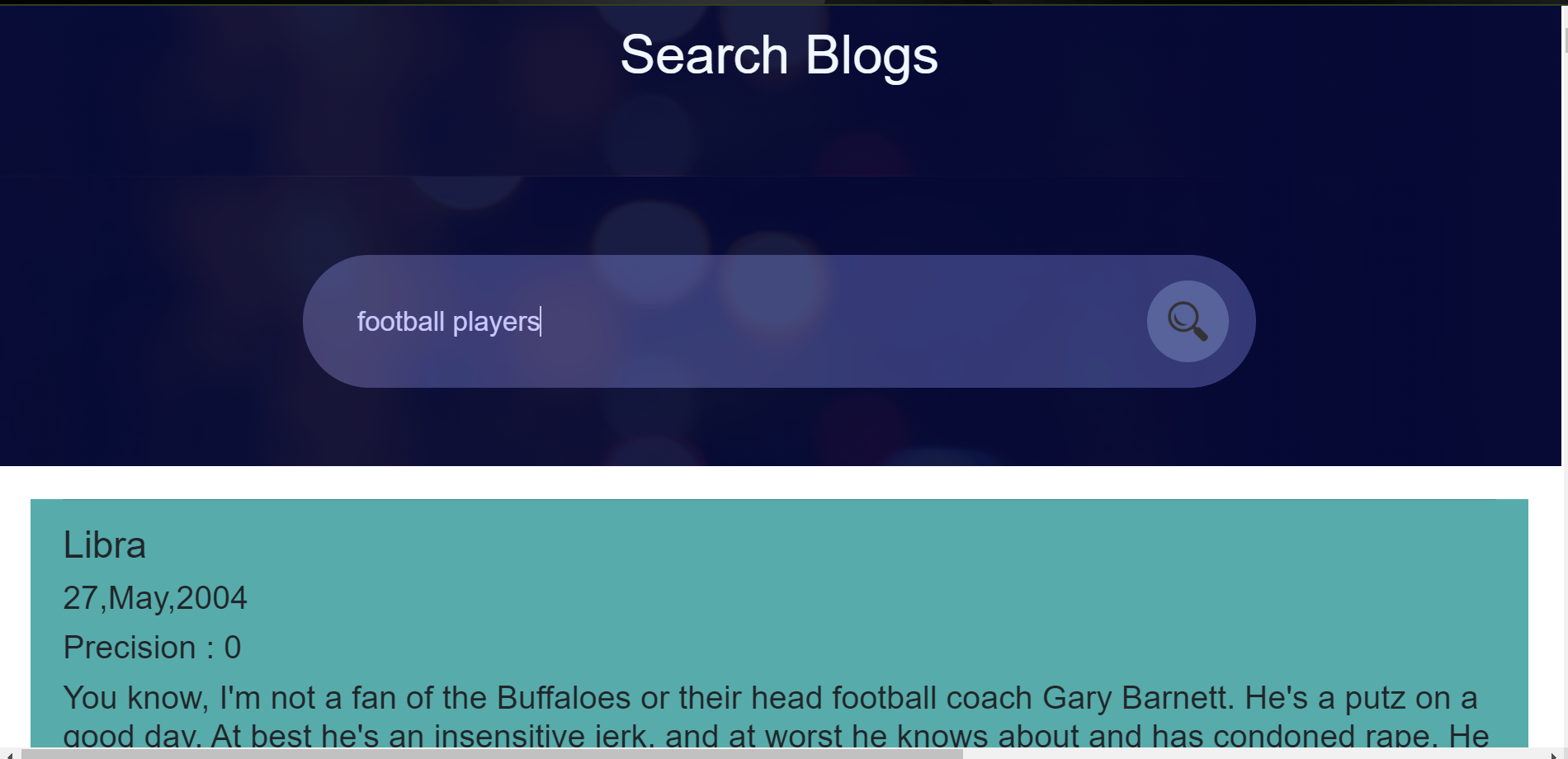
# Searching



Search function takes query as a string and then generate tokens out of it using preprocess function. Retrieves all the required posting lists and stores them in variable results and then their intersection is taken to generate an array of document ids.

Query vector is calculated using weights\_norms function.

