Indian Institute of Technology, Kanpur

CS657: Information Retrieval
Project Report

Title: Efficient Information Retrieval System using Stack Overflow data.

Supervised By: Prof. Arnab Bhattacharya

27 April 2022

Submitted By: Group 8

Gajender Sharma (21111028)	gajenders21@iitk.ac.in
Kajal Sethi (21111033)	kajals21@iitk.ac.in
Pranshu Sahijwani (21111048)	pranshus21@iitk.ac.in
Utkarsh Srivastava (21111063)	utkarshs21@iitk.ac.in

Contents

List of Abbreviations

BERT	Bidirectional Encoder Representations from Transformers
CSV	Comma Separated values
ES	Elastic Search
HTML	Hyper Text Markup Language
IR	Information Retrieval
IDF	Inverse Document Frequency
NLP	Natural Language Processing
TF	Term Frequency

1 Abstract

Everyone must have used Stackoverflow when searching for a query in a code or installing some applications. Stackoverflow is a public platform building the definitive collection of coding questions & answers with millions of questions and multiple answers written by users so our idea is to build an efficient information retrieval system which is question search engine for the queries which gives blazing fast results in measurements of milliseconds with added different functionality like spellchecker, showing results by relevance, popularity, creation date, etc for giving variability to the use case of query search. People suffer from latency issues when the dataset is too large to work with so we want to access the data quickly, other issues are that working with large amount of data requires high computational resources and server costs, so our working will work toward solving this issues as well. Earlier few authors have made use of stackoverflow in predicting whether questions will be upvoted, downvoted, or closed based on their text or Predicting how long questions will take to answer and other similar things to enhance the functionality of the search engine but our work is mainly focused on extracting the top most relevant questions to a particular query in less time using Elastic search and also showing similar questions by using semantic searching using BERT transformer model. Moreover, Efficient Information Retrieval is a crucial part in the field of NLP.

2 Goals of the Project

- Low latency in providing search results even with large dataset.
- An IR system with low computational/server cost.
- An IR system with high precision and recall rate.
- Easily deployable and scalable

3 Dataset

The dataset consists of 10% of questions and answers from the Stack Overflow programming QA website which has a size of around 12,00,000 question.

The dataset is organized into three tables(csv's):

- **Questions** - Contains the title, body, creation date, closed date (if applicable), score, and owner ID for all non-deleted Stack Overflow questions whose Id is a multiple of 10.
- **Answers** - Contains the body, creation date, score, and owner ID for each of the answers to these questions. The ParentId column links back to the Questions table.
- **Tags** - Contains the tags on each of these questions.

The dataset roughly contains 1.26M questions with 7 columns in csv file and 2.01M answers with 6 columns each. Figure 1 shows the data in raw form.

```
df_questions.loc[0]['Body']
```
✓ 0.8s Python

```
'<p>I\'ve written a database generation script in <a href="http://en.wikipedia.org/wiki/SQL">SQL</a> and want to execute it in
my <a href="http://en.wikipedia.org/wiki/Adobe_Integrated_Runtime">Adobe AIR</a> application:</p>\n\n<pre><code>Create Table
tRole (\n      roleID integer Primary Key\n      ,roleName varchar(40)\n);\nCreate Table tFile (\n    fileID integer Primary
Key\n    ,fileName varchar(50)\n    ,fileDescription varchar(500)\n    ,thumbnailID integer\n    ,fileFormatID integer\n
,categoryID integer\n    ,isFavorite boolean\n    ,dateAdded date\n    ,globalAccessCount integer\n    ,lastAccessTime date\n
,downloadComplete boolean\n    ,isNew boolean\n    ,isSpotlight boolean\n    ,duration varchar(30)\n);\nCreate Table tCategory
(\n    categoryID integer Primary Key\n    ,categoryName varchar(50)\n    ,parent_categoryID integer\n);\n...\n</code>
</pre>\n\n<p>I execute this in Adobe AIR using the following methods:</p>\n\n<pre><code>public static function
RunSqlFromFile(fileName:String):void {\n    var file:File = File.applicationDirectory.resolvePath(fileName);\n    var
stream:FileStream = new FileStream();\n    stream.open(file, FileMode.READ)\n    var strSql:String =
stream.readUTFBytes(stream.bytesAvailable);\n    NonQuery(strSql);\n}\n\npublic static function
NonQuery(strSQL:String):void\n{\n    var sqlConnection:SQLConnection = new SQLConnection();\n
sqlConnection.open(File.applicationStorageDirectory.resolvePath(DBPATH);\n    var sqlStatement:SQLStatement = new
SQLStatement();\n    sqlStatement.text = strSQL;\n    sqlStatement.sqlConnection = sqlConnection;\n    try\n    {\n
sqlStatement.execute();\n    }\n    catch (error:SQLError)\n    {\n        Alert.show(error.toString());\n    }\n}\n</code>
</pre>\n\n<p>No errors are generated, however only <code>tRole</code> exists. It seems that it only looks at the first query (up
to the semicolon- if I remove it, the query fails). Is there a way to call multiple queries in one statement?</p>\n'
```

