Fall Term 2021

An IR System based on Economic News using Bert Embeddings

ITC6008A1 – Search Engines and Web Mining

December 13, 2021

Michalis Koinakis | Dimitra Kouvara | Iris Kleopa

A black and white image of a person with a mustache

Description automatically generated with low confidence

Instructor: Dr. L. Polymenakos

**Contents**

[Abstract 3](#_Toc1206101399)

[Introduction 3](#_Toc11100972)

[Web scraping 5](#_Toc1413740986)

[Storing the data 8](#_Toc360671699)

[Choosing the most appropriate language model 10](#_Toc1112466380)

[Text preprocessing 11](#_Toc1571060471)

[Model Deployment and Optimization 12](#_Toc300749439)

[Evaluation 14](#_Toc134554158)

[Graphical User Interface 15](#_Toc1363796641)

[Future Implementation 18](#_Toc1035153450)

[Conclusion 19](#_Toc971055775)

# **Abstract**

Information Retrieval (IR) systems are widely used in attempt to obtain information relative to a query, from collections of various resources. There are numerous applications and implementations of such systems along with interesting research conducted in recent years. We present an IR system based on the recent breakthrough of Bert Embeddings. In this framework, Bert Embeddings are used to encode economic articles in 768 dense vector space, which serve as the basis of all computations and calculations conducted by the IR system.

Our approach is using crawlers to get the data from online economic sites and store them in a database. All articles are turned into Bert Embeddings and used to calculate similarities among them and queries, to extract common ground or context among them. The system offers access to the user through a Graphical User Interface (GUI), which presents the results collected from the IR ranked from the most relevant to the less relevant. Finally, instead of printing the whole article, the IR system provides summarized text output, created from another Bert Embeddings application, the text summarization.

To evaluate the results, an experiment of random words from random texts in the database was conducted, comparing whether the context of the random texts matches that of the extracted documents. Also, some user evaluation experiments were conducted, by assessing the results printed from a user perspective.

# **Introduction**

In recent decades, the traditional method of reading newspapers has shifted, and the number of online news websites has expanded. People no longer wait the distribution of the hardcopy of their favorite newspaper since easily access of up-to-dated news that allows readers to learn about what it is going on in the economic or financial markets by just visiting one of the many economic websites available today. The abundance of online resources giving updates, explanations and new important terminologies about business related news, finance and stocks enabled consumers to stay updated regarding the industry and learn increasingly every day. Following the Covid-19 pandemic, many people are now focusing on economic recovery, and this has produced a significant increase in the daily frequency of global English language economic news websites. The incapacity to comprehend, assimilate and use such a huge amount of information is becoming increasingly obvious as the number of internet news sources grows. Additionally, the development of search engines provides consumers with a huge amount of this information, delivered at a pace and convenience that few people could imagine some years ago.

The scope of this project is to create an application utilizing various Natural Language Processing techniques focusing on gathering economic and financial news, storing them in a dynamically changing database, providing a way for the computer to classify a text based on semantics and presenting the results to the user.

The rapid growth of numerous websites and browsers has also resulted in an increase in the massive amount of information available daily. As a result, web scrapers were developed with the purpose of extracting exceptionally large volumes of data from websites by using an automatic method. Most of this data are unstructured data in an HTML format, and it is converted to structured data in a database. Web scrapers can retrieve all the data on a website or just the information that a user is looking for. To extract the articles from the internet, two web scrapers were developed. Different web scraping approaches were explored and implemented to make the scrapers get data consistently, daily.

To store the data, a database was created, which can hold the articles, their creators, time and date of post, a summary, and a contextual label, grouping similar articles together. On top of this, a parallel JSON file is created, which relates to the database by sharing the same primary keys and is used to store the important to our approach Bert embeddings.

Optimizations regarding the speed and accuracy of the IR system are explored through text clustering and classification algorithms, which reduce the amount of data processed for any given user query.

