Design and Implementation of an Image Search Engine

Khizer Ahmed Biyabani
khizer.biyabani2@mail.dcu.ie
School of Computing, Dublin City University
Dublin, Ireland

ABSTRACT

The process of recovering data from a computer database or a huge collection of data is known as information retrieval. The study that follows proposes a system for implementing an image retrieval system. It discusses many methods for crawling various photos from a website, and then returning related images to a text query. The following sections cover several text-processing algorithms as well as calculating the similarity score using the BM25 probabilistic model, which produced better results in the prior assignment. Finally, we provide a simple user interface for conducting our search.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction**; • **Information systems** → **Query representation**; **Information extraction**; **Probabilistic retrieval models**; **Similarity measures**; **Relevance assessment**; **Presentation of retrieval results**; **Retrieval efficiency**; **Search interfaces**; **Image search**.

KEYWORDS

Information Retrieval, Image Retrieval, BM25, Web Scrapping, BeautifulSoup, OpenCV, Natural Language Processing, Anvil, User Interface, Inverted Index

ACM Reference Format:
Khizer Ahmed Biyabani. 2022. Design and Implementation of an Image Search Engine . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Information retrieval is all about gathering useful or relevant piece of information from a collection, this collection can be textual collections like 'Corpus' i.e. a collection of texts, or any other content based searching like Image retrieval. Image retrieval is a method of searching, viewing, and retrieving images from a large database of images. Close text choices are typically accepted by image search engines.

A search engine is one of the most often used means for retrieving information saved online, and it has become embedded in our culture. There are many common search engines like Google, YouTube, Bing and many other resources with the help of which we can easily find the piece of information we need. But eventually the big question is how these search engines use different retrieval task to search for the user desired information.

The following article discusses how to build an image search engine comparable to Google Image Search or any other common image search engine. The next parts go through how to gather various photos from a website, generate textual annotations for each of these images, and index them. Finally, the user's submitted text query is compared using the BM25 technique to gather relevant photos. This assignment also requires the creation of a basic user interface.

1.1 About the Data

The data that is used is been gathered using a web crawler, in this case we have used pythons' BeautifulSoup. The images are gathered from the website https://www.freepik.com/. As we are implementing a simple search engine that searches a particular fruit from a database of several fruit images. In this case we have used 12 category of fruits. Further, section explains about the gathering and collection of images of each category.

2 SYSTEM ARCHITECTURE

The following figure 1 explains the entire process through the task and different stages:

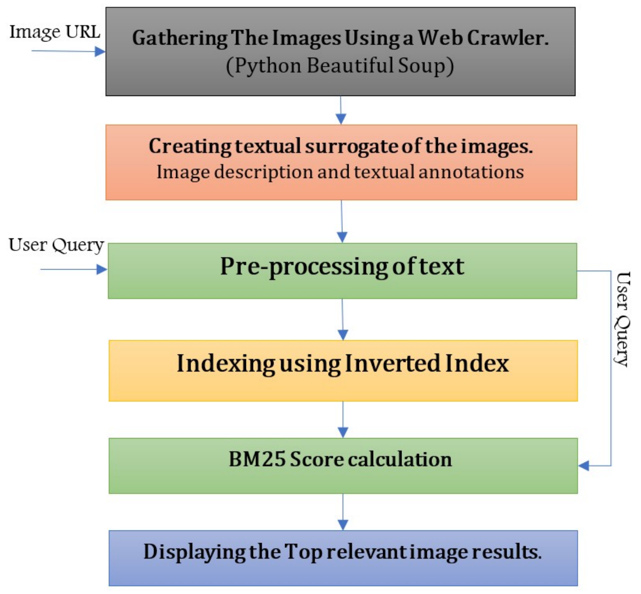


Image URL
Gathering The Images Using a Web Crawler.
(Python Beautiful Soup)
Creating textual surrogate of the images.
Image description and textual annotations
User Query
Pre-processing of text
User Query
Indexing using Inverted Index
BM25 Score calculation
Displaying the Top relevant image results.

Figure 1: A Flowchart explaining the system architecture or the structure of my solution.