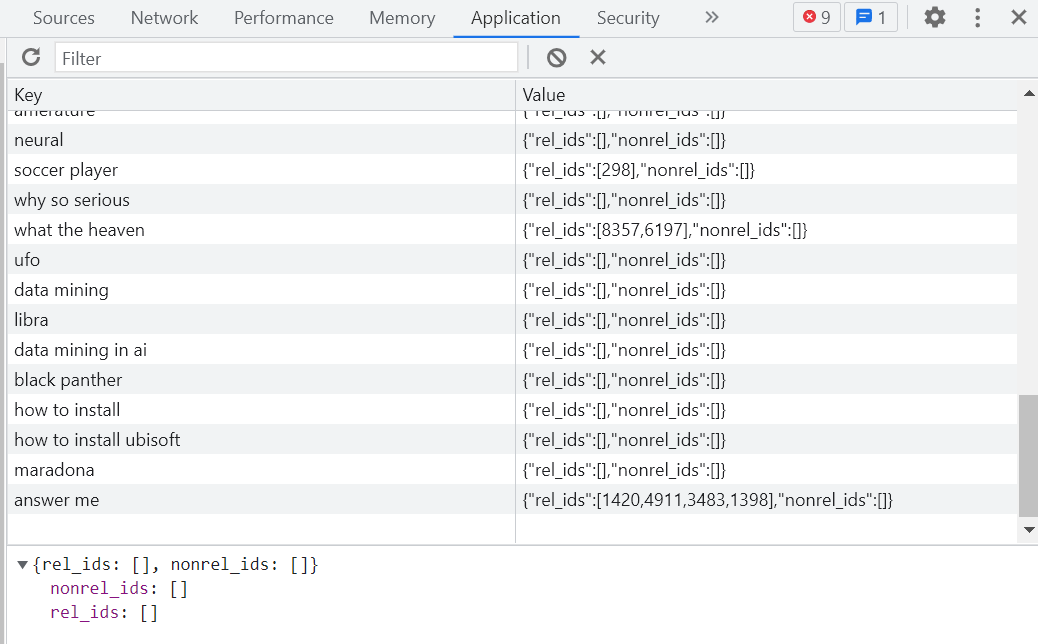
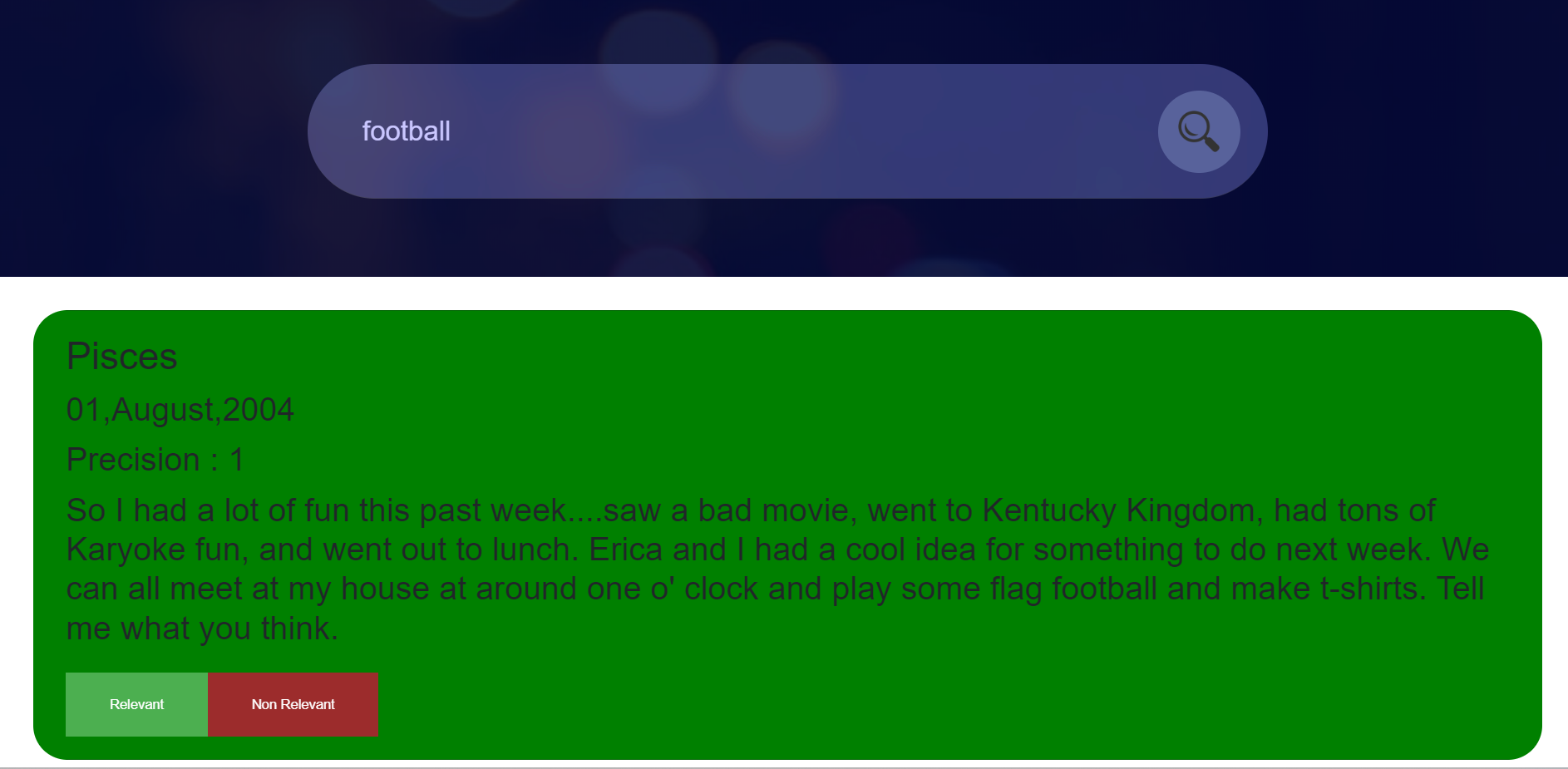
Then cosines are calculated between the query vector and all the retrieved documents using cosine function defined just above search and are stored in an array along with docID which is then sorted and returned.