Finally, to test the accuracy of the IR model, two experiments were conducted. One, which involved the users and their satisfaction over the returned results based on query context. The second, an automated test, checks the results returned from an embedded classification algorithm and how it relates with the expected context of the query entered.

# **Web scraping**

For this project, a web scrapper with two different targeted websites was implemented. Those websites are the following:

* The Market Beat which “empowers individual investors to make better trading decisions by providing real-time financial data and objective market analysis”.
* The Economic Times which publishes the latest Business news and updates on Finance, share market, IPO, economy. Discover Business News Headlines, Top Financial News and more on The Economic Times.

A brief description of the scrapper layout can be found in the below figure.

Diagram

Description automatically generated

Figure 1: Brief description of web scraping

When a web scraper needs to scrape a site, first the URLs are provided. Utilizing Python requests library, we get the HTML code for making HTTP requests. The library of requests provides two methods of HTTP, get and post. In this case, requests.get() is issued and, as a result, data is retrieved from the specified websites and stored in a Python object. The HTML content of the webpages can be parsed and scraped with Beautiful Soup, one of the most popular external libraries for parsing with Python. The aim is to parse the HTML content of a given URL and access its elements by identifying them with their tags and attributes. The graph, seen below, shows these steps:

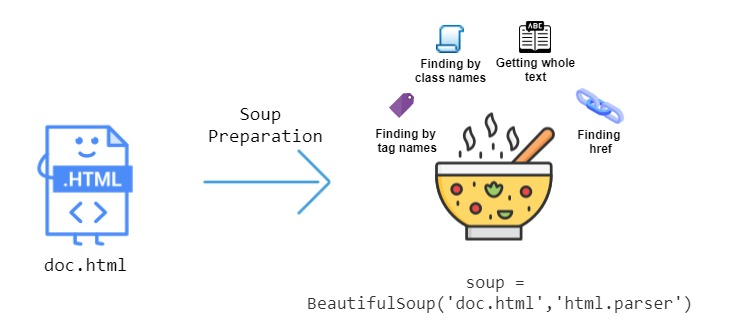


Figure 2: Beautiful Soup procedure (Gunasekaran,2020)

The way to find specific elements on the webpage is either by their ID or the HTML class. To find out which holds the information needed, web pages were inspected using the Developer Tools utility from the popular browsers. Finally, extracting text content from HTML elements is achieved by .text. In this case, the selected parts of the websites from the web scrapper were the article’s link, its title, the author, and the publication date. As the websites used their own pattern, the difficulty was regarding the different HTML structure of each of the websites.