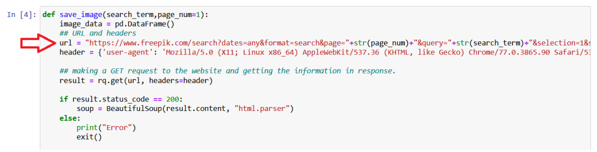
As it is evident from the architecture, the entire process is divided into various sections or stages. The images are crawled from the website url, these downloaded images are then annotated with their textual descriptions. The text is then pre-processed and an inverted index is created. The user query is then taken as an input, which is again pre-processed and the BM24 score of each query term is calculated with each image text term and finally the top relevant images are displayed. The further sections explain in detail about the entire process.

2.1 WebScraping of Images

The technique of obtaining information from the internet is known as web scraping. When we scrape the web, we develop code that makes a request to the server that hosts the page we want. The server will return the source code for the page (or pages) we requested, which is usually HTML. Beautiful Soup is a Python package for extracting data from HTML and XML files for web scraping. It generates a parse tree from the page source code, which may be used to extract data in a more legible and hierarchical manner. [1]

For this assignment we have crawled 100 images of 12 different fruit categories: 'apple', 'banana', 'watermelon', 'grapes', 'guava', 'pineapple', 'orange', 'raspberry', 'mango', 'kiwi', 'melon', and 'avocado'.

The URL consists of the search_term and page_num variable and other paramaters and this URL is address to the content from where we are going to extract the images.The search_term is used to search the images on the website from which we are going to extract these images and alt attribute for the image information. The page_num indicates the current page number from where we are scraping the images, we use this variable to go to the required page and this will help us to extract the images from multiple pages. The following figure 2 shows how search term and the page number are used in the URL.



```
In [4]: def save_image(search_term,page_num=1):
            image_data = pd.DataFrame()
            ## URL and headers
            url = "https://www.freepik.com/search?dates=any&format=search&page="+str(page_num)+"&query="+str(search_term)+"&selection=1&s
            header = {'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/53

            ## making a GET request to the website and getting the information in response.
            result = rq.get(url, headers=header)

            if result.status_code == 200:
                soup = BeautifulSoup(result.content, "html.parser")
            else:
                print("Error")
                exit()
```

Figure 2: The URL

Beautiful soup in Python generated a parse tree based on each HTML element for the full HTML document. Because we're dealing with pictures in this situation, the'soup' parse tree we developed places a greater weight on the <img> tag. As illustrated in figure 3, we first extract the source URL of each of our images from the website. The source URL of an image may be determined in the HTML <img> tag.

These images are then stored and downloaded as shown in in figure 4. These images are gathered from https://www.freepik.com/

2.2 Annotations of the Images and Preparing the DataFrame

The photographs that we obtained in the previous stage must be organized in such a way that we can quickly process the text and

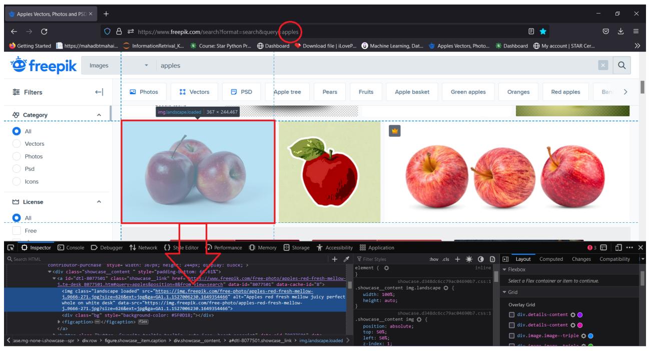
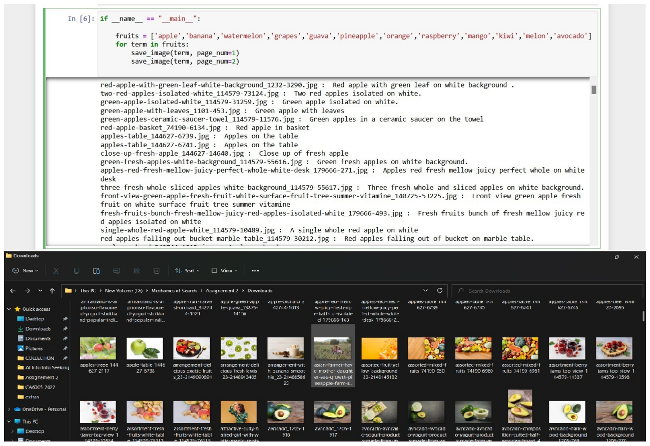


