Biblioteca: A Mulitmodal Sentiment Frame Search Engine

Pranav N Venkit
pnv5011@psu.edu
Pennsylvania State University
University Park, Pennsylvania

ABSTRACT

This research focuses on developing a vertical search engine for a sentiment frame corpus, generated from parsing through books available in Google Books Repository. This custom search engine, build using elastic search, is then evaluated for the value of Precision and is compared with a Custom Google Search Engine to gain greater insight of the performance of the same. The cardinal feature exhibited by the vertical search engine will be the double search index that can be used to independently search for both text and sentiment frame of a given book.

KEYWORDS

Vertical Search Engine, Sentiment Frames, Elastic Search, Natural Language Processing, Information Retrieval

1 INTRODUCTION

A web search engine or Internet search engine is a software system that is designed to carry out web search, which means to search the World Wide Web in a systematic way for particular information specified in a textual web search query. A search engine can be categorized into two types based on the methodology of searching, i.e. Vertical and Horizontal. A Horizontal search engine is used to crawl or search through the web to acquire results based on a topic and topics related to the specified keyword, through the implementation of a breadth wide search. A public search engine such as Google or Bing can be categorized as a horizontal search engine. But for our problem statement here, we will be focusing more on a vertical search engine. A vertical search engine is used to search for a topic in detail, through the implementation of a depth wide search. An Enterprise Search Engine can be categorized as a vertical search engine due to its focus of a given topic. Hence, for our problem statement of generating and finding semantic tokens, in a specific repository of books, executing a custom vertical search engine will prove to be more effective in obtaining a greater precision because of the focused solution required to implement the task.

For our problem statement, we will be focusing on crawling through a set of thirty online books, for the greater goal of semantic story plot prediction. Hence the project will not just focus on crawling through the given book set, but also generating and indexing semantic frames from each sentence in the book. With the expected combination of searching through text and semantic frame, the search engine is intended to help complement the study of predicting the flow of a storyline in a given book. For this purpose, the custom search engine, Biblioteca, is expected to have a parallel search which lets the user search through both texts or characters, and the required semantic frame in question.

Before we understand more of the functionality of Biblioteca, we shall look at the definition of a semantic frame with respect to this project. A semantic frame can be thought of as a conceptual structure describing an event, relation, or object and the participants in it. A frame therefore is a schematic representation of a situation involving various participants, props, and other conceptual roles. Examples of frame names are Being_born and Locative_relation.

Once the frames have been generated and indexed, Biblioteca is assigned to crawl through the whole corpus of books and semantic frames, with the implementation of Elasticsearch and Kibana. Finally, using the integration of Django [3], we will be creating a double frame search engine to implement the solution. The report will also cover the efficiency testing of Biblioteca by analysing the top 10 retrieved results with respect to various formats of queries. Finally, we will also illustrate how our search engine implements the problem statement in a more holistic strategy as compared to the Google Custom Search engine created for the same task.

2 MOTIVATION

From the perspective of Natural Language Processing, it is always interesting to automate and create a kernel that is able to understand the linguistic meaning of a given language. With the ideation created by this field, it is also engrossing and revolutionary to understand the greater scope of narration. The objective is executed by creating a machine learning algorithm able enough to understand the generation of a story plot. This is the underlying motivation of this research endeavour.

With the ability to create and understand the semantic frame of a sentence, it is required to have a search engine developed that can complement the research by finding each

sentiment or plot related to a given sentence and also finding all possible sentences exhibiting a certain event or emotion. The customer for this project is Chieh-Yang Huang, a PhD student, in the Crowd-AI Lab of the College of IST in Pennsylvania State University. The secondary customer is Dr. Kenneth Huang, Assistant Professor of the College of IST. The intention and the problem statement of this project, that is worked on by the customers, is to gain a greater understanding of story plot generation through the help of a double frame search engine. The greater scope of the same is to predict and analyse the overall plot of a given genre by the data-points provided by the search engine.

The current work requirement stated by the customer is to find and identify the various events and emotions contained in a given plot. Hence for this we need to extract the sentiment frame constituted in every single sentence. The following and the pivotal goal will be to create a vertical search engine and a custom google search engine that allows a user to search for each sentence and sentiment frame, to understand and identify the plot of a given text. The larger objective, as stated above, will be to be able to predict the most possible following plot with a given sentiment frame. With the help of a suitable search engine, it will be easier to visualise and understand the content of a given story. This will result in better analysis of the prediction and the natural language classifier that is intended to be created by the customer at the Crowd-AI Lab. To develop an effective solution, a custom Google search engine and a enterprise search engine, created through Elasticsearch and Django. will be compared with respect to accuracy and performance. The final result is expected to be a search engine capable enough to index and provide effective result that denote search capability for both a sentence and a sentiment frame.