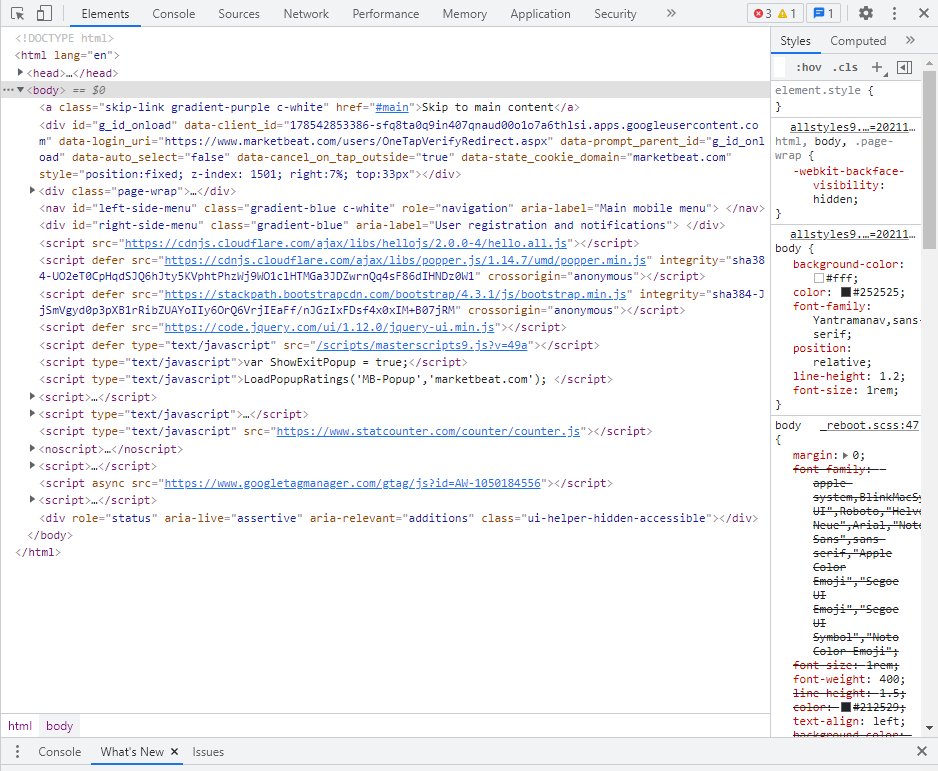


Figure 3: Structure of HTML page through Developer Tools

Next, the retrieved data was saved in JSON file format of the following structure:



Figure 4: Sample of the retrieved data in JSON format

Engaging with creating a web scraper has highlighted a vast variety of challenges:

* Dealing with legal limitations

Legal limitations are a major issue that should be considered before applying web scraping, as it is not permitted to scrape confidential information for profit (Roberts, 2020).

* Selecting the sources

Another widespread problem in the appropriate selection of news websites is the paywalls among news websites. The subscription to very well-known sites, such as The Wall Street Journal, The New York Times and the Financial Times is mandatory to have full access to the daily news. The selection of the two websites used for scraping was based on non-subscription criteria.

* Infinite scroll down sites

A problem to overcome, gathering all desired articles from the second site, was a list container, holding the URLs, which was refreshed with scroll-down. The more someone scrolled down, the more articles were becoming available. To overcome this issue, a web driver API, from selenium module was used, to navigate the site to the moving refreshing point. It was found out through observation and experimentation that the spot where someone needs to scroll to refresh the list, is moving linearly and can be found using the following formula: .

As web scrapers need to regularly visit the target websites to collect the data, many requests are sent from a single IP address. In the case where the website has strict regulations on scraping, certain IP could be blocked. To overcome this issue, web scraping tools are utilized:

* Adding Proxies

Proxies are an important part of web scraping projects. Adding proxies to the scraping software offers several benefits. When making a HTTP request to a site using a proxy server, instead of travelling directly to that site, the request first passes through the proxy server, and then on to the target site. Thus, the proxy server is making the request on the programmer’s behalf (“by proxy”) and then passing the response from the target site back to him.

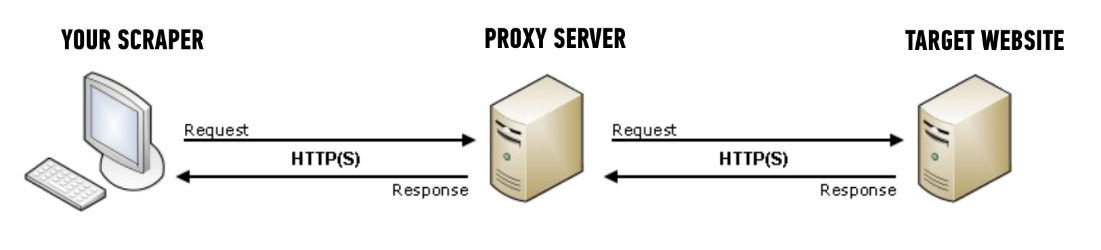


Figure 5: Description of proxies’ utility as an intermediate mean between the scraper and the target website (Hartley, 2018)

The main benefits of utilizing proxies on a web scraping project are hiding the programmer’s source machine’s IP address and getting past rate limits on the target site.