Top 10 documents are shown on first page and the following on the next page and so on…

Relevance Feedback

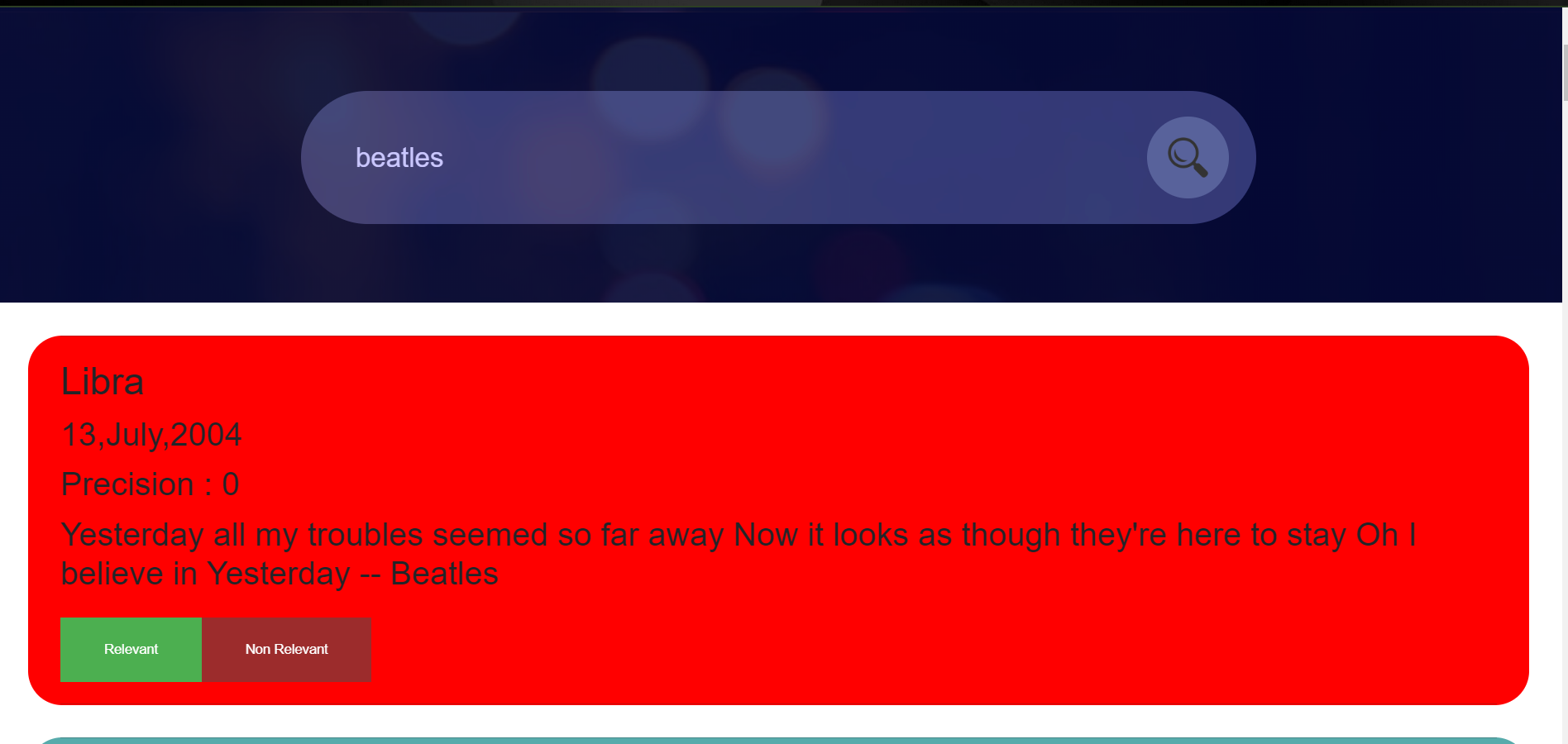
# Capturing Feedback

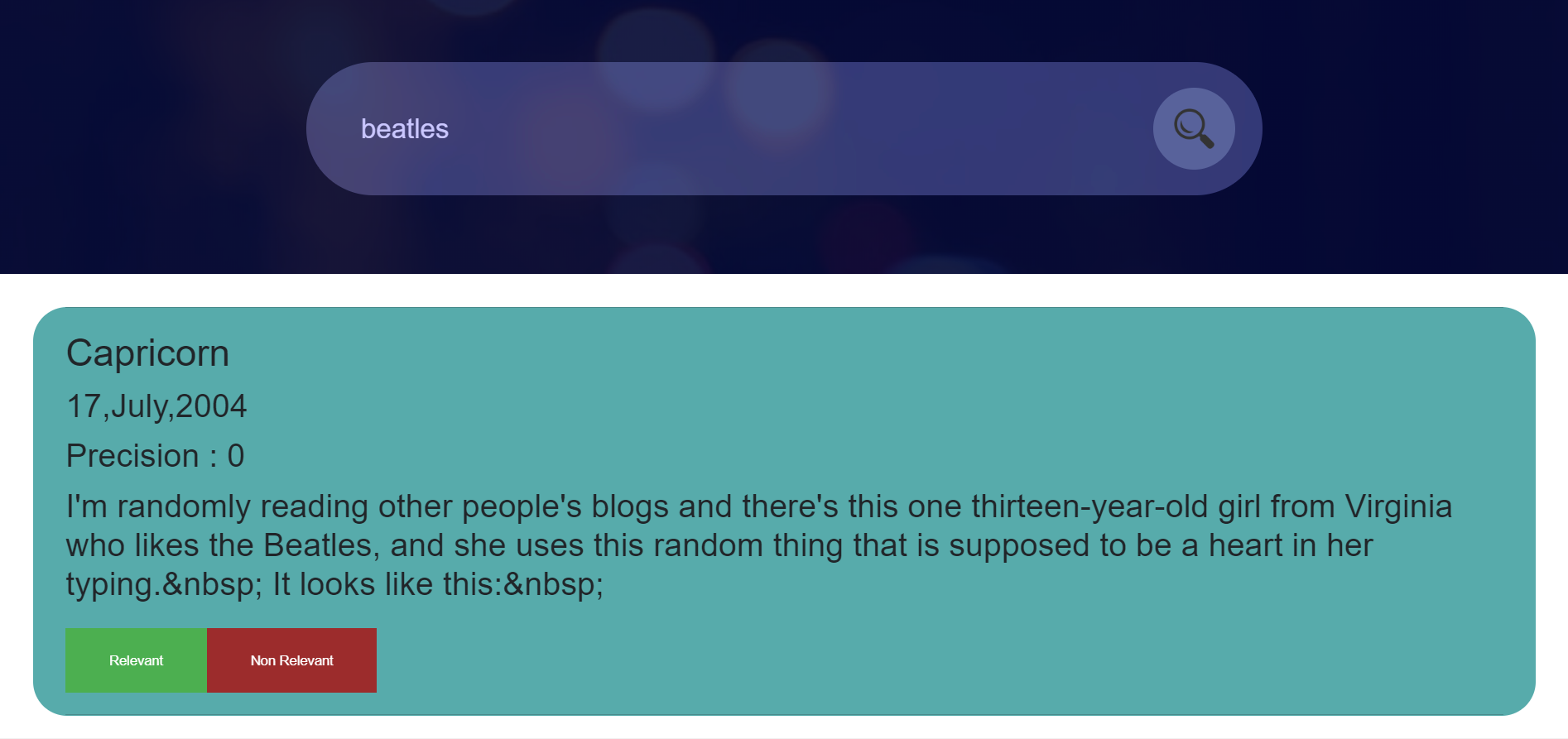
There are 2 buttons (Relevant and Non relevant provided on each documents). As soon as user provides their feedback by clicking either of them it gets stored in the localStorage along with the query as a key. 