3 DATASET PREPROCESSING

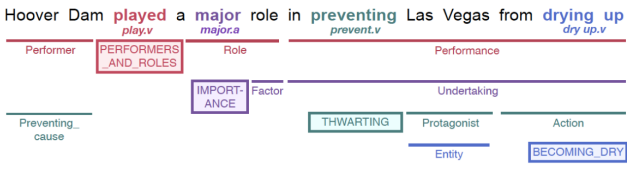

Figure 1: An example of a frame-semantic parse, with respect to a sentence [8]

The dataset that is primarily used in this project will be novels and textbooks obtained from a google book repository. The total number of books required to be indexed and crawled is selected to be 30. These books constituted various genres of literature, from comedy, fiction and research. The total size of this repository is $\tilde{3}$GB, averaging to $\tilde{5}$00MB per book. This complete repository was parsed through a JSON line format, consisting of one datapoint for each line in the book. Our main document, that is defined for this search engine, is a sentence and its constituent sentiment pair. Therefore the total number of documents indexed and crawled by the search engine is $\tilde{2}$50,000 documents. The flow chart for this preprocessing is well illustrated in Figure 5.

Before the sentence representative of each line was obtained, the data had to cleaned and sorted to avoid noise. The text was broken down to individual lines by separating them with respect to the "period" delimiter. The individual line was then stored as a single implementation unit in JSON. To execute semantic parsing, generated frames with the aid of FrameNet[1] methodology. A frame in FrameNet contains a textual description of what it represents (a frame definition), associated frame elements, lexical units, example sentences, and frame-to-frame relations. The execution of this methodology was done with the aid of the open_SESAME NLP repository [8]. Open_SESAME is a frame-semantic parser for automatically detecting FrameNet frames and their frame-elements from sentences. The model is based on softmax-margin segmental recurrent neural nets, described in our paper Frame-Semantic Parsing with Softmax-Margin Segmental RNNs and a Syntactic Scaffold. The example of a semantic frame with respect to a sentence is shown in Figure 1. The final output retrieved with the execution of Open_SESAME was the final JSON format consisting of both the line and its respective semantic frames. This was the fundamental format of the dataset that was used to index and create the search engine solution.

Finally, the indexing of the search engine, performed through Elasticsearch, indexed both the document, i.e sentences and the all the sentiment frame extracted from the same. The basic structure of the indexed document is shown in Figure 2. From the figure, the conversion of the sentence is illustrated. The final indexed document consisted of the following variable: "name" held the name of the book that the sentence was parsed from; "Frame" consisted of all the sentiment unit we are interested in searching for; "text" contained the complete sentence, which is to be made searchable as well; and finally "index" contained the sentence number in that given document. With the help of all these variables, we were able to create a searchable index that can be used to generate the required data to the customer, for future analysis of possible plot generation.

4 COMPETITION

The potential competition for this search engine doesn't technically exist as the data in concern are semantic frames and individual sentences retrieved from a collection of books. The enterprise search engine that is created concerns just the implementation and retrieval of information that is in review

```
{
  "_index" : "books",
  "_type" : "doc",
  "_id" : "66",
  "_score" : 1.0,
  "_source" : {
    "index" : 66,
    "text" : "Would you care for some tea.",
    "frame" : [
      {
        "LU" : "care.v",
        "Frame" : "Desiring"
      },
      {
        "LU" : "for.prep",
        "Frame" : "Taking sides"
      },
      {
        "LU" : "some.art",
        "Frame" : "Proportional quantity"
      },
      {
        "LU" : "tea.n",
        "Frame" : "Food"
      }
    ],
    "Name" : "Book 0"
  }
},
```