* Spoofing user agents

Spoofing user agents is essential when scraping data on websites, as web servers can identify browsers, web scrapers, download managers, spambots, etc. based on their unique user-agent strings and ban a working IP. By using fake user-agent strings, that give them legitimate identities belonging to popular browsers, web scrapers can scrape data successfully from antibot websites.

* Code delaying

Code delaying is helpful in case where many requests are fired in rapid succession on a server, as they can take up all free connections and effectively become a DoS Attack. Limiting the number of requests per certain - random time is possible. It is considered an effective way for not putting strain on a site’s server and abusing the server’s tolerance.

# **Storing the data**

Websites are great at sharing data with anyone that views their pages in a browser. Web browsers read the HTML code at a URL and display content embedded in the HTML. The creation of a database, where the retrieved articles are stored, is essential for the search engine. Since the data is relational, SQL server is an ideal choice. As the web scraping script is executed daily, the newspaper database is being updated regularly.

Following a small scale of articles’ table is presented:

A screenshot of a computer

Description automatically generated with low confidence

Figure 6: Sample of articles table in the SQL server

The information that has been retrieved by the online newspapers are the attributes: link, title, author, publisher, date, article’s text, summary, and contextual label, which corresponds to a topic/category of articles. The attributes’ type is predefined, since SQL belongs to relational databases, see Figure 7.

A picture containing table

Description automatically generated

Figure 7: Types of attributes

Link, title, author, publisher, article and summary are of string data type. Note that the type of article and summary is noted as text due to the excessive need of bytes. The difference between varchar(size) and text is the maximum length of bytes the string holds. Strings that hold a maximum length of 65535 bytes seem to satisfy the needs of the project so far, which is why text has been selected. The label, representing the number of different article’s topics (clusters), is of type small integer, holding up to 2 bytes.

In terms of data maintenance, a routine that frees up disk space is implemented: rejecting duplicated and deleting near-duplicated documents. The absence of duplicated documents is satisfied, as the articles’ links are set as Primary keys. Regarding near-duplicated documents, their selection is based on documents’ similarity, which is derived from the cosine similarity of the respective Bert embeddings. With the term near-duplicated, we refer to the articles that have similar, but not necessarily identical, content. In the case where their similarity exceeds a certain threshold (cosine similarity > 0.9 was selected), the compared documents are considered near duplicated, and so one of them is removed.

# **Choosing the most appropriate language model**

Natural language processing (NLP) techniques aim to intelligently analyze documents and capture their meaning, and thus providing a deeper analysis of text than some of the approaches which are currently used in IR (Donglas.2021).

The first model with which we experimented was the Boolean model, though due to its many disadvantages it was rejected. In the Boolean model the document is represented as a set of keywords and the queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate scope. This characteristic was restrictive, as it raised a difficulty on expressing complex user requests. Since the model’s similarity function is Boolean and as such the Boolean logic gives only definite, black-or-white results, there would be no partial matches. Furthermore, to compare the retrieved documents, computations with large vectors would be needed which make the model computationally expensive. Finally, the model would consider as similar documents only those that contain the exact words of the query, which would lead document ranking problems. These are a few reasons that led to the need to find a more appropriate model.

Following, another popular model was explored: word2Vec vector space model. However, its limitations, which follow, were proven significant and led to its rejection. The word2Vec model generates embeddings that are context-independent, meaning that there is just one vector representation for each word without any semantic meaning considered. Moreover, word2Vec embeddings do not to consider the word position. Since they are available to use directly off-the-shelf, meaning that the model’s input is a single word and the output a vector representation of that word, is dependent to the pretrained data. This is rather limiting as the context, or the surrounding words are not considered before generating the word vector. Finally, word2Vec model does not support out-of-vocabulary (OOV) words — which is one of its major disadvantages, meaning that the word representations cannot generate vectors for words encountered outside the vocabulary space (Gupta, 2021).