Figure 3: A image showing how the source URL is taken



```
In [6]: if __name__ == "__main__":

            fruits = ['apple','banana','watermelon','grapes','guava','pineapple','orange','raspberry','mango','kiwi','melon','avocado']
            for term in fruits:
                save_image(term, page_num=1)
                save_image(term, page_num=2)
```

```
red-apple-with-green-leaf-white-background_1232-3290.jpg :  Red apple with green leaf on white background .
two-red-apples-isolated-white_114579-73124.jpg :  Two red apples isolated on white.
green-apple-isolated-white_114579-31259.jpg :  Green apple isolated on white.
green-apple-with-leaves_1101-453.jpg :  Green apple with leaves
green-apples-ceramic-saucer-towel_114579-11576.jpg :  Green apples in a ceramic saucer on the towel
red-apple-basket_74190-6134.jpg :  Red apple in basket
apples-table_144627-6739.jpg :  Apples on the table
apples-table_144627-6741.jpg :  Apples on the table
close-up-fresh-apple_144627-14640.jpg :  Close up of fresh apple
green-fresh-apples-white-background_114579-55616.jpg :  Green fresh apples on white background.
apples-red-fresh-mellow-juicy-perfect-whole-white-desk_179666-271.jpg :  Apples red fresh mellow juicy perfect whole on white
desk
three-fresh-whole-sliced-apples-white-background_114579-55617.jpg :  Three fresh whole and sliced apples on white background.
front-view-green-apple-fresh-fruit-white-surface-fruit-tree-summer-vitamine_140725-53225.jpg :  Front view green apple fresh
fruit on white surface fruit tree summer vitamine
fresh-fruits-bunch-fresh-mellow-juicy-red-apples-isolated-white_179666-493.jpg :  Fresh fruits bunch of fresh mellow juicy re
d apples isolated on white
single-whole-red-apple-white_114579-10489.jpg :  A single whole red apple on white
red-apples-falling-out-bucket-marble-table_114579-30212.jpg :  Red apples falling out of bucket on marble table.
```

Figure 4: A snippet showing the crawled images.

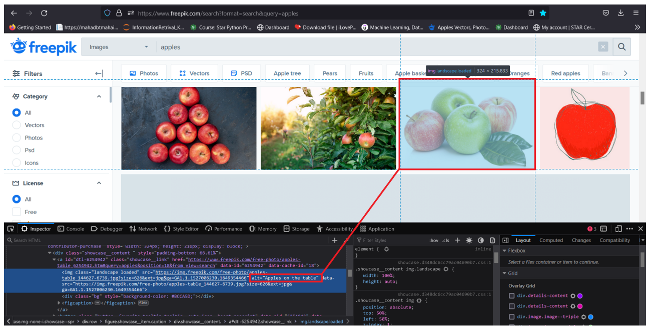
complete our required work. These images needs some annotations or text surrogates or in other words some textual information that tells us about a particular image of that fruit.

The information about each image can be found in the <img> tag of the HTML doccument itself, it contails an attribute called as 'alt' or 'Alternative Text'. The text in this 'alt' attribute explains what the image is supposed to symbolize. Another, importance of this text is in terms of search engines it helps to give relevant results for a particular user query. By reading the 'alt' tag, search engine crawlers may determine the visual content of an image, and then display a page in search engine results.[2]

In this assignment, the 'alt' attribute from the <img> tag of the HTML document of the website is extracted at the time of the web scrapping function. We created a separate 'csv' file 'photos' that contains the information regarding the fruit category, the 'alt' text , the 'src' or source link of the image. The following figure shows how the alt text can be seen in the inspect element of the website.

Pandas library of python has an amazing concept of creating a Dataframe, this helps us to organise our data and then we can perform all the necessary actions for image retrieval.