Figure 2: Data Structure of an indexed document

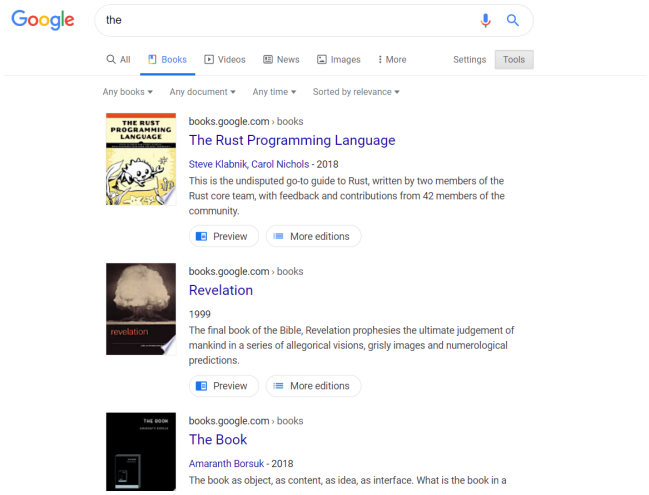

Figure 3: Google Books UI

to emotions, events and actions retrieved from a sentence. Hence this approach is very novel in terms of information retrieval and search engine development.

The possible or the most closely available public search engines are those developed to search and provide results pertaining to a library task. The basic UI of this search engine is shown in Figure 3. Such search engines are used to retrieve a book based on the textual input provided, which is related to the name or title of the book. Google Books search engine and Internet Archive- Text Archive are the most popular and widely used retrieval system with respect to books and textual search. We hence will be mimicking such a search engine by creating a Google custom search engine for the same. But the larger objective still remains unresolved as none of these systems actually retrieve or even handle sentiment and event based queries with respect to a collection of texts.

Few prominent secondary competitors for this problem statement can also be commercially implemented search engines [7] such as Amazon and EBay which focus more on both books and retail searches. These search engine also imitate enterprise search engines such as Google Books, but also have additional features such as advanced multi-faceted search that enable to search for various mercantile features such as price and type of book.

5 EXPERIMENT IMPLEMENTATION

As expounded in the previous section, this venture of creating 'Biblioteca', a semantic based book and textual search engine, is novel due to the unique dataset in concern. Hence to execute the same, we had to implement semantic frame parsing, to generate the expected data format for indexing. This step was illustrated in our discussion in "Data Preprocessing". Post the complete execution of the cleaned JSON line sentences, to generate these required format of sentiment frame, we created our cardinal dataset that was required to be indexed through a Lucene index [5], executed by Elasticsearch. We also intended to generate a possible Google Custom Search Engine to implement an alternate solution to the problem statement. The comparison and development of the two search engine will be our preeminent focus in the solution of concern. Both the implemented methodologies are discussed in the following subsection. Finally, we will be able to acquire a more holistic view of this research by comparing and understanding, in detail, the functionality of these two vertical search engines.

Google Custom Search Engine

Google Custom Search is a platform provided by Google that allows web developers to feature specialized information in web searches, refine and categorize queries and create customized search engines, based on Google Search [2]. Hence the formulation of a search engine based on GCSE (Google Custom Search Engine) will be related to the links that have been placed, for crawling the said pages.

The issue with using the GCSE is related to the fact that it operates only with links and not other forms of crawlable data. The input for our problem statement is expected to be the parsed lines and its corresponding dataframes. This cannot be placed in as the input of the GCSE. Hence, this solution seems to be unfeasible for our problem statement.

But to understand the functionality of such a search engine better, we tried to replicate a book search engine, which

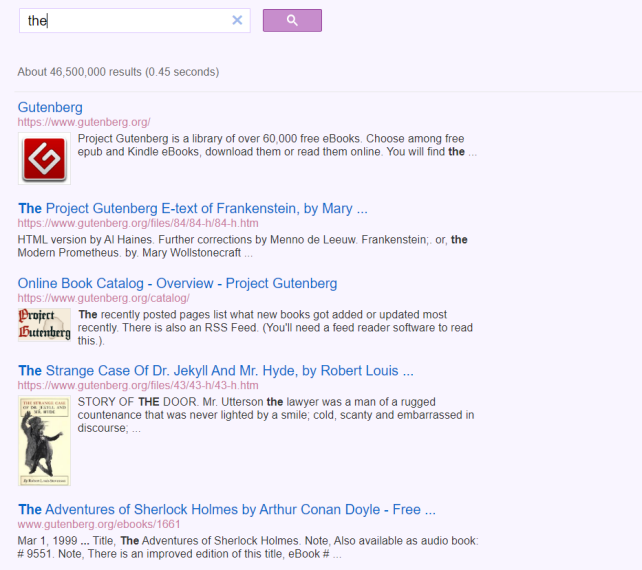

Figure 4: Screenshot of the Google Custom Search Engine developed for this project