Vector Space model also rejected as on one side is a simple model and weights are not binary, however for long documents it is a poor model as the similarity values are low and in the case of articles it is not helpful as the number of articles is large and vector space is limited for this use. Also, vector space model is not useful for articles due to semantic sensitivity, as the search of keywords must be exact matches to the document terms, else similar documents will not be associated. This may lead to dysfunctionality of the use of keywords from the user. Moreover, the order in which the terms appear is lost, since the model is based on a Bag of Words approach.

Bert model, on the other hand, is one of the latest milestones; a model that marked the beginning of a new era in NLP. The reasons behind Bert’s success, which also led to its selection for the project, are the following:

* It is pre-trained on an absurd amount of data.
* It is open source.
* It explicitly takes as input the position (index) of each word in the sentence before calculating its embedding.
* It accounts for a word’s context considering the semantics of words, returning different vectors for the same word depending on the words around it.
* It can generate a vector representation for any arbitrary word and is not limited to the vocabulary space. It is essentially an infinite vocabulary, as it provides support for out-of-vocabulary (OOV) words (Wning,2018).

# **Text preprocessing**

Text preprocessing is an important step for the following natural language tasks, as it transforms the text into a more digestible form so that machine learning algorithms can perform better. However, preprocessing procedure is crucial mostly for a very narrow domain such as Tweets for a specific subject. These types of data are noisy and many steps of preprocessing such as lemmatization or stemming are necessary. As articles are – most of the time – well written texts, preprocessing is not so important. For this reason, five main components of text preprocessing performed in this project:

|  |  |
| --- | --- |
| Removing HTML tags | Since the documents were retrieved by web scraping, chances are that they contain HTML tags. However, as tags do not contribute to any NLP task, they should be removed. Only the plain text is left. |
| Removing punctuation |  |
| Expanding contractions |  |
| Lowercase |  |
| Removing numbers | Numbers might not add significant information to text processing. |
| Bert tokenizer | BERT was trained using the WordPiece tokenization. It means that a word can be broken down into more than one sub-word. |

Table 1: Steps of text preprocessing - Preparing the articles for BERT

The model used for Bert Tokenizer, is derived from the sentence\_transformers API, and specifically the sentence-transformers/paraphrase-distilroberta-base-v1. This framework generates embeddings for each input sentence or text. It maps sentences and paragraphs to 768-dimensional dense vector space and can be used for tasks like clustering or semantic search (Reimers & Gurevych, 2019).

It is a distilled version of the RoBERTa-base model, and it follows the same training procedure as DistilBert. It is twice as fast as Roberta-base and is created on 82 million parameters.

All Bert embeddings created, are stored in a JSON file, where the link of the article is set to be the key and the respective embedding the value, for any given pair. Like this, there is an embedding for every article stored in the database, which reduces computation expenses during the IR deployment.

The exact same steps are applied to the user input query and the embedding is created in real time.

Other Bert models that were used were Bert-base-uncased model and the longformer-base-4096, which is pretrained for long sequences of length up to 4.096.

# **Model Deployment and Optimization**

The process of the IR system was deployed primarily using the following sequence of actions. The user enters input, which is cleaned and then transformed to Bert Embedding. The outcome of two previous steps is compared with similar Bert Embedding representations which are saved in a database. Those articles with higher cosine similarity to the query are finally ranked and printed to the user.

That method was simple and effective, though in a constantly growing and dynamically changing database, there were delays because the program had to compute thousands of similarities. Also, memory capacity needs, become bigger every time the database is loaded in the IR system. To save time and memory, two more steps were inserted in the preprocessing part.

Since the articles are unlabeled, it was decided to use clustering techniques in attempt to create meaningful categories of documents. Since Bert Embeddings are 768 dimensional dense vectors, they are given as input to a K-Means clustering algorithm. In the graph below is shown a representation of the data in space, using dimensionality reduction from UMAP. As it may be observed, there is group of articles on the top right corner. This is giving the intuition that the articles gathered have some common context. And there are three groups of data, which are outlier’s context wise, and will produce some groups of data with less similarity to the other groups.