Figure 1: Raw unclean data.

4 Data Preprocessing

The following steps were done in order to clean the data.

- Removal of HTML tags- Since every HTML tag is enclosed within angular brackets (<>), we used egex to replace every string starting with '<' and ending with '>' with an empty string
  Example Input: str = "IITK"
  Example Output: IITK

- Removal of stopwords- Stop words are the commonly used words (such as "the", "a", "an", "in") that a search engine ignores, both when indexing entries for searching and when retrieving them as the result of a search query.

- Removal of punctuations- Just like stopwrods, punctuations also need to be removed as they solve no useful purpose.

- Fill NA values.

- Concatenated Questions.csv, Answers.csv and Tags.csv.

- Removed questions for which there were no answers.

- In Answer.csv there were multiple answers for a given question Id so we concatenated all the answers to each questions.

- In Tags.csv there were many tags corresponding to a questions so we added them to their respective questions too.

- Then we also encoded the question's title and question's body with google encoder which generates the 512-Dimensions vectors for each questions which will be used in the ingestion part of the Elastic search to aid our latency in query searching.

Figure 2 is the preprocessed data.

```
df_questions.loc[0]['Body']
```

✓ 0.5s Python

```
"I've written a database generation script in SQL and want to execute it in my Adobe AIR application:I execute this in Adobe AIR
using the following methods:No errors are generated, however only  exists. It seems that it only looks at the first query (up to
the semicolon- if I remove it, the query fails). Is there a way to call multiple queries in one statement?"
```

Figure 2: Clean data.

4.1 Encoding

For generating a 300 dimensional vector for each paragraph we have used Fast Sentence Embeddings "fse" which is a python library for Ultra Fast Sentence Embeddings. It acts as an addition to the Gensim library. FSE has many features which make it suitable for our embedding task at hand. Some of them are listed below-

- It can process up to 500.000 sentences / second
- Supports Average, SIF, and uSIF Embeddings
- Induction of word frequencies for pre-trained embeddings
- Ram-to-disk training for large corpora.

The final attributes that were used to build the query engine are : Id, CreationDate, Score, Title, Body,Answer_Score, Answer_Body, Tag,Encodings that were fed to the Elastic search

4.2 Challenges

1. Data was taking too much space on our system's RAM memory so we worked our dataset in chunks of 2 lakhs rows at a time.
2. Most challenging part was to vectorize the question's title and tags for a particular question for the quick search so that most relevant words in questions body is used but not the whole paragraph which we resolved using Glove model.
3. Ingesting sentence vectors into Elastic-Search (JSON Serialization issues).Data was taking too much space on our system's RAM so we worked our dataset in chunks of 2 lakhs rows at a time.
4. Most challenging part was to vectorize the question's title and tags for a particular question for the quick search so that most relevant words in questions body is used but not the whole paragraph which we resolved using Glove model.
5. Communication between Elastic-Search and Python is not so fluent