With the help of the 'csv' file created as mentioned above we create a pandas Dataframe to create the final data set, as shown below. Now we have, the following attributes in it:

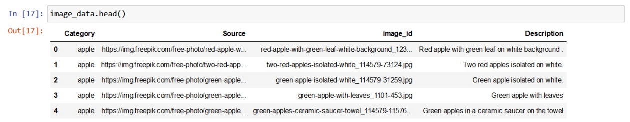


```
<div class="showcase__content " style="padding-bottom: 66.61%">
  <a id="dtl-6254942" class="showcase__link" href="https://www.freepik.com/free-photo/apples-table_6254942.htm#query=apples&position=18&from_view=search" data-id="6254942" data-cache-id="18">
    <img class="landscape loaded" src="https://img.freepik.com/free-photo/apples-table_144627-6739.jpg?size=626&ext=jpg&ga=GA1.1.1527006230.1649354466" alt="Apples on the table" data-src="https://img.freepik.com/free-photo/apples-table_144627-6739.jpg?size=626&ext=jpg&ga=GA1.1.1527006230.1649354466">
    <div class="bg" style="background-color: #BCCA5D;"></div>
    <figcaption>…</figcaption>
  </a>
```

Figure 5: 'alt' text extraction

(1) Category: The fruit image category from the motioned list of categories.
(2) Source : The original image link from 'src' attribute of <img> tag.
(3) image_id: A unique id for each fruit image as per the category, for example the apple images has ids as apple0, apple1, ... , apple 101 and so on.
(4) Description: The text annotated in the 'alt' attribute pf <img> tag.

The Dataframe looks like the one in figure 5:



```
In [17]: image_data.head()
Out[17]:
```

	Category	Source	image_id	Description
0	apple	https://img.freepik.com/free-photo/red-apple-w...	red-apple-with-green-leaf-white-background_123...	Red apple with green leaf on white background .
1	apple	https://img.freepik.com/free-photo/two-red-app...	two-red-apples-isolated-white_114579-73124.jpg	Two red apples isolated on white.
2	apple	https://img.freepik.com/free-photo/green-apple...	green-apple-isolated-white_114579-31259.jpg	Green apple isolated on white.
3	apple	https://img.freepik.com/free-photo/green-apple...	green-apple-with-leaves_1101-453.jpg	Green apple with leaves
4	apple	https://img.freepik.com/free-photo/green-apple...	green-apples-ceramic-saucer-towel_114579-11576...	Green apples in a ceramic saucer on the towel

Figure 6: The final Image Dataset

2.3 Pre-processing of the Text (Image Descriptions)

The text pre-processing portion involves converting the original textual data to a raw-data structure, which identifies the most important text features for distinguishing across the text categories. This is the most important and difficult stage since it leads to the representation of each text in the description by selecting a collection of index words. This is one of the essential parts of the process of information retrieval.

The textual data that we have was very disorganised, as they contain a lot of unusual text that has to be treated, like text with numbers, alphabets and special symbols. As well as mixed case texts, hence we need text pre-processing.

There are several methods that are involved in the pre-processing of the text, which include:, b3

- Changing the entire text to Lower Case.
- Removing Punctuation and Special Symbols
- Lemmatising the text Tokenisation of the text and Parts of Speech (POS) Tagging
- Stop word removal

2.3.1 Changing the Case of the text to Lower. Its very important to make the text uniform in nature, so that it can be processed and handled easily. As we know that upper and lower case letter are treated differently, for example 'MAP' is different that 'map' in terms of the ASCII values. But sometimes changing the case of the text may also affect its actual meaning, as some upper case letters convey emotion or excitement. The python string has inbuilt '.lower()' function that can be used to change the case.

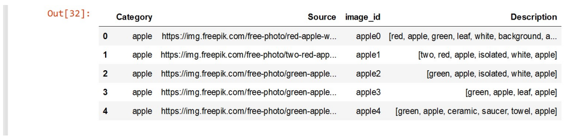
2.3.2 Removing Punctuation. The punctuation marks or special symbols used in English should also be removed as they can also interfere in the processing of the text. In python string.punctuation has a list of all punctuation symbols, that can be stripped out from the corpus.

2.3.3 Lemmatization and Stemming of the Text. Stemming is the method of converting the text into its root form. This involves converting the present or past participles into their stem. For example, rocks can be converted rock. But sometimes this method does not give meaning full results. Hence, lemmatization is done in this the words are converted into their root form or lemma. For example better can be converted to good and so on. NLTK is a Natural Language Toolkit in python which has a WordNetLemmatizer class that can been used to Lemmatize the text into their root form. [4]

2.3.4 Tokenization and POS Tagging. Another important text pre-processing step is to convert the sentences in the text into tokens or individual words. These tokens are accompanied by POS tagging or Parts of Speech tagging, which means to identify the nouns, verbs, adjectives and other parts of speech of the English language from the text. This extra information can make the process of lemmatization more easier. Also, tokenization of the text makes it easier for further text analysis. NLTKs 'wordtokenize' and 'getwordentpos()' can be used for this purpose. [4]

2.3.5 Removal of Stop Words. There are certain words that do not add any meaning to the text, like the articles a,and,the,are etc. These stop words can be removed by using NLTKs nltk.corpus.stopwords, which contains the list of stop words. The concept is to simply remove terms that appear in all of the papers in the corpus. Articles and pronouns are typically categorized as stop words.