Chart, scatter chart

Description automatically generated

Figure 8: Graphical representation of the data in space after dimensionality reduction - Groups of data and Outliers

The clusters were chosen with the elbow method. In the diagram below, it is presented graphically that the cut-point where the biggest elbow is observed is at 18 clusters. Similar points can be found around 11, 24, 35, and 43 clusters, though the choice was done at the point where the convex of elbow was the largest. After the clustering, a label is passed in the database which is from now on the contextual group of the articles.

Chart, line chart

Description automatically generated

Figure 9: Selection of clusters based on the elbow method

Since there are now labels, a classifier may be trained and used to choose which are the documents of interest for the respective query.

To train the classifier the labels from the database and the Bert Embeddings are given as input to the models, with 90% train – test sets splitting.

The choice of classifiers was SVM classifier and Logistic Regression classifier. On the table below, there are the F1-scores, from the test sets, from both the classifiers. Logistic Regression although produced a slightly better average F1-score – 0.939 against the SVM F1-score – 0.936, so that was the model which was embedded in the IR process to predict the query context.

Text

Description automatically generatedText

Description automatically generated

Figure 10: Classifiers F1-scores. Top – SVM, Bottom – Logistic Regression

A final note on the choice of classifiers, Logistic Regression provided good results for all the experiments done with 2.000, 4.000, 6.000, and finally 8.000 documents, while the SVM provided better results with more than 6.000 documents. In the future, the choice of the classifying model will be re-assessed.

Finally, the classifier is embedded in the IR system. It “predicts” the contextual group of the query, and the system retrieves only the selected articles. The cosine similarities calculated are reduced to hundreds rather than thousands, giving a substantial speed up to the system. From a rough 8 seconds of search in the database, the time is cut to ¼.

# **Evaluation**

The effectiveness of information retrieval systems is measured by comparing performance on a common set of queries and documents. Since the classifiers above, provided good results, in case a full article or paragraph was passed in, it was needed to create an experiment where the number of words in the query are completely random.

The experiment that was executed was formulated as following. Since there are labelled data in the database, retrieve all the documents with their respective label. For any given random document, sample a random part of text, which is of random length. Use the classifier to predict the contextual group of the given query and compare it to the actual query. The results are provided below.

In a population of 1.000 samples, only 78 were predicted correctly, which is a ratio of 7.8%. Although the classifier provided good results for an embedding created from an article (big text), it provides poor results for an embedding created from smaller text. It means that a user probably would get irrelevant documents to their search.

As first step to increase the accuracy of the system, similarities among the contextual groups were tested. It was found out that there were 144 possible relationships, in a group of 18 teams, with an average cosine similarity greater than 0.20. It was decided that these groups would be included in the database search for similar documents.

The experiment was conducted again, with the same parameters of random text of random length. This time in the same population of 1.000 samples, we got results for 551 samples, which is a probability of 55.1%. The results are roughly 7 times better than before. The precision of the IR system is 0.549 on average, with a recall of 0.0653 on average, and a F1-score 0.093 on average.

The last evaluation was conducted by the users. In a set of ten random queries by each user, the results were satisfactory, since the words or phrases searched, were present in the printed results. The context of the query and the results was the same. This is giving indication that the concept of comparing similarities among the groups of documents should be expanded. Although the speed of results will be dropped, due to more computations, the recall and F1-score would rise.

# **Graphical User Interface**

In this section, the creation of a graphical user interface (GUI), thanks to which the visualization of the constructed IR system is feasible, is being presented. Like many great search engines, such as Google, Yahoo, etc., a simple search engine interface - a single input box is constructed utilizing Streamlit. Streamlit is an open-source Python library that makes it easy to create and share custom web apps for machine learning and data science in minutes.

In the Python script, the UI is being built and connected with the search engine.