searches for all books based on the title and name of the same. We incorporated links found from the Google Books, Internet Archive, Gutenberg, and other public library repository. This developed search engine was then able to retrieve books and books related to the searched query, with a dataset that is effectively a colossal database. The URL to the search engine is: https://cse.google.com/cse?cx=016399572544277902795:c23siiye906. The execution of the search engine is shown in Figure 4. From this screenshot, we can view the tota number of documents indexed to be ~46,500,000 documents. The query "the" provides almost all the possible books that have been parsed by the search engine. And from the top 5 results, we can see the importance of a given document with respect to PageRank [6] and the query provided. The most visited sites are called first, i.e https://www.gutenberg.org/, followed by other most visited sites with the query "the".

Custom Enterprise Search Engine

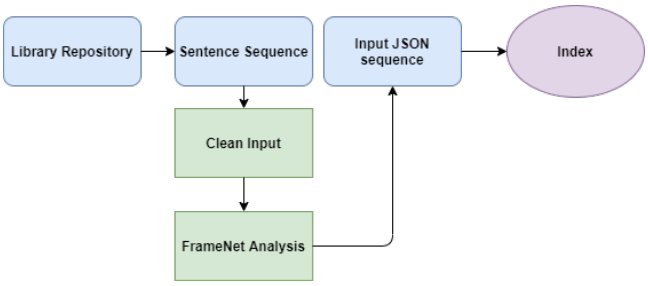

Figure 5: Preprocessing and creation of Index file

We now understand that using a Google Custom Search Engine does not solve the problem statement that we are trying to solve. We need a custom search engine that can parse through a specified dataset and not the URL of various website. In a GCSE, the engine was able to only parse through content and entities in the internet space, i.e. URLs and hyperlinks. Therefore the solution ceased to be feasible.

For the vertical search engine, we intend to create an index of documents consisting of both sentences and sentiment frames. To execute this, as explained before, we parse the book through our preprocessing step to acquire a JSON line document consisting of the required dataset. With the help of Elasticsearch, we index every individual document or a sentence. This will be the next step to creation of the search engine. Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch can be used to search all kinds of documents. Due to this salient feature of Elasticsearch, we use the same to index our semantic document. The indexed document format is shown in Figure. 2.

We use Elasticsearch as it implements indexing that results in efficient searching for documents with unstructured text using terms as queries. Our unstructured data is then made searchable by converting it to an index file. This index file is used to provide results, through a query, using an indexer, i.e. Elasticsearch. Hence, Elasticsearch also provides a function that retrieves the indexed file through the query passed from the User Interface. As our search engine will require a double search bar, each executing a different operation, we will be passing two individual query to Elasticsearch. The user interface for this search engine is built using Django [3]. Django is a Python-based free and open-source web framework, which follows the model-template-view architectural pattern. Using this framework, we can develop an interface and a database that can retrieve and exhibit the required results from the index files. The home page design of the search engine is shown in Figure 6. From this, we can visualize the double frame feature of the search engine, one for searching for particular semantic frame and the other for searching for texts and sentences present in the book.

The search engine implements additional features, with the help of the Django framework, for the ease of user interface. A few such characteristics are the pagination and highlight feature, that were included for the same reason. Only ten features are shown in a single page. The user is allowed to scrolled through the pages to see results in a group of 10 results. The top most favoured result appears top. This is how the pagination feature was executed. For the highlight feature, a highlight of the searched term appears in the result, to illustrate the relevance of the given search. The double bar search also eases multi search facility. A user can search,

Figure 6: Home page of Biblioteca: The Double frame search engine

Figure 7: 'Text' search of the keyword "the" in Biblioteca

in a multimodal fashion, by selecting the type of data that is required to be indexed. Both the results differ with respect to what the user searches for. The feature of multi-faceted or advanced search was implemented in the search engine, but as this feature did not prove to be very useful in ease of the solution, it was discarded. The total number of documents that is retrieved, at maximum, is 10,000 documents. This is the maximum threshold provided by Elasticsearch for its query function.