Finally, after pre-processing we have a clean and tokenized text corpus. It must be noted that it took about 3hrs to lemmatize, tokenise and POS tag the given corpus. The following figure shows the pre-processed text:



```
Out[32]:
```

	Category	Source	image_id	Description
0	apple	https://img.freepik.com/free-photo/red-apple-w...	apple0	[red, apple, green, leaf, white, background, a...
1	apple	https://img.freepik.com/free-photo/two-red-app...	apple1	[two, red, apple, isolated, white, apple]
2	apple	https://img.freepik.com/free-photo/green-apple...	apple2	[green, apple, isolated, white, apple]
3	apple	https://img.freepik.com/free-photo/green-apple...	apple3	[green, apple, leaf, apple]
4	apple	https://img.freepik.com/free-photo/green-apple...	apple4	[green, apple, ceramic, saucer, towel, apple]

Figure 7: A screen-shot showing the pre-processed image description text

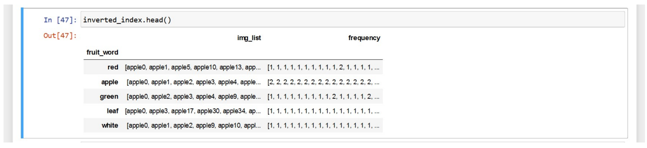
3 INDEXING

Text indexing is the process of extracting statistics from a text in order to reflect the information provided and/or to allow quick searches on its content. Text indexing processes can be conducted on nearly any sort of textual information, including source code for computer programs, DNA or protein databases, and textual data stored in typical database systems, in addition to natural language texts.

There several methods of indexing the text such as document-term matrix, but this method of indexing is merely a sparse matrix, that involves mostly zero computation and can be said as a waste of time. For full-text indexing, most information retrieval (IR) systems employ inverted indexes as their primary data structure. Inverted index is one of the robust ways of accessing the corpus text and its frequency. A word, or atomic search item, is mapped to the collection of documents, or indexed units, that include that word and its postings in an inverted index. This helps in easy retrieval of the word and its frequency in that document. [5, 6]

For this system we have created a pandas dataframe of inverted index. First, we created a list of vocabulary or collection of all unique words in the image descriptions, which is about 821 words. Then in the inverted index dataframe we considered each vocabulary word as the index and in the next column we determined the number of images with their image ids, where we can find that particular term or word. The third column contains the frequency of the word in a particular image description.

As you can see from the figure below, the inverted index has three columns, the 'fruit_word' which is the word from the vocabulary of words, the second column 'img_list' gives the list of images in which the fruit_word is present in its description and the last column gives the frequency of that fruit_word in those image descriptions.



```
In [47]: inverted_index.head()
Out[47]:
```

	img_list	frequency
fruit_word		
red	[apple0, apple1, apple5, apple10, apple13, app...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, ...
apple	[apple0, apple1, apple2, apple3, apple4, apple...	[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
green	[apple0, apple2, apple3, apple4, apple9, apple...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, ...
leaf	[apple0, apple3, apple17, apple30, apple34, ap...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
white	[apple0, apple1, apple2, apple9, apple10, appl...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

Figure 8: A screenshot of the project notebook depicting the inverted index created.

From, the above figure 8 we can determine for example the word 'red' can be found in a list images' descriptions apple0, apple5, .. and so on. This helps, us when we have a user query we can compare and determine which image description has the similar word in the query and we can display all those related images. This is further explained in the next section.

4 IMAGE RETRIEVAL

4.1 BM25 Implementation

When we deal with information retrieval there are different ways to extract the relevant information for a particular query. Let us recall the system architecture that we discussed in section 2, our main aim is to find the relevant images for the users text query. Now, the text that user enters needs to be compared with the text related to each of our images which is the image description. Lets say, we want gave a query 'green apple', now the search engine must find the similar images for the given query for that it need to find similar text in the image descriptions from our dataset. Hence, there are certain similarity measures that can be used to find the top similar or relevant results for the user entered query.