The main elements of the search engine application are:

1. Create the Title
2. Render a Logo
3. Create the Search bar

Resulting in the following structure:

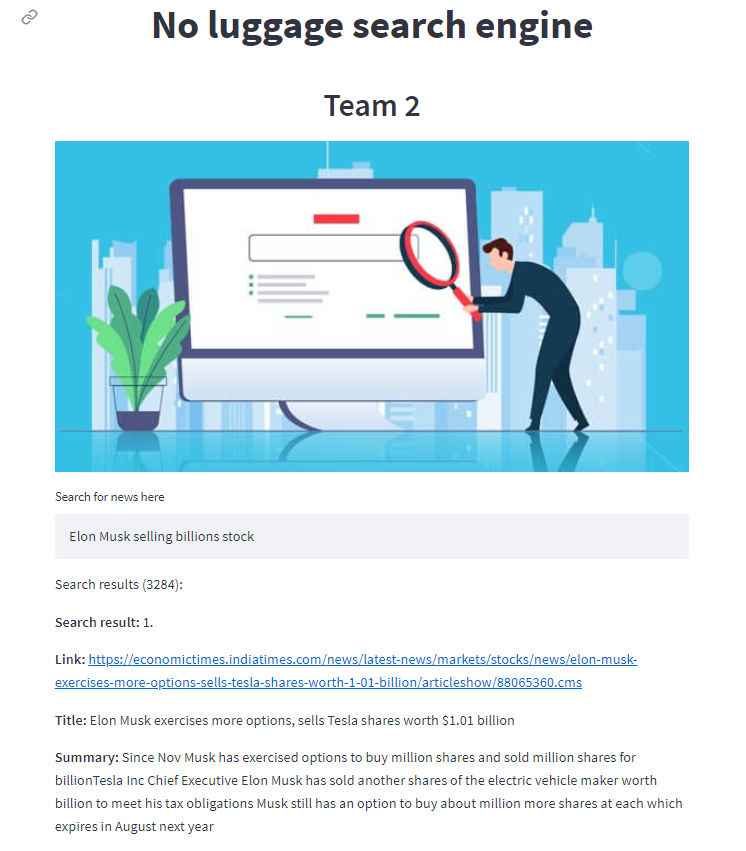


Figure 11: The main page with a text bar to enter the search words

If the search\_query is non-empty, the program calls the Python script, which returns the similar documents in a dictionary. The output is then processed in such way, so that only the title, link and summary of the most relevant articles appear on the Streamlit application.

As mentioned, the summary of each article is presented to the user. It was considered that a small summary of the full article would serve both visualization and utilization purposes. Regarding the first, it is designing wise better to print documents of shorter length instead of long documents. Regarding the latter, the user can understand whether it is needed to read the full article from the summary. This can save him time and give him the choice to redirect to the data provider site, where the full article may be found.

Text summarization in this concept is implemented again with the help of pretrained Bert Model. In this case, BART (large-sized model) is a transformer encoder-encoder model with a bidirectional encoder and an autoregressive decoder. It is pre-trained by computing text with an arbitrary noising function and learning a model to reconstruct the original text. The version is used has been fine-tuned on CNN Daily Mail, a large collection of text-summary pairs (Lewis, et al., 2019).

In the scenario where the search\_query is empty, the message “No Search results, please try again with different keywords” appears.

# **Future Implementation**

The immigration of the IR system into the cloud is perfectly attuned to the global corporate trend. It is a fact that both lower-level (storage, computing) and higher-level (application solutions) resources benefit from cloud migrations not just in terms of cost: security can be improved by a cloud migration, as cloud vendors can spend more on top-grade security staff in large quantities due to these vendors’ scale. Elasticity is another huge cloud advantage; the number of servers or application users can be rapidly increased or decreased, since cloud vendor infrastructure is available in abundance.

Scraping more documents from more sites would be another goal. For this paper, only two sites were scrapped daily. Since the IR system will be deployed on the cloud, it would be easier to gather articles from tenths of sites and create a big database.