5 Flow of the project

5.1 Query processing

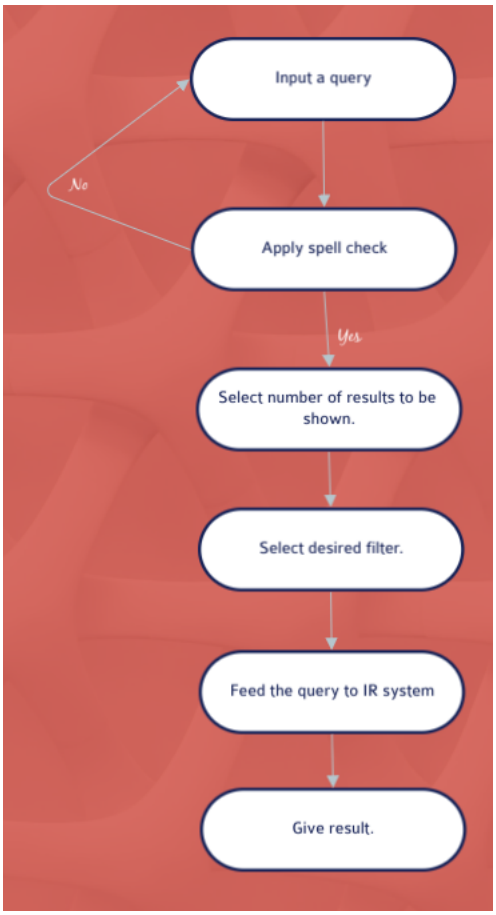

Figure 3: Flow of the Query processing.

The user inputs the query to our information retrieval system which then applies a spelling check on the query entered. If the query contains spelling error then the user need to input the query again. If the query is accepted, then the user is asked to select the number of results which is set to a default value of 10. After this, user can apply desired filters. The query is send to the Information Retrieval model(explained in detail in next section) which show a list of questions according to the filters applied by the user. The questions are given with hyperlinks which will directly relocate to the Stack Overflow page of the selected question.

