Graphical user interface, application

Description automatically generated

**SEARCH ENGINES AND WEB MINING**

**ITC 6008**

Instructor: Dr. Lazaros Polymenakos

TEAM PROJECT:



Search Engine dedicated to Tech Products

Panagiotis Trafalis

Ioanna Veizi

Ioannis Fitsopoulos

Thodoris Diamantopoulos

Table of Contents

[1. Abstract 3](#_Toc90305353)

[2. Introduction 5](#_Toc90305354)

[3. Crawler creation 6](#_Toc90305355)

[4. Pre-processing the corpus 7](#_Toc90305356)

[5. Query processing 9](#_Toc90305357)

[6. Information Retrieval system 10](#_Toc90305358)

[7. Evaluation of the Information Retrieval System 12](#_Toc90305359)

[8. Constructing a classifier 13](#_Toc90305360)

[9. Setting up the User Interface 17](#_Toc90305361)

[10. Possible Upgrades to the Search Engine 19](#_Toc90305362)

[11. Conclusion 21](#_Toc90305363)

[Bibliography 21](#_Toc90305364)

# Abstract

This project revolves around the creation of a search engine that is dedicated to the retrieval of information about tech products. This information would be retrieved specifically on three sites (tomdsguide, digitaltrends, techadvisor). Several links were scrapped, specifically on the reviews and news sections offered by these sites. This procedure yielded around 102k articles which would be the repository for the search engine. After that, the whole corpus was pre-processed to be in the appropriate form for analysis. 3 different variations of the tf-idf were then computed for the articles along with the Bert embeddings for each sentence inside the corpus. The matrices produced were stored in a SciPy sparce matrix form in separate files. This would allow for a faster retrieval of them instead of doing this process each time a search is carried out. The umap algorithm was also performed on the Bert embeddings in order to reduce their high dimensionality nature. Moreover, the pre-trained models of tf-idf, Bert and umap were stored in a pickle format to be at hand for the encoding of the query. As for the query itself, a spell checker was constructed that would correct any misspells on it and also a corpus was constructed manually that would expand the query posed with similar ones. Then the query was embedded with tf-idf, and similarities were computed with the weights of the already embedded articles (once with stopwords and once without them). The 10 top articles were returned to the user based on the similarity of the tf-idf. Afterwards the query was embedded once again with Bert and similarities were computed this time with the sentences of the top 5 articles that the previous tf-idf method had fetched (hierarchical ranking). Several queries were used for testing and based on the relevance of the articles retrieved with the query posed, some metrics were computed that proved that hierarchical ranking was yielding the best results. The last step of the information retrieval system was to build a classifier that would filter out any non-relevant articles and then some metrics were once again computed to evaluate the system. All these procedures would though be bypassed if the user performed a query that was already applied to the search engine. Finally, a user interface was set up for the user by utilizing the flask module. The flowchart below encapsulates the whole approach (Figure1).

Timeline

Description automatically generated

Figure A flowchart representing the steps for the search engine

Keywords: search engine, tech, crawling, tf-idf, Bert

# Introduction

Back in 1990, the World Wide Web (WWW) was invented. This innovative creation led us to the so called “Information Age”, which managed to connect the world. But its creator, Tim Berners-Lee, quickly understood that the WWW could be utilized as a free resource for everyone around the globe to communicate and share knowledge. This revolutionary invention changed every possible aspect of people’s lives but the flaw of such a breakthrough was evident even before its creation. How would users easily retrieve the information they seek without wading through several irrelevant sites? The idea that led to the solution for this problem predates the debut of the World Wide Web and is no other than the so-called search engines. A search engine is a software accessed on the Internet that searches a database of information according to the user’s query (computerhope, 2021). The first tool that was used for searching information online was called Archie and was created by Alan Emtage, but it bears no resemblance to the idea of a search engine that people have in the present days. It took several years and many different approaches to the search engine archetype until the widely known Google rose to prominence with its innovative algorithm PageRank. At this moment in time the information residing online is so vast that it basically covers the majority of human knowledge. Now imagine that you have “someone” that can help you retrieve that knowledge precisely, within an instant. This makes it evident that search engines are the backbone of today’s information retrieval and play a crucial role in our everyday lives.