Since more data will be added, clustering and classification algorithms used in this IR system deployment should be revisited and re-evaluated. More articles would land better results for clustering, creating better and more cohesive groups.

Additional ideas that would improve the search engine’s efficiency are inspired by Google’s search engine.

* Adding quotes, so that the user could search for specific phrases. This would minimize the guesswork for search and locate the specific information that may be buried under other content if not sorted out correctly.
* Adding the hyphen option, in the case the user searches for ambiguous words. Hyphen could be used as in Google search, for excluding words.
* Adding colons for searching specific sites. For example, typing Data Science site: acg.edu, would result in retrieving for all content about Data Science, but only on acg.edu. All other search results would be removed.
* Using the wildcard asterisk in a search term on search would leave a placeholder that may be automatically filled by the search engine later.
* Shortcuts should be included in a dictionary or list. In case the user was to type acronyms, abbreviations, idioms, or slang, the IR system should be able to “translate” them into their formal writing or vice versa.

Finally, the model could be packed and deployed in iOS or Android, as a personal assistant to track down the news, in this case financial news, of interest.

# **Conclusion**

After deploying and evaluating the full IR model, we draw the conclusion that the attempt had some success, it is scalable and there is room for improvements, as noted in the previous section.

First, the document retrieval pipeline is consistent, meaning that it gets daily the same amount of data and stores them in a database, along with an extracted summary of the article, the estimated document group it belongs to and stores the respected Bert embedding to a JSON file, used along with the database. Three future changes would be suitable for implementation here. Gathering more articles from different sites, using parallel computing to speed up and parallelize the processes, and deploy the modules on a local or cloud server to achieve 24/7 scrapping.

Regarding the database maintenance, there are modules which consistently clean the articles, remove near duplicate documents, can reassess the creation of document groups and create a classifier model to embed in the IR system. To fully automate these processes, deployment on a local or cloud server is necessary.

Finally, to improve the already fair results provided from the Information Retrieval system implementation, further research and optimizations need to be done, to increase accuracy and performance speed.

**References**

Brody, Hartley. “Web Scraping with Proxies: The Complete Guide to Scaling Your Web Scraper.” *Web Scraping with Proxies: The Complete Guide to Scaling Your Web Scraper*, Hartley Brody, 3 Sept. 2018, https://blog.hartleybrody.com/web-scraping-proxies/.

ET. “Business News Today: Read Latest Business News, India Business News Live, Share Market & Economy News.” *The Economic Times*, 2021, https://economictimes.indiatimes.com/.

Gunasekaran, Sathiya Sarathi. “Guide to Parsing HTML with BeautifulSoup in Python.” *LaptrinhX*, LaptrinhX, 18 Nov. 2020, https://laptrinhx.com/guide-to-parsing-html-with-beautifulsoup-in-python-796626741/.

Gupta, Lavanya. “Differences between word2vec and Bert.” *Medium*, The Startup, 7 Jan. 2021, https://medium.com/swlh/differences-between-word2vec-and-bert-c08a3326b5d1.

My Market Beat , d. “Stock Market News and Research Tools.” *MarketBeat*, 2021, https://www.marketbeat.com/.

Roberts, Edward. “Is Web Scraping Illegal? Depends on What the Meaning of the Word Is: Imperva.” *Blog*, 7 Sept. 2020, https://www.imperva.com/blog/is-web-scraping-illegal/.

Wenig, Phillip. “Creation of Sentence Embeddings Based on Topical Word Representations.” *ResearchGate*, Nov. 2018, <https://www.researchgate.net/publication/330761695_Creation_of_Sentence_Embeddings_Based_on_Topical_Word_Representations>.

Lewis, et al. (2019, 10). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. Retrieved from http://arxiv.org/abs/1910.13461

Reimers, N. & Gurevych, I. (2019,11). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics. Retrieved from <http://arxiv.org/abs/1908.10084>

Donges, Niklas. “Introduction to NLP.” *Built In*, May 2021, https://builtin.com/data-science/introduction-nlp.