5.2 Information Retrieval processing

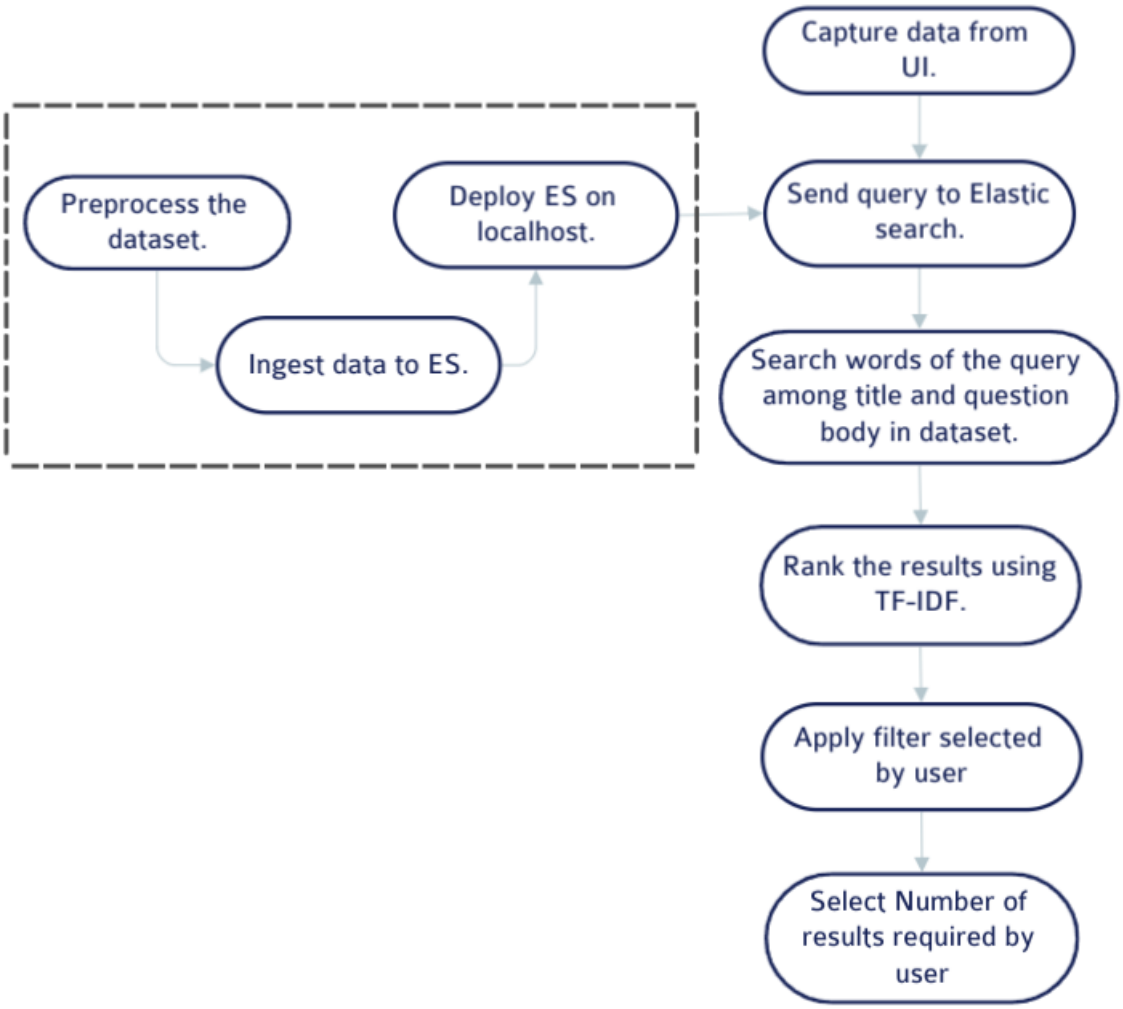

Figure 4: Flow of processing query in backend.

We preprocess the dataset containing 12,64,216 questions. After preprocessing the data we finally got 11,02,568 questions. This dataset is then ingested to Elastic Search which is a popular search platform and deployed on the localhost. Elastic search has been used as it could run on distributed sytems thus able to process large amount of dataset. It also enables us to give realtime results and is a inverted indices based datastore which is key requirement for performing the operation. This state of the system is saved.

When the user inserts a query in the system along with desired number of results and the method to filter the results, it is send to Elastic Search, which uses TF-IDF algorithm to rank the documents according to the similarity score. This similarity operation is performed on the question title, body and tag of questions in the dataset. This gives us a ranked list of documents according to cosine similarity. We then perform the filters specified by the user and then show the list of results with the hyperlinks.

6 Methodology

In this section, we explain the methodology adopted and followed for the design and development of this project.

6.1 Inverted Index

An inverted index is created of all the unique words in the vocabulary.Inverted index is a type of data structure that allows for extremely quick full-text searches. An inverted index lists every unique word in each document and identifies all of the documents in which that word appears. Each document is a collection of fields, which are the key-value pairs that comprise your data, and an index may be thought of as an optimised collection of documents. Inverted indexing was done with Elastic Search, which indexes all data in every field and gives each indexed field has a specialised, optimised data structure. As a result, we produced an inverted index of each unique word, with values corresponding to question IDs, containing this specific word and its frequency of appearances in that question. Figure 5 shows the data in Elastic search.