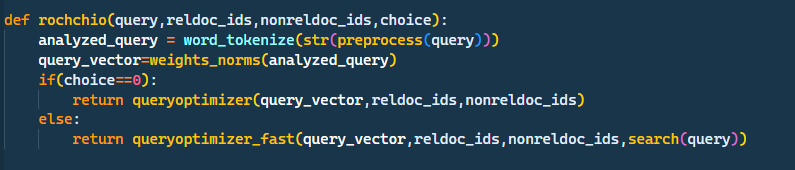
Any no. of documents can be marked as relevant and non relevant.

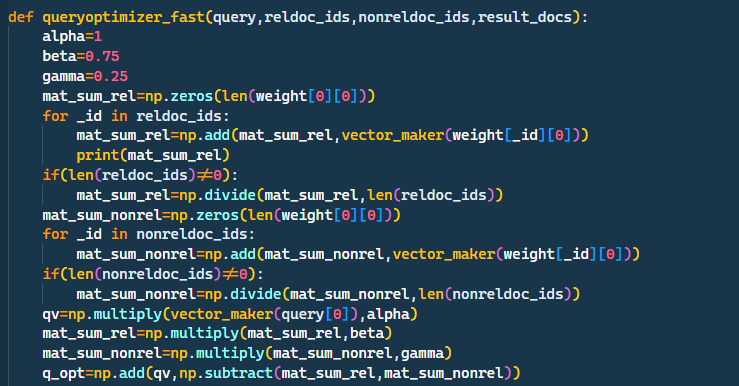
# Using Feedback to Improve Retrieved Results

So, as the user enters the same query again, they get refined results.

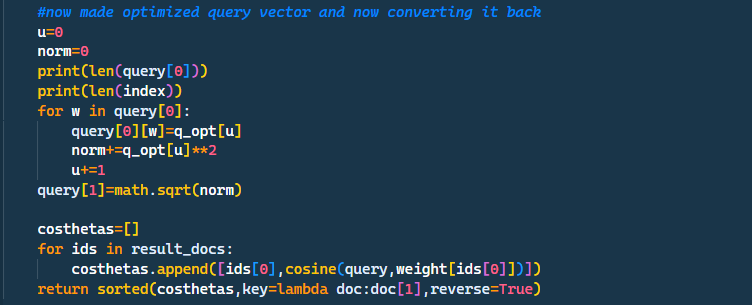
For eg- for the query ‘beatles’, I have marked first document as non relevant

So the next time I search it , this document will not be retrieved. 

To achieve this Rochchio’s (smart) algorithm is used in the backend which takes in the query vector and an array of relevant document IDs and non relevant document IDs

This is just a function the main algorithm is in queryoptimizer\_fast function

Here, I have hardcoded the alpha, beta and gamma values. At first, I am calculating the centroids and then applying the formula.

Then I am searching for the optimized query vector and returning the results. 