For our project, we created a search engine that can retrieve information about tech gadgets. Although limited at the time to a specific part of the internet, this can be the base of a search engine that may fetch information from all around the web. In our report, we will explain the way the specific sites were crawled, how their content was pre-processed and finally the approach of filtering the results to satisfy the user’s query.

# Crawler creation

The first step in creating a search engine was to acquire the content that would be returned to the user after he poses his query. In order to do that a crawler was required that would browse 3 tech related sites (tomdsguide, digitaltrends, techadvisor) and specifically their news and reviews sections. So, by utilizing the Web spidering framework called Scrapy a spiderbot was initialized to browse the aforementioned sites. Certain URLs were set as the initial ones for the crawler, that would then perform concurrent requests to the domains to obtain each URL’s content and all the links contained in it. All the links found were then filtered to restrict them to the 3 websites and then stored in a list. By storing them into a list, the crawler would be able to track which URLs had already been visited and scrapped and browse the ones that were not. From each address’s content the crawler would parse only the parts with the URL, title and paragraph tags. All the information contained in them would then be stored in an xml file. Due to the huge selection of tech articles in sites like the ones crawled and the limited broadband speed it was easily understood that the process should be performed in small batches and in different computers simultaneously. This procedure granted us 102k unique articles of tech related products that would be the base of our search engine’s corpus.

# Pre-processing the corpus

The first step before setting up our information retrieval system was to pre-process our data to be in the appropriate form. First, all files that were stored in small batches were collected and concatenated in order to produce one data frame containing all the articles. The structure of the data frame produced was set to be, first the URL of the corresponding article, then its title and finally the actual content of it.

After creating the data frame, we had to make some minor adjustments to the text, like removing extra full stops or adding a whitespace after some of them since these caused issues with the tokenization. Moreover, we had to replace any existing contractions. A contraction is a word made by shortening and combining two words (edu.gcfglobal, n.d.). So, by replacing those, a more uniform corpus was produced, avoiding same words counting as different ones. Next, sentence tokenization was performed on the articles, which is the process of dividing a large quantity of text into smaller parts called tokens (Johnson, 2021).

The resulting data frame consisted of 2 million rows, with each row containing a sentence along with the corresponding URL that it was found in. From this set of sentences, all rows containing duplicate sentences were dropped along with the ones that had only punctuation marks in them. Moreover, sentences deriving from the headers and footers of a page were also dropped as they did not hold any useful information for our information retrieval system.

From the remaining sentences, Bert embeddings were produced. The reason behind that, is that Bert model helps computers understand the meaning of ambiguous language in text by using surrounding text to establish context (Lutkevich).

Then, the tf-idf vectorizer was applied on each one of the articles of the corpus. The tf-idf is a statistical measure that evaluates how relevant a word is to a document in a collection of documents (Stecanella, 2019). This procedure was performed in 3 different variations. Once for just the tf-idf of the articles. Another one for the tf-idf after the stopwords were removed. The last one was the tf-idf with the stopwords removed from the articles and by performing lemmatization to it.

The resulting embeddings of both Bert and tf-idf were stored in separate files in a SciPy sparse matrix form. By doing so, it would be easier and notably faster to perform a query and use these pre-made files than creating the embeddings each time. The data frame with all the articles containing the URLs, the titles and the content was saved in a csv format. Along with it, the data frame containing the final 1.1 million sentences and their corresponding URLs was also stored in a csv file. By inspecting the Bert embeddings, it was evident that keeping them as they were along with their high dimensionality nature would cause time inefficiencies. So, it was decided to apply the umap dimension reduction technique to them in order to reduce the dimension of their matrix. This was performed twice, with the first time reducing the dimensions from 768 to 350 and the second one from 768 to only 10. Besides the corpus, the query must be encoded with the same already trained model. Considering that, the models of Bert, tf-idf and umap were stored in a pickle format to be ready for use when a query is posed. Finally, in order to have a more tech-oriented corpus than the ones that the previously applied models use, a list with all the unique words contained in the articles was stored in a pickle format.