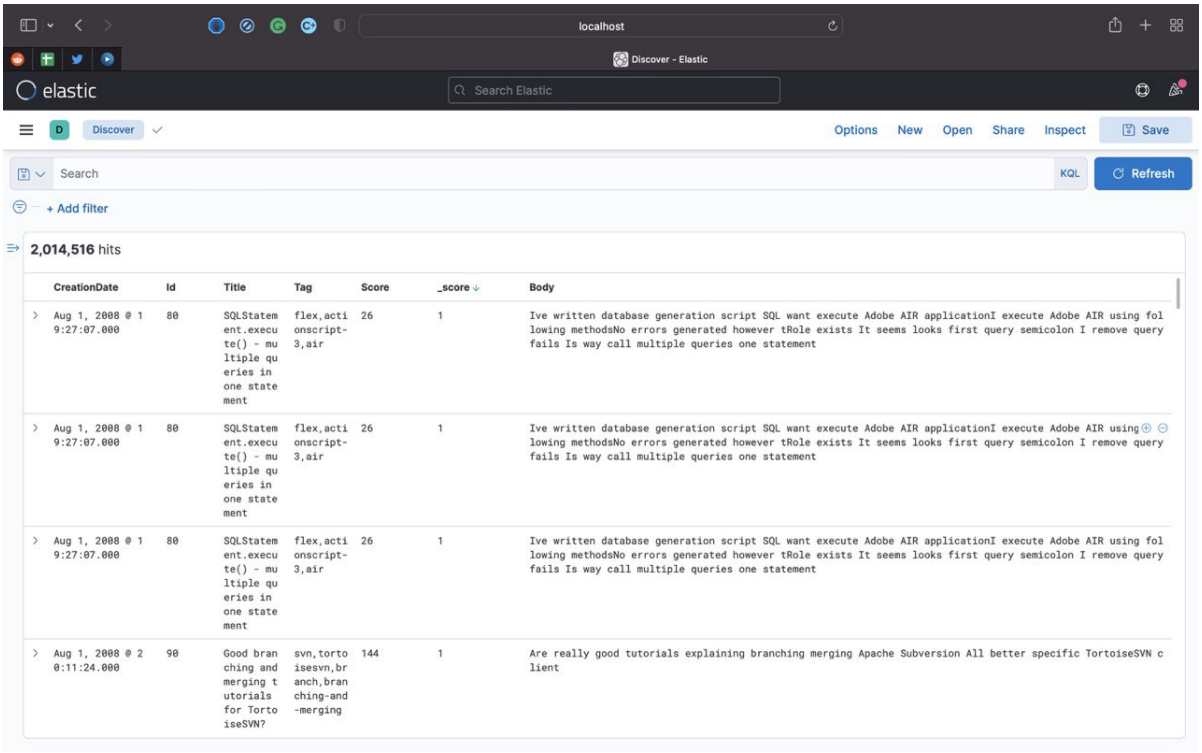

Figure 5: The indexed data in Elastic Search.

6.2 Query searching

We can use various Information retrieval models like TF-IDF, Word2Vec,Bag-of-words, BERT,BM-25 etc but we are chose Tf-Idf, applying other models can be one of our future scopes for this project. Let's dig deeper into the Tf-Idf information retrieval model and understand its working.
TF-IDF (Term frequency-Inverse document frequency and is used in the fields of information retrieval (IR) and machine learning. In a collection of documents, TF-IDF can quan-

tify the importance or relevance of string representations (words, phrases, lemmas, etc.) in a document (also known as a corpus). TF-IDF is a useful technique that determines the relevance of words based on their frequency in a document. It's a straightforward but intuitive method for weighing words, making it a useful starting point for a range of activities. Building search engines, summarising documents, and other information retrieval and machine learning tasks are examples.
TF-IDF comprises of two parts TF (term frequency) and IDF (inverse document frequency).

6.2.1 Term Frequency

It works by looking at the frequency of a particular word you are concerned with relative to the document. There are multiple measures, or ways, of defining frequency:

- Number of times the word appears in a document (raw count).
- Term frequency adjusted for the length of the document (raw count of occurences divided by number of words in the document).
- Logarithmically scaled frequency (e.g. log(1 + raw count)).
- Boolean frequency (e.g. 1 if the term occurs, or 0 if the term does not occur, in the document).

6.2.2 Inverse Document Frequency

Inverse document frequency looks at how common (or uncommon) a word is amongst the corpus. IDF is calculated as follows where t is the term (word) we are looking to measure the commonness of, and N is the number of documents (d) in the corpus (D).. The denominator is simply the number of documents in which the term, t, appears in. It can also be possible for a term to not appear in the corpus at all, which can result in a divide-by-zero error. One way to handle this is to take the existing count and add 1. Thus making the denominator (1 + count).

Scikit-Learn

- IDF(t) = $\log \frac{1+n}{1+df(t)} + 1$

Standard Notation

- IDF(t) = $\log \frac{n}{df(t)}$

The reason we need IDF is to help correct for words like "of", "as", "the", etc. since they appear frequently in an English corpus. Thus by taking inverse document frequency, we can minimize the weighting of frequent terms while making infrequent terms have a higher impact.
Finally, IDFs can be retrieved from either a background corpus, which corrects for sampling bias, or the dataset utilised in the current experiment. To summarise, the essential premise of TF-IDF is that a term's relevance is inversely proportional to its frequency across texts. TF informs us about the frequency with which a term appears in a document,

while IDF informs us about the relative rarity of a term in the collection of documents. By multiplying these values together we can get our final TF-IDF value.

$$Tf - Idf(t, d, D) = tf(t, d) \ . \ idf(t, d)$$

The higher the TF-IDF score the more important or relevant the term is; as a term gets less relevant, its TF-IDF score will approach 0.

6.2.3 Advantages of Tf-Idf

- TF-IDF can be a very handy metric for determining how important a term is in a document.
- TF-IDF also has use cases in the field of information retrieval, with one common example being search engines. Since TF-IDF can tell you about the relevant importance of a term based upon a document, a search engine can use TF-IDF to help rank search results based on relevance, with results which are more relevant to the user having higher TF-IDF scores.
- The biggest advantages of TF-IDF come from how simple and easy to use it is. It is simple to calculate, it is computationally cheap, and it is a simple starting point for similarity calculations (via TF-IDF vectorization + cosine similarity).
- Elastic search uses Tf-Idf for its inherent working of ranking each document or questions in our case to give top most results that has the given query keywords present inside it.