For text search, in Biblioteca, as shown in Figure 7, the keyword that was passed as test was "the". "The" is commonly use to retrieve the maximum number of documents that are indexed. Hence as shown in the result, 10,000 documents are retrieved. There are approximately 250,000 documents in the index but as Elasticsearch has a limit of just 10,000 words that can be retrieved for a query, the search engine shows only 10,000 documents. From the search result, it is obvious as to how the search engine shows the best case output. A sentence having the highest Cosine similarity [4] to the query is shown topmost. The first result has the greatest number of the keyword in its sentence, hence is shown to be the best result. We can also notice the highlight feature at work here. The keywords where the query is present, is highlighted in the sentence, to showcase as to where the queries appear. The results also show the book name as well as the sentence it appears in, for analytic purpose of the plot prediction project.

The sentiment frame search feature of the search engine is the salient entity of this endeavour. This is illustrated in Figure 8 where the sentiment search bar is used to search for a keyword. To search for a sentiment of an action, the user must be well versed with the entity being searched. There are many attributes that have been generated through the FrameNet architecture. A few examples are: "Kinship", "Being Named", "Being Operational", etc. where each frame denotes either an action, sentiment or an event. From the search result we can identify that the topmost result was chosen due to the greater occurrence of that sentiment frame in that sentence, as compared to other sentences in the result. The result show the required data required for understanding relevant details of the sentiment frame. The necessary data such as the book name and the sentence of concern is shown in the user interface. This result is allowed to be downloaded by the user as well.

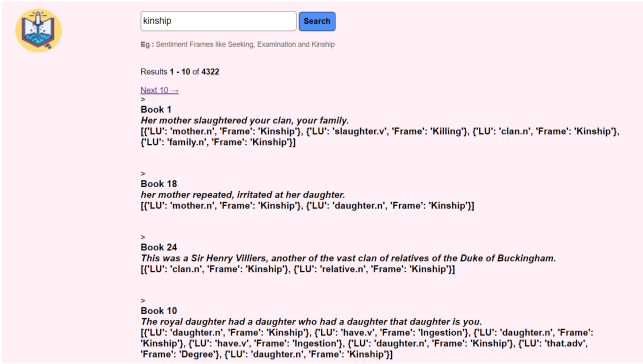

Figure 8: 'Semantic' search of the keyword "kinship" in Biblioteca

Hence, with the above features and characteristics, the required multimodal search engine was created for identifying a semantic frame and the sentence present in the book. Through the retrieved data, we will be able to define and understand the flow of a plot with respect to a book and the various ways of implementing not just a single search feature but a multi search feature, with the inclusion of two search bars, representing different queries.

The Indexer, Crawler, Query Engine and the Interface were all built in a local machine and not the IST441 server, provided in the class. Due to this, the operation and execution

were more flexible to change as it did not follow the template provided in the server.

6 QUERY ANALYSIS

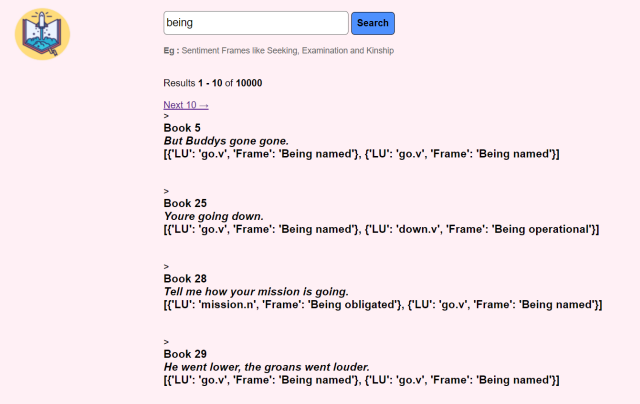

Figure 9: 'Semantic' search of the keyword "being" in Biblioteca

To understand the implementation of a query in our search engine, we first will understand more about how Elasticsearch implements ranking. Post this, we will execute various different queries to understand the relevance of each. Finally we will compare the Google Custom Search Engine, with other public search engine, to see the effectiveness of the same. As Biblioteca is an individual and novel venture in the field of information retrieval, it will be difficult to compare it with other commercially available search engine. We hence will try to understand and compare the GCSE with another search engine to understand more of the ranking features.