# Query processing

The next step after processing our main corpus was to adjust the way each query would be handled. The first step was to build an efficient spell checker by utilizing the minimum edit distance method. The minimum edit distance between two strings is the minimum number of editing operations (insertion, deletion, substitution) needed to transform one word into the other (Jurafsky, 2017).

An issue that had to be addressed was that by posing a query, the similarities with the articles on the corpus would be computed based only on the exact words that were used inside the query. In order to avoid missing any articles that use different wording than the query but with the same meaning, a query expansion was set up. It would create similar queries to be searched along with the original one by substituting words with the same meaning (Figures 2 & 3). This was done manually by creating a corpus that contained words along with their synonyms.

Text

Description automatically generated

Figure Expanded query

Table

Description automatically generated with low confidence

Figure Results after query expansion

# Information Retrieval system

For our information retrieval system, we had several different approaches. For the first part that was about the tf-idf 3 variations were tested. The first one was to use the tf-idf along with the default settings set by scikit learn, the next one by removing any existing stopwords and then compute the tf-idf and the last one to remove the stopwords, lemmatize the corpus and then calculate the tf-idf. Then each performed query is embedded with the tf-idf vectorizer in order to compute the cosine similarity between the query and the computed tf-idfs of the articles. The cosine similarity is a measurement that quantifies the similarity between two or more vectors (Alake, 2020). So, after computing all the similarities the articles were sorted based on that metric. Considering that only a certain number of articles would be relevant to the performed query there was no reason of keeping all of them, so it was decided to return only the top 10 of them. This would also serve the purpose of creating the Bert embeddings much faster since there would be less sentences left. The next step was to embed the query once again, but this time by utilizing Bert. The similarities were then computed between those embeddings and the ones that were stored previously for the sentences using 3 different variations. The first with the matrix of the Bert embeddings having all 768 dimensions, the second one with the dimensions of the matrix reduced with umap to 350 and the last one with the dimensions reduced to only 10. This step was performed only for the top 10 articles that the previous tf-idf method had fetched. Next, after the reranking was performed only the top 5 articles were kept since the corpus was limited and only a few of the articles would be representative of the query posed. The approach above is called hierarchical reranking and is the process of reranking a list of items based on a metric that was already ranked given another metric. By applying both tf-idf and Bert methods we tried to avoid any unwanted results to be returned by our information retrieval system that a single method would not filter.

# Evaluation of the Information Retrieval System

In order to evaluate our information retrieval system some metrics had to be calculated. To do that a set of 30 specific queries was created and performed for all of the different methods, and the results were marked either as relevant or irrelevant to the posed queries. By deciding each of the returned article’s relevance, the precision, recall and f1 score were computed as the factors of efficiency of our method. After all the different combinations that were tested, the approach that proved to return the best results was to first use the tf-idf with lemmatization and the removal of stopwords and then apply Bert without reducing the dimensions with umap. So, the hierarchical method (with the classifier) was chosen to do the “dirty” work of filtering and ranking the articles. In the figure below, the f1 score off the several methos is presented (Figure 4).

Chart, bar chart

Description automatically generated

Figure 4 The f1 scores of the different methos used

# Constructing a classifier

Even if the hierarchical ranking method seemed to return pretty satisfying results, it was possible that by classifying those results based on relevance could enhance our information retrieval system even more. Considering that, a dataset was used that consisted of the queries, the similarities based on tf-idf and Bert, the unique document id and a column that pointed whether a returned article was relevant or not (Figure 5).