6.2.4 Disadvantages of Tf-Idf

- **Lack of semantic analysis:** Something to be aware of is that TF-IDF cannot help carry semantic meaning. It considers the importance of the words due to how it weighs them, but it cannot necessarily derive the contexts of the words and understand importance that way.
- **Memory Inefficient:** it can suffer from memory-inefficiency since TF-IDF can suffer from the curse of dimensionality.

6.3 Similarity Function

Elastic search allows you to configure a text scoring algorithm or similarity per field. The similarity setting provides a simple way of choosing a text similarity algorithm other than the default tf-idf, such as boolean,cosine similarity,bm-25,etc. Lets discuss the cosine similarity function a little more

1. Cosine similarity is a metric used to determine how similar the documents are irrespective of their size.Mathematically, Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space.
2. When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.

3. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word 'cricket' appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

4. Formula for cosine -similarity is:

$$cos\theta = \frac{\vec{a}.\vec{b}}{|\vec{a}|.|\vec{b}|} = \frac{\sum_i^n a_i b_i}{\sqrt{\sum_i^n a_i^2}\sqrt{\sum_i^n b_i^2}}$$

where, $\vec{a}.\vec{b} = \sum_i^n a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n$ is the dot product of the two vectors.

5. As you include more words from the document, it's harder to visualize a higher dimensional space. But you can directly compute the cosine similarity using this math formula.

7 Implementation

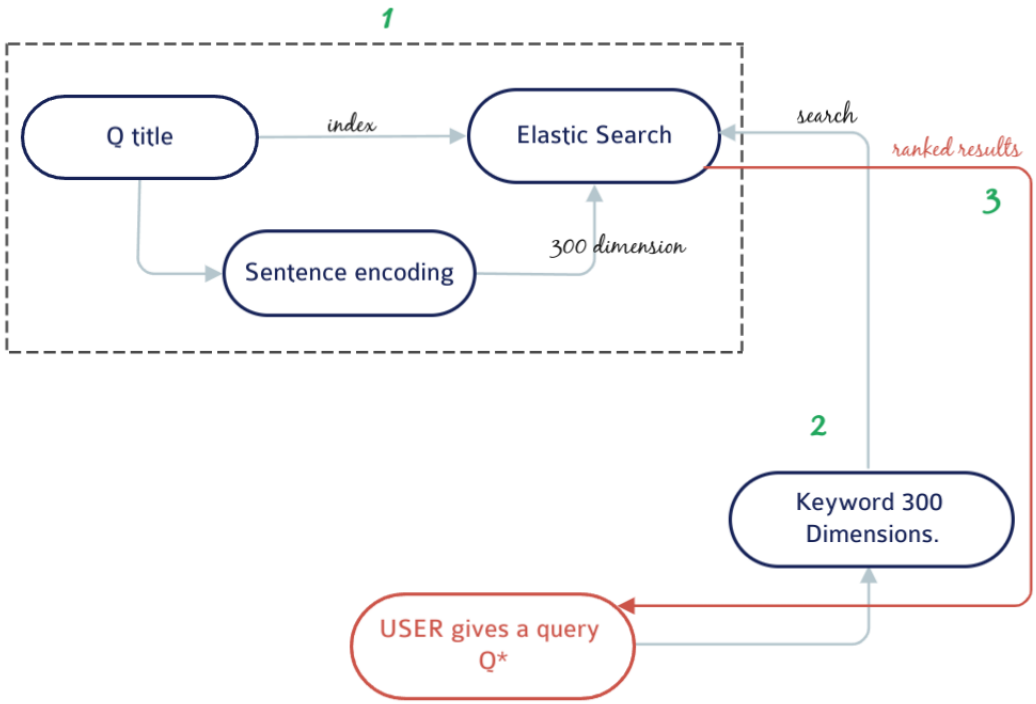

Figure 6: Architecture of Information Retrieval.