Assessment Components

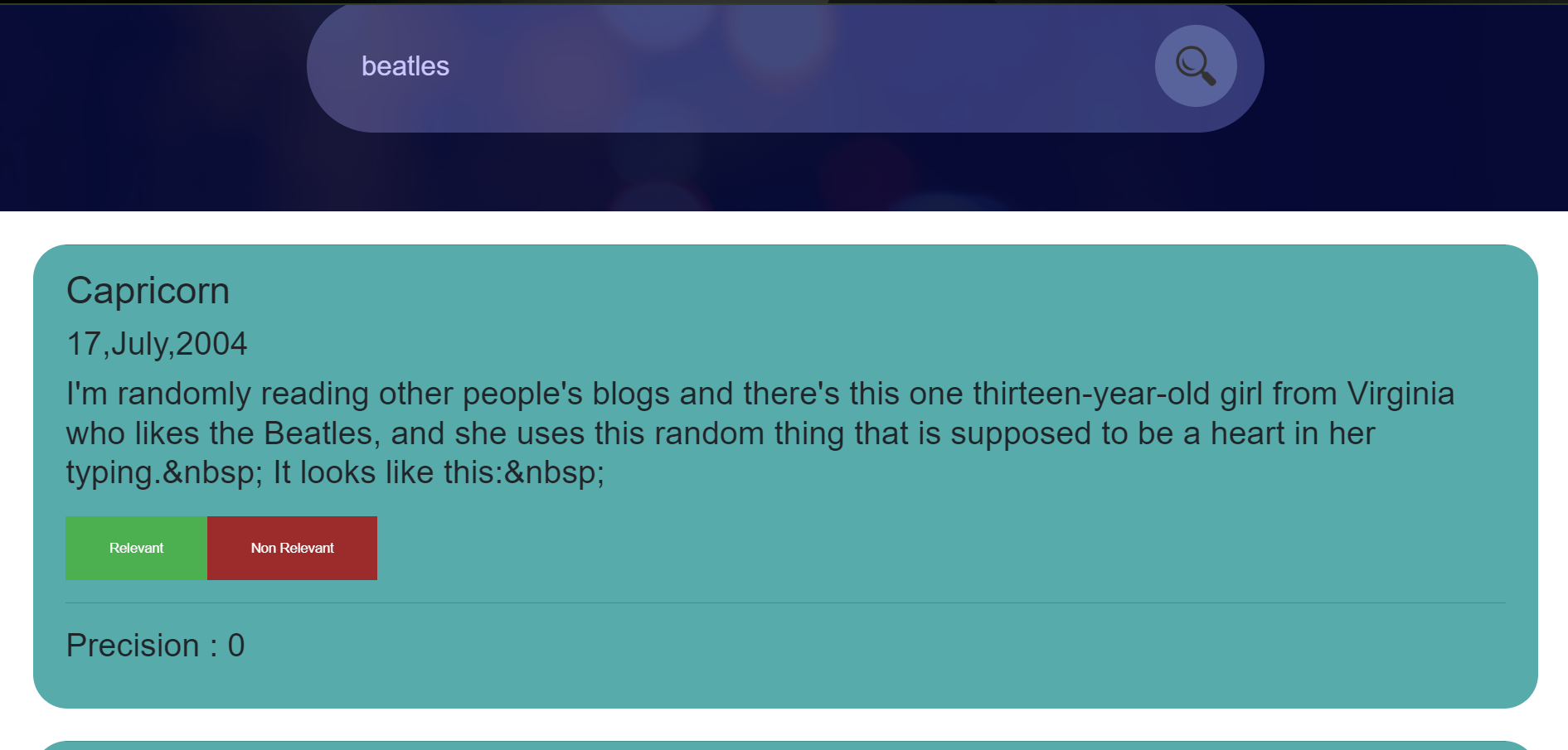
# P@n

Precision is defined as total no. retrieved documents that are relevant.

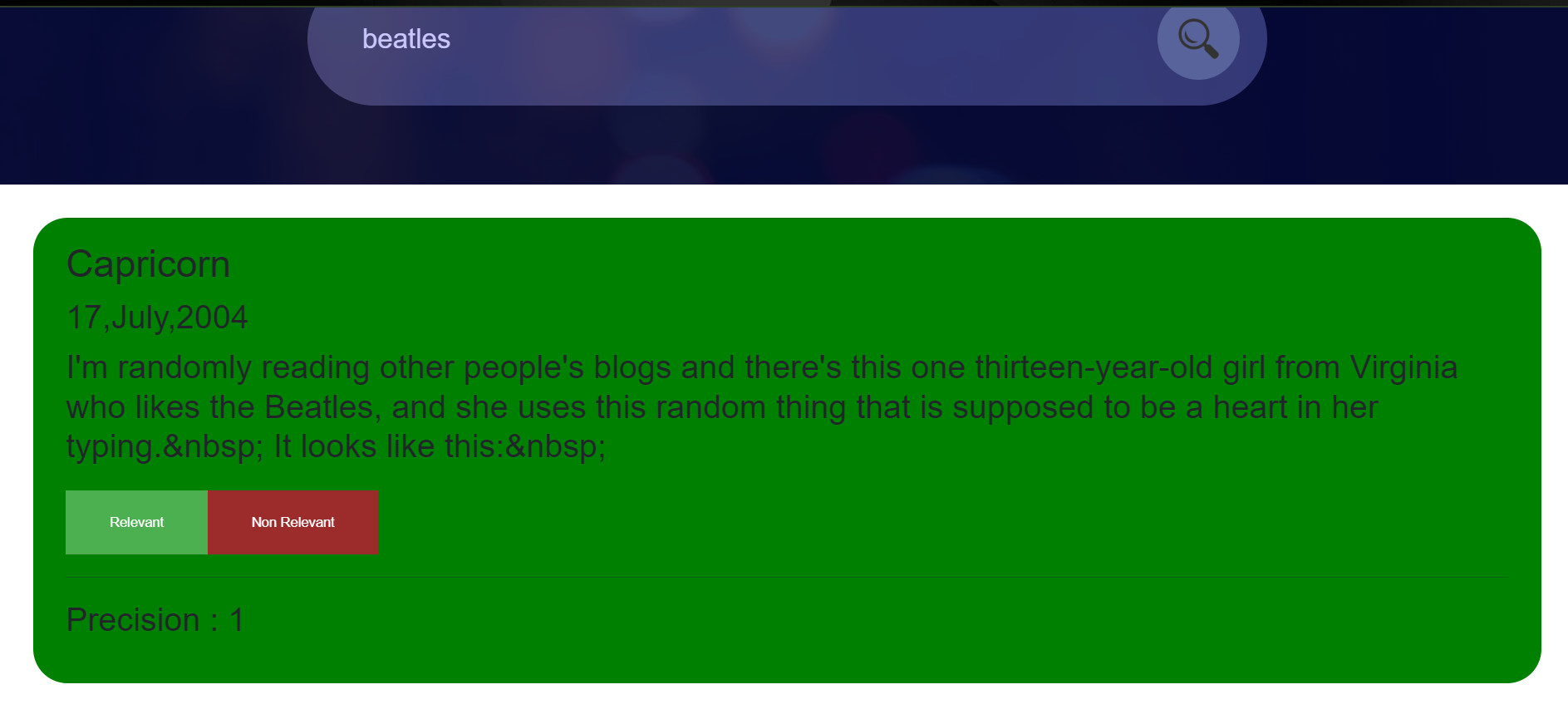
As the user provides their feedback. Precision at that position is calculated and displayed.