As mentioned before, the index of this search engine is created using Elasticsearch. Therefore, the ranking and query scoring is performed by the libraries present in the same. Elasticsearch runs Lucene under the hood so by default it uses Lucene's Practical Scoring Function. This is a similarity model based on Term Frequency (tf) and Inverse Document Frequency (idf) that also uses the Vector Space Model (vsm) for multi-term queries. Hence the results provided will be considered with respect to not just the frequency of that term but also the closeness of the document for the given query. This is illustrated will in Figure 7 and Figure 8. To illustrate the working of Lucene's Practical Scoring Function, we can see this well demonstrated in the search of the word "being" in the semantic search bar. The Figure 9 shows how the ranking works. The results obtained show frames having the word "being" in them, such as "Being Named", "Being Operational", "Being Obligated". The result is arranged with respect to the books indexed, as all the top 10 queries have

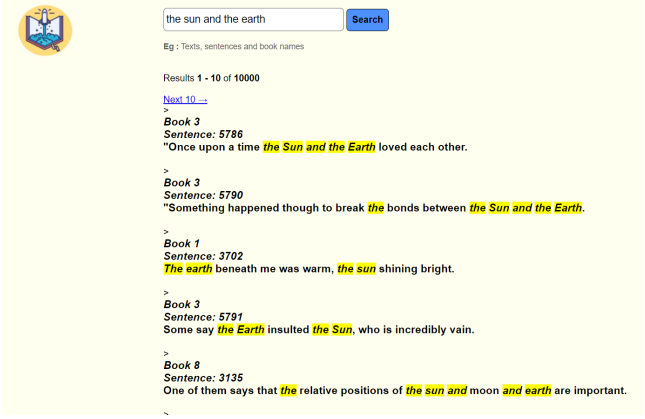

Figure 10: 'Text' search of the keyword "the sun and the earth" in Biblioteca

the same ranking. We can understand more of the scoring functionality by trying to run a multi-keyword query to see the performance of the search engine. This is shown in Figure 10. The query passed was "the sun and the earth". From the top results, we can see how sentences with lesser words are given more preference due to the calculation of tfidf. The keyword of "sun" and "earth" are given more weightage as compared to other simple stopwords such as "the" and "and". This greatly elaborates how Elasticsearch retrieves its queries from the indexed file. To explore more into the performance of Biblioteca, we will be passing 10 queries each, to sentiment and text frame to understand the performance of the search engine, through precision calculation.

The above table 1 shows the precision performance of various query with respect to the sentence search feature performed by the search engine. Precision is calculated on the basis of the ratio between the relevant pages that were retrieved to the total number of retrieved pages. For both the precision test, we take in consideration of the top 10 queries that is generated from Biblioteca, i.e. the first page of the result is considered. As Biblioteca searches through one sentence at a time, it is easier to compute and retrieve the result. Hence, the results obtained will be direct with respect to the number of documents available to represent that term. The reduce in value of precision is related to documents not having that given term. Most of the keywords passed, to test the precision, is related to emotions or events concerning a plot. The results retrieved were well related or synonymous to the search.

Table 2 shows the performance of Biblioteca with respect to a sentiment frame. You can see how all the keywords provided result leading to an accuracy of 1. This is because we search for specific keywords present in the index for a sentiment frame. This operation therefore concludes itself

Table 1: Precision calculation of Biblioteca: Text

Query	Search	Precision (@10)
Near death experience	Text	1
Once upon a time	Text	0.9
sadness	Text	1
Batman	Text	0.7
happy marriage	Text	0.5
Heart break	Text	0.8
fear and horror	Text	0.9
And then there were none	Text	0.9
Happy birthday	Text	1
murder weapon	Text	1

Table 2: Precision calculation of Biblioteca: Sentiment Frame

Query	Search	Precision (@10)
Killing	Sentiment Frame	1.00
Being named	Sentiment Frame	1.00
Kinship	Sentiment Frame	1.00
Goal	Sentiment Frame	1.00
Desirability	Sentiment Frame	1.00
Building	Sentiment Frame	1.00
Fear	Sentiment Frame	1.00
Sensation	Sentiment Frame	1.00
Death	Sentiment Frame	1.00
Taking time	Sentiment Frame	1.00

as a simple binary query search as there aren't many other keywords to compare the queries significance with. If the query is present in the index, it is extracted. If not, it is deemed as incorrect and hence is not taken. The rank of the result is determined by the Boolean value of that query with the statement and the frequency of the mentioned frame in that given statement.