1. There are two functionality of Elastic search it can take both vector encoding and index as an input to generate query results

2. Now we have the vector encoding of all sentences. We capture query from the user and checks for spelling mistakes (as in 'tort':'sort'). If there is a spelling mistake,

we notify the user for the corrected version.After this,we feed the query to our IR system .

3. We convert the keywords of the query also into the encoding. And these encoding would be of 300bits and we ingest them into the Elastic search to find the top relevant questions.

4. Simultaneously the result of these encodings are amplified by the index results,Index results are questions converted into indexes that fed into the Elastic search and query also get indexed to show Its ranked results of the questions which are relevant most.Elastic search aides the Tf-Idf Information retrieval model for both the task.

5. Now we refine our ranked list(ordered list of question) by combining both encodings list of questions and indexed based list of question ranked results.

8 Results

- Top 10, Top50, Top100 and MAX: 500 number of queries can be processed by the IR system we developed.
- Average time taken by a query ranges between 0.005 to 0.020 seconds.
- The system has high recall and precision rate.
- The system is easily scalable and can also work efficiently with larger dataset.
- The IR system is easily deployable on website using streamlit.
- The IR system is memory friendly, i.e. the computational cost is low as Elastic Search has been used for inverted index.
- User friendly UI has been designed to facilitate easier search overheads for the user.

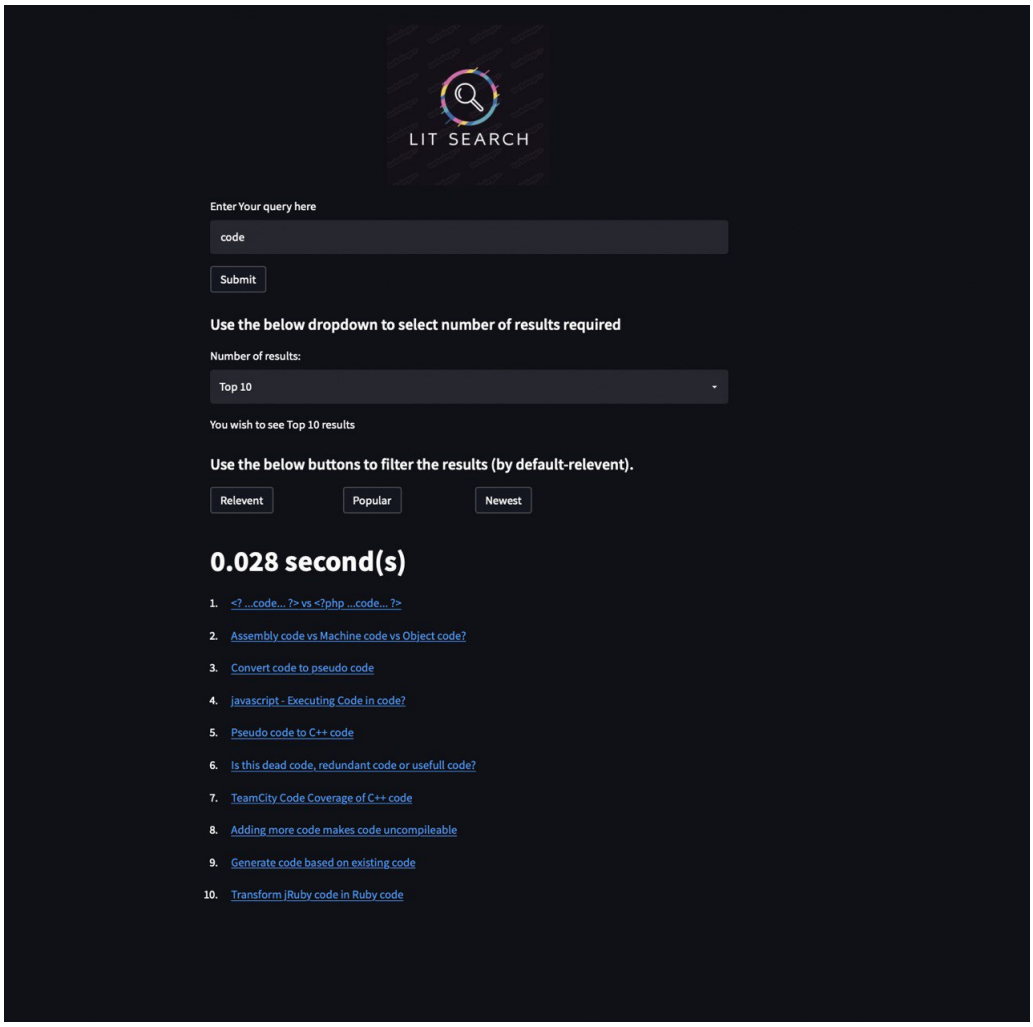

Figure 7: Search result for query 'code', according to filter 'Relevent'.