A picture containing text

Description automatically generated

Figure 5 The dataset used to evaluate the returned articles

The classifier used as target column the one containing the relevance and the similarities and the documents’ ids as attributes and its purpose was to discard any non-relevant articles that were not filtered out by the hierarchical ranking that was performed prior to this. This target column was constructed by performing manually 200 relevant to the corpus queries and specifying whether the 1000 articles returned, were relevant or not. Our first attempt on constructing the classifier used the dataset without the unique document id with the only attributes being the similarities of tf-idf and Bert. This approach returned an f1 score of 0.62 with the ANN classifier and far worse scores with the rest that were tested. After implementing the unique document id as an attribute that was related with the similarities corresponding to it (Figure 6), the classifier started to present its value. Even though the ANN this time scored a petty f1 score of 0.34, the decision tree and random forest classifier showed a significant improvement and returned an f1 score of 0.72. In order to visualize the results, the ROC curves were plotted for each classifier (Figure 7).

Table

Description automatically generated

Figure 6 The dataset used for training with the Doc\_id included

Chart, line chart

Description automatically generated

Figure 7 ROC curves without scaled data

Another approach that had to be tested, was to scale our data (Figure 8). Metrics were computed once again for each classifier, along with the corresponding ROC curves (Figure 9 & 10).

Table

Description automatically generated

Figure 8 Train set with scaled data

Chart, line chart

Description automatically generated

Figure 9 ROC curves of the classifiers with scaled data

Graphical user interface, text, application, email

Description automatically generated

Figure 10 Precision, Recall, F1 score of Random Forest Classifier with scaled data

Despite the slight improvement of the ANN classifier when the data got scaled, the rest of the classifiers appeared to be unaffected by the scaling. So, the random forest classifier without scaling the data was picked as the best approach, to perform the filtering of the results. If more articles could be evaluated for the training of the classifier, better results would be expected, but with the resources available the model is unsurprisingly underfitted.

# Setting up the User Interface

But a search engine cannot exist as long as the user is not able to interact with it and perform queries. In order to resolve that, a user interface was constructed by utilizing flask (Figure 11). Flask is a web framework, included as a module in python that enables developers create web applications (pythonbasics, 2010).

Graphical user interface, text, application, email

Description automatically generated

Figure 11 The search engine from the user's point of view

While the flask server is launching, all tf-idf, Bert and umap models get initialized. The reason behind that is to confine the calculations that need to be performed in the article retrieval process. The ones that are calculated in real time are the similarities between the query and the articles with both the tf-idf and Bert, the encoding of the actual query and finally the prediction of the classifier that was constructed. All these procedures are not time consuming and that is the motive behind storing all the models once, when our information retrieval system was set up. The benefits of such an approach proved to be extremely beneficial timewise. For example, with the tf-idf being computed each time the ranking was performed, a single search would take up to 14 seconds to return the results, while with the tf-idf stored the time taken was reduced to less than a second.

It is easily understood that when a user performs a query there is a high chance of a misspelled word being submitted. In order to resolve this issue most search engines that exist will correct the query and search for the edited one. Since the articles that our search engine is based on, are tech oriented, which means that many words may not have a meaning and be just names of gadgets, we decided that the correction of a typo must not be performed automatically. The search engine just suggests to the user a similar query to the one he posed and leaves the choice to his own discretion (Figures 12 & 13).

Graphical user interface, text, application, email

Description automatically generated

Figure 12 Suggestion after misspelled search

Text

Description automatically generated with medium confidence

Figure Search results after spell correction

After the query is performed by the user the results of the top n articles are returned to him. These results contain the corresponding URL of each article, the most similar sentence to the query in each page as a demonstration of the content that it encapsulates and the value of the similarity that was calculated for the hierarchical reranking by using tf-idf and then Bert. Finally, a bypass was inserted into the user interface that skips the whole procedure of the information retrieval system if the query performed by the user has already been searched for.

# Possible Upgrades to the Search Engine

The whole procedure that our team followed led to an effective base for a tech-oriented search engine. Even though satisfying results were returning from the queries posed there will always be room for improvement. First, considering that a search is most of the times specified to a certain gadget, topic modelling could be performed on the articles. Topic modeling is a technique to extract the hidden topics for a collection of documents (Prabhakaran, 2018). By doing so, all the procedure would be performed to a subset of the corpus that has the same topic with the query, which would reduce the time of the whole process to less than a quarter.