Initially all the documents are considered as non relevant. This changes only on user feedback.

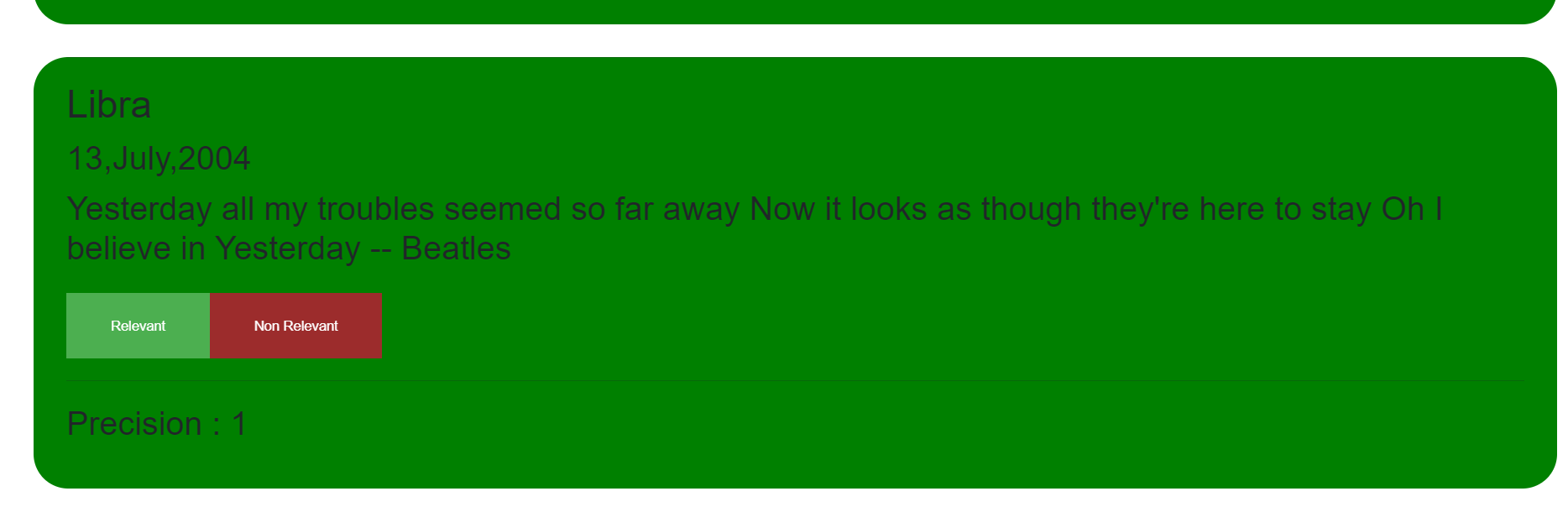
We are calculating it totally based on user’s input as we don’t know which is document is actually relevant beforehand. That’s why, Precision is 0 initially and changes as user provides their feedback.