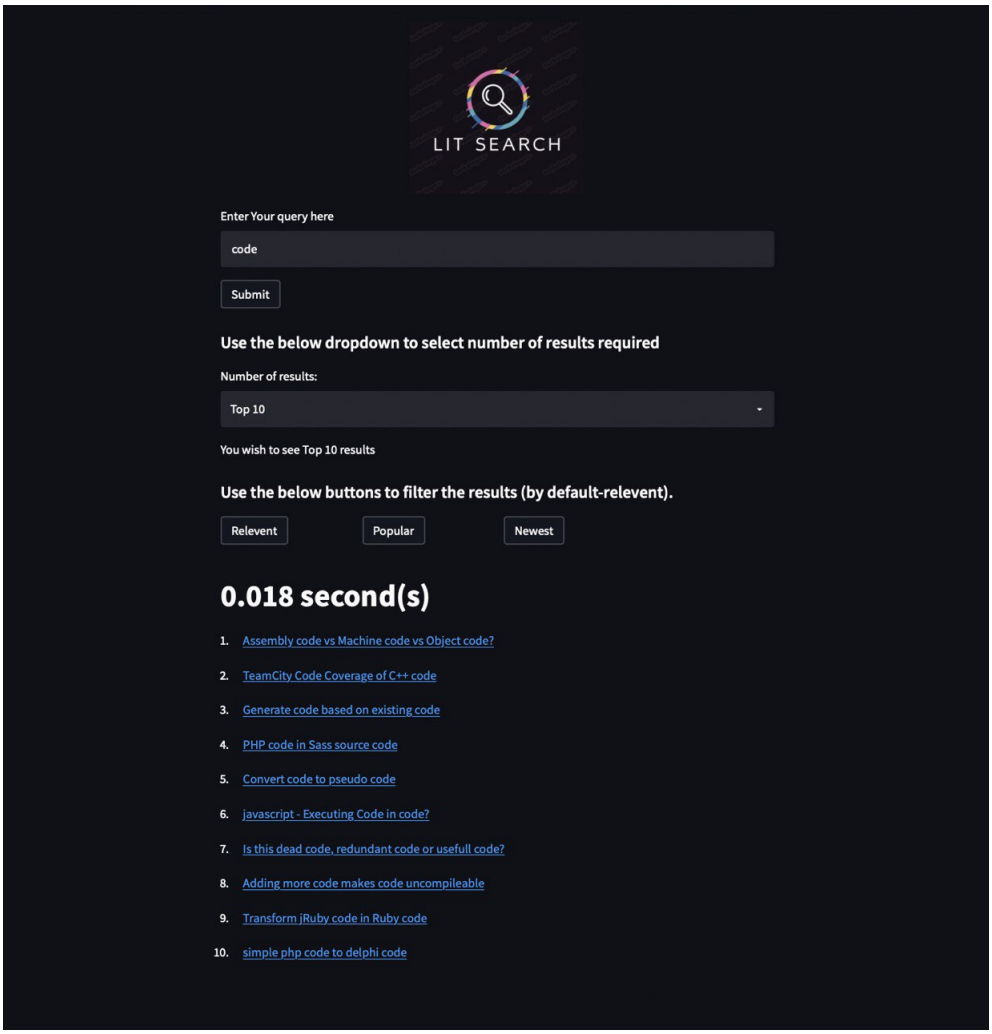

Figure 8: Search result for query 'code', according to filter 'Popular'.

9 Conclusion

We were finally able to retrieve relevant results from stack overflow data using the IR system we developed using Elastic search. The developed system requires lower computational resources and precise results were obtained with a lower latency.

10 References

1. StackSample: 10% of Stack Overflow QA | Kaggle
2. Fast Sentence Embeddings fse — A Python library for Ultra Fast Sentence Embeddings - Praveen Govindaraj - Medium
3. Understanding TF-ID: A Simple Introduction
4. Understanding TF-IDF for Machine Learning | Capital One