Another approach that would lead to better results, would be to take into account the date and the location that an article was posted and re-rank the results based on them. For example, by not considering the date of the article, a query like “best phone” could bring results like a page about Nokia 3310 which may have been considered one of the best phones but that happened several years ago. Most of the times, the user performing the search looks for the most updated info about a topic. This issue occurred because our first approach was to store certain parts of each page, and the date and location were not contained in them, but that could be easily fixed if there were enough resources to re-download the whole corpus.

An alternative method that could be utilized, would be to store the articles in an online database instead of using data frames, as most popular search engines do. This would certainly be more time and space efficient and further improve our information retrieval.

Moreover, we could retrain the Bert model based on our corpus. Even if Bert is pretrained in two huge corpora like BookCorpus and the English Wikipedia, by training it once again on more domain-oriented documents, would probably lead to better results.

The last issue that our team faced and could be resolved is the difficulty in downloading articles from sites that use JavaScript in their coding and are not static. This could be tackled by either utilizing the Selenium module in Python or the Splash library of Scrapy and would allow us to access a wider variety of sites for our corpus.

# 11. Conclusion

By constructing our information retrieval system, we managed to create the base of a search engine where a user can look for tech-oriented articles. The way that the crawler, the preprocessing of the corpus and query, the information retrieval system, the user interface and generally the whole design was created proved to be efficient as no complications have emerged upon returning results to the user. The resulting outcome incorporates all the different traits required for an effective search engine even though tackling such a task proved to be really challenging. It is evident that by adjusting our method as mentioned in the previous section, our information retrieval system could be further improved to return more accurate results faster. But even for Google, it took over 20 years to reach the level of sophistication that we see today and get amazed of its performance. Concluding, the whole project allowed us to further understand the way that search engines operate which after all play a key role in people’s everyday lives in acquiring information and exploring the world. As Steve Jobs once quoted, “If you haven’t found it yet, keep looking” …in our search engine!

# Bibliography

(n.d.). Retrieved from edu.gcfglobal: https://edu.gcfglobal.org/en/grammar/contractions/1/

Alake, R. (2020, 09 15). *towardsdatascience.* Retrieved from https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a

Bechhofer, S. (2004, 2 10). *w3.* Retrieved from https://www.w3.org/TR/owl-ref/#FunctionalProperty-def

*computerhope*. (2021, 11 10). Retrieved from https://www.computerhope.com/jargon/s/searengi.htm

Gibbins, N. (2012, 3 15). Ontology Design Patterns. University of Southampton. Retrieved from https://pdfs.semanticscholar.org/1200/ee8fe811390c858c17fce74d266b4596ffe3.pdf

Horridge, M. (2011, 3 24). A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.3. p. 108.

Johnson, D. (2021, 11 1). *guru99.* Retrieved from https://www.guru99.com/tokenize-words-sentences-nltk.html

Jurafsky, D. (2017, 03 17). Retrieved from https://web.stanford.edu/class/cs124/lec/med.pdf

Lutkevich, B. (n.d.). *techtarget.* Retrieved from https://searchenterpriseai.techtarget.com/definition/BERT-language-model

Prabhakaran, S. (2018, 03 26). *machinelearningplus*. Retrieved from https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/

*pythonbasics*. (2010, 10 25). Retrieved from https://pythonbasics.org/what-is-flask-python/

Stecanella, B. (2019, 05 11). *monkeylearn.* Retrieved from https://monkeylearn.com/blog/what-is-tf-idf/

Stevens, R. (2011, 1 12). Retrieved from ontogenesis.knowledgeblog: http://ontogenesis.knowledgeblog.org/1001/

Tim Berners-Lee, J. H. (2001, 5 17). The Semantic Web. *Scientific American*.