We cannot compare the performance of this search engine with another potentially available search as this is a novel approach in sense of information retrieval. We therefore look at the Google custom search engine that we created and the public Google Book search engine, to get a greater perspective of the PageRank feature exhibited by the search engine. The custom search engine, like explained in the previous section, is meant to search for books with respect to its title. It contained a large repository of various public library websites.

From the table 3 we can understand the specific performance of the custom search engine as compared to the public search engine. The custom search engine performs really well with respect to specific book searches as it contains a greater

Table 3: Precision calculation of Google Custom Search Engine(GCSE) and Google Books(GB) search engine

Query	GCSE P(@10)	GB P(@10)
Once Upon a Time	1.0	0.9
1Q84	1.0	0.6
Chambers of secrets	0.9	0.9
Natural Language Processing	1.0	1.0
brief history of time	0.5	1.0
Catcher in the rye	0.8	0.9
horror	1.0	0.9
And then there were none	0.9	1.0
Theory of relativity	1.0	1.0
The horse the boy	0.4	0.6

collections of books from various well known public library sites. Its performance with respect to finding books from incomplete query and finding books based on laws or principles are also is good. But it doesn't perform exceptionally well when a genre is provided as its input. On the other hand, the Google Book search engine performs well with respect to a book search but doesn't out perform the custom search engine as it does not contain enough datapoints as compared to the same. But Google Books performs well when a genre is searched in specific. It provides a more holistic result as compared to the custom search engine.

7 FUTURE WORK

The search engine, Biblioteca, was created with the intention of having an information retrieval system that can parse and search through a data repository, in a multimodal fashion. The solution was intended to provide greater insights to the Natural Language Processing project of predicting a story plot. Hence the data intended to be parsed were story frames or semantic units that denoted the events and sentiments present in a given sentence. This was executed with the development of Biblioteca where the search engine enabled the search of both sentence, text and sentiments, in a separate method. This was well received by the client as well. For future work, with a more elaborate discussion, it was concluded that we can create a second version of this system by including a feature that helps run parallel query implementation. This will serve as a separate information retrieval system. With respect to story plot generation, it will be useful to pass through a pair of sentiment frame, to acquire an answer that mentions the pair of sentence that has the same sentiment pair, This will provide a more greater and holistic analysis to the problem statement. This feature is hence termed as our future work to this project.

8 CONCLUSION

From this research endeavour, we have now understood how a specific search engine operates. We have also created two different architecture of a search engine. The first is the Google custom search engine and the second is the vertical multimodal search engine, Biblioteca. The project also depicts, in detail, the data preprocessing, indexing and creation procedures of the vertical search engine. The salient feature of the project was its architecture, where Biblioteca implemented multimodal search feature by exhibiting two parallel search bar, each for a separate unique search operation. Finally, the performance of these search engines were explained in detail with the value of precision.

Finally, we also get to understand the comparison of creating a Google Custom Search Engine and an actual public search engine, Google Books. Through this comparison, we can elucidate how various engines perform differently for various usecases. This result explicitly demonstrates the ranking feature implemented by the Lucene Elasticsearch library, in both the enterprise search engine and the public google search engine.

9 APPENDICES

The Google custom search engine can be found in https://cse.google.com/cse?cx=016399572544277902795:c23siiye906 and the multimodal enterprise search engine code is present in https://github.com/PranavNV/Biblioteca. The source data and the crawled index can be found in the same link.

REFERENCES

[1] Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 86–90.

[2] Rajashree Gavali. 2015. Discovery service for engineering and technology literature through google custom search: A case study. *DESIDOC Journal of Library & Information Technology* 35, 6 (2015), 417–421.

[3] Adrian Holovaty and Jacob Kaplan-Moss. 2009. *The definitive guide to Django: Web development done right.* Apress.

[4] Baoli Li and Liping Han. 2013. Distance weighted cosine similarity measure for text classification. In *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 611–618.

[5] Michael McCandless, Erik Hatcher, Otis Gospodnetić, and O Gospodnetić. 2010. *Lucene in action*. Vol. 2. Manning Greenwich.

[6] Matteo Pasquinelli. 2009. Google's PageRank algorithm: A diagram of cognitive capitalism and the rentier of the common intellect. *Deep search: The politics of search beyond Google* (2009), 152–162.

[7] Amit Singhal. 2005. Challenges in running a commercial search engine. (2005).

[8] Swabhs. 2019. swabhs/open-sesame. https://github.com/swabhs/open-sesame