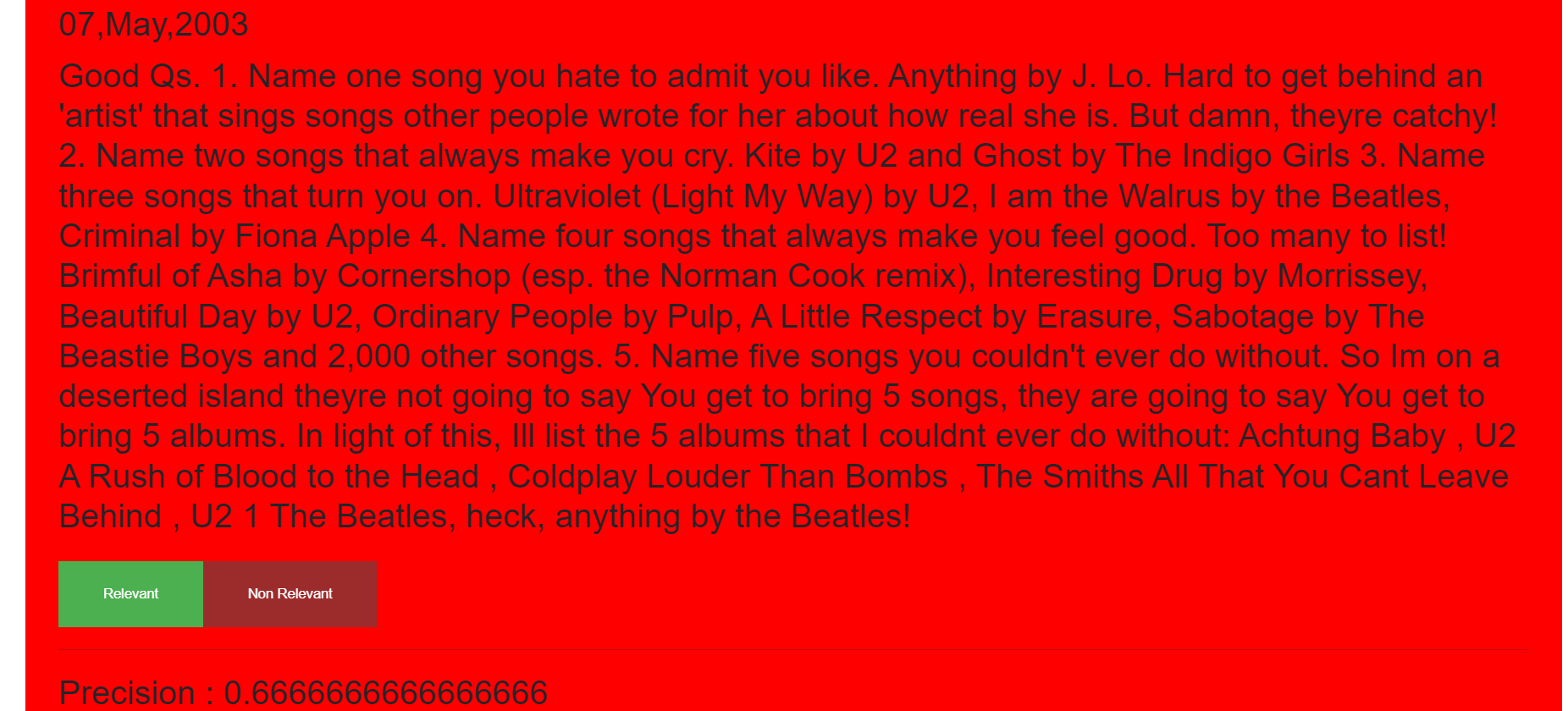
For eg- for query=”beatles” initially, 

As we choose relevant and non relevant, precision changes

P@1

P@2



P@3

And So on……….

# Recall

Recall is total no. of relevant documents that are retrieved. We are not displaying recall since we can’t calculate it as we do not know the no. of relevant documents beforehand.

# P-R Graph

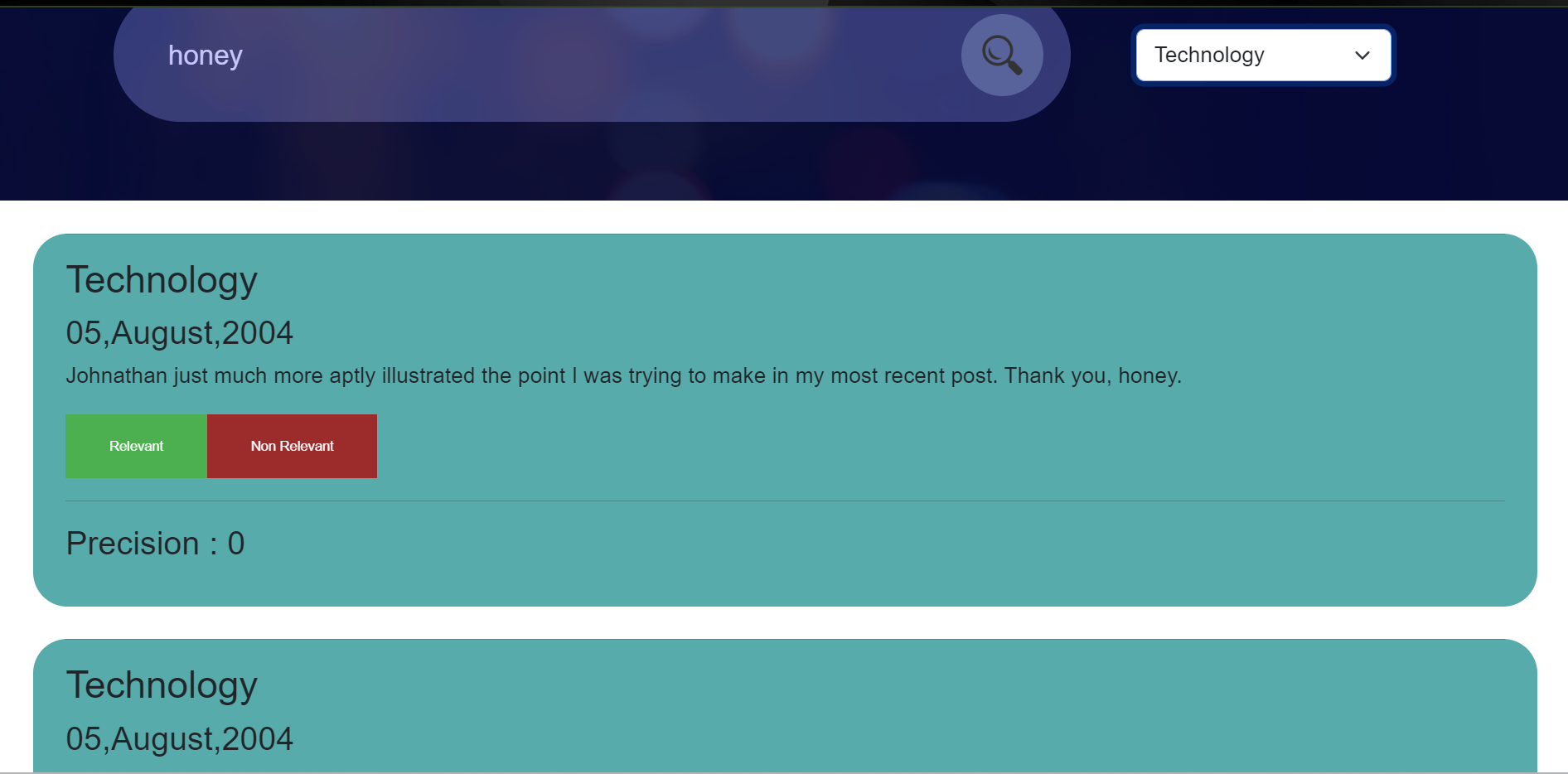
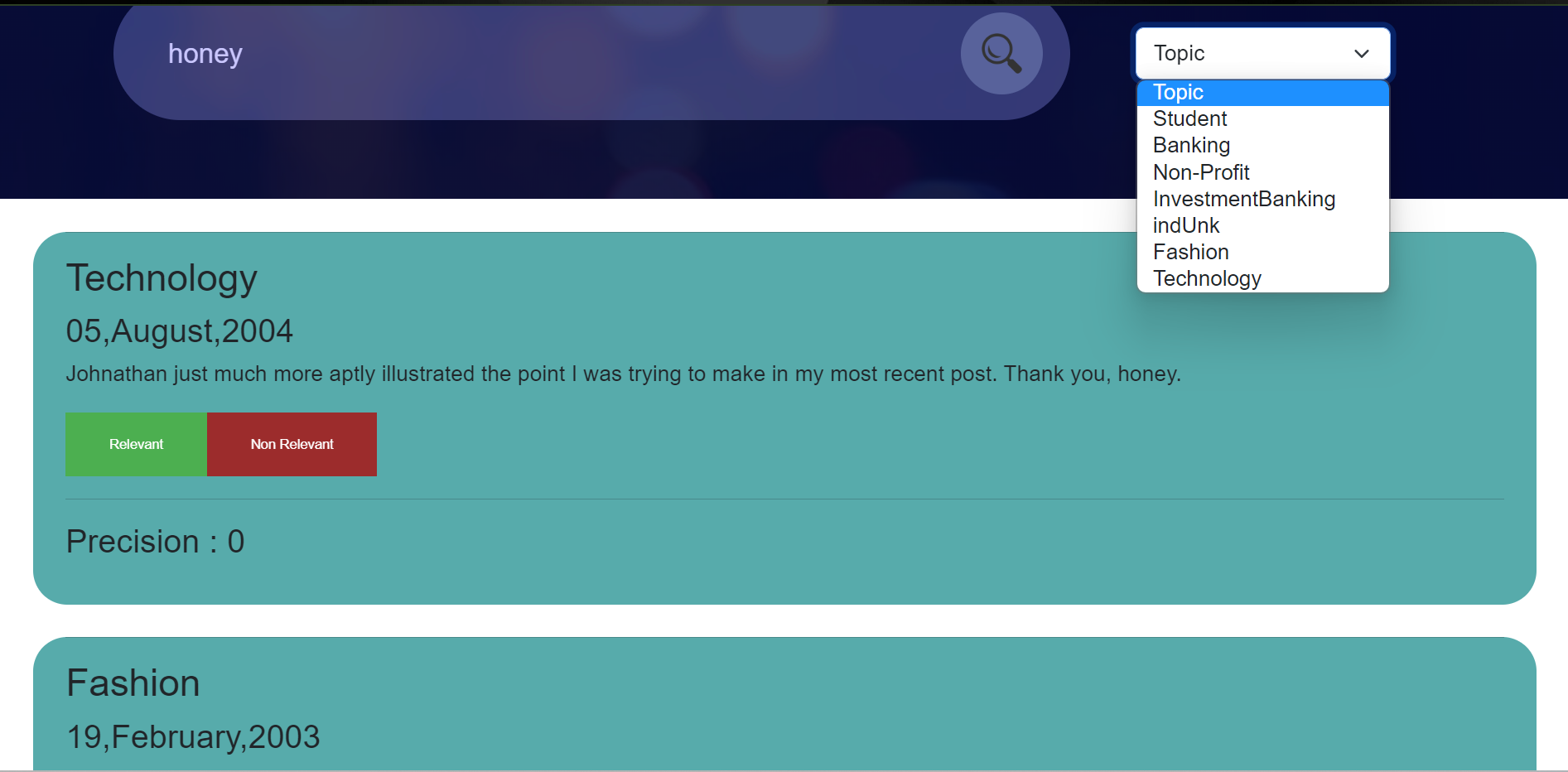
Since we can’t calculate recall (reason given above). It doesn’t make sense to plot a P-R graph.

Refining Searches based on specific filters

# One way to refine searches is by giving feedbacks

Discussed above

# Second Way

We have provided a filter facility based on the topic field in the csv data. User can select a topic and results from only those topic will be retrieved. 

Please note that the blog topic is completely given by blogdata corpus and we aren’t responsible if they don’t make sense with the blogs. (You can check with the dataset)

Contributions

Indexing, Searching(with ranking) , Using feedback to improve retrieved ranked results 🡺 Lakshya Boob S20200010109

Assessment component, Refining Searches , capturing feedback 🡺 Ashutosh Rawat S20200010025