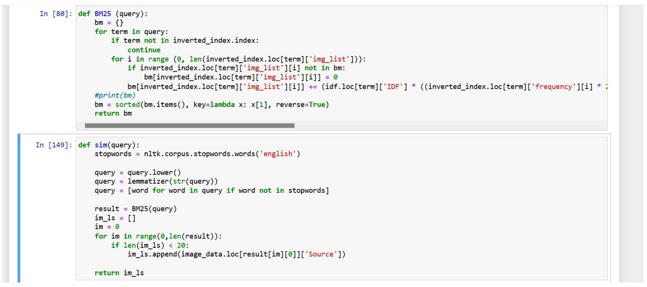
One such method is 'BM25 Information Retrieval model', this model based on calculating a probability of relevance for each pair of documents (images and their description in this case) and ranking them in descending order of likelihood of relevance in respect to a particular query. This model basically was basically derived from 2-Poison model. The 2-Poisson model offered useful information regarding a weighting function's behavior and saturation. To find the relevance of a query with its document we use the following BM25 score:

$$score(D,Q) = \sum_{i=0}^{n} IDF(q_i) \frac{f(q_i,D)(k_1+1)}{f(q_i,D) * (1 - b + b * \frac{|D|}{avgdl})} \qquad (1)$$

In the above equation (1) $IDF(q_i)$ is the inverse document frequency of each of the query terms, $f(q_i, D)$ is the frequency of the ith query term in the document that can be determined by the inverse index created, |D| is the length of the document D in words, and $avgdl$ is the average document length in the text collection from which documents are drawn, this can be calculated from the original corpus stored in python dictionary 'data3', and finally the terms 'b' and 'k1' are constants, that can take the values from 0.5 to 0.8 and 1.2 to 2 respectively. For this system the values are k=1.2 and b=0.75.[7]

As mentioned in the above equation (1), we need inverse document frequency for each of the term in the image description, this is calculated by the formulae:$IDF(t,d) = \log \frac{N}{|\{d \in D : t \in d\}|}$

The following figure shows the function for similarity score (BM25) calculation of a 'query' text.



```
In [80]: def BM25 (query):
             bm = {}
             for term in query:
                 if term not in inverted_index.index:
                     continue
                 for i in range (0, len(inverted_index.loc[term]['img_list'])):
                     if inverted_index.loc[term]['img_list'][i] not in bm:
                         bm[inverted_index.loc[term]['img_list'][i]] = 0
                     bm[inverted_index.loc[term]['img_list'][i]] += (idf.loc[term]['IDF'] * ((inverted_index.loc[term]['frequency'][i] *
             #print(bm)
             bm = sorted(bm.items(), key=lambda x: x[1], reverse=True)
             return bm

In [149]: def sim(query):
              stopwords = nltk.corpus.stopwords.words('english')

              query = query.lower()
              query = lemmatizer(str(query))
              query = [word for word in query if word not in stopwords]

              result = BM25(query)
              im_ls = []
              im = 0
              for im in range(0,len(result)):
                  if len(im_ls) < 20:
                      im_ls.append(image_data.loc[result[im][0]]['Source'])

              return im_ls
```

Figure 9: A screenshot of the project notebook depicting the BM25 score calculation

With help of the 'BM25' scores calculated we can determine, which are the top relevant images based on a particular query, by simply calculating the similarity of the query with each image description text.

4.2 User Interface for Image Search

With the help of 'Anvil,' a basic User Interface was constructed. Anvil is a free tool for creating user interfaces in Python. It gives us with a user interface through which we can develop our web app's design. It connects two components: client-side python code, which runs our jupyter notebook, and server-side python code, which operates the Anvil server.

This project's user interface merely accepts a user query or the fruit whose images must be obtained. This query is then passed to our jupyter notebook's similarity function, which displays the image results.

Below is a brief representation of the search user interface snippet.

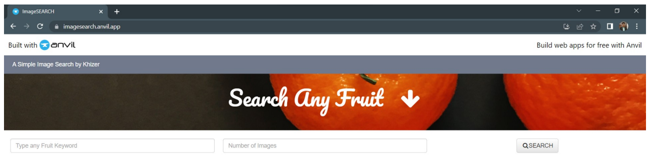


Figure 10: A snippet of the search UI

5 EVALUATION

Three distinct queries for searching the photographs are used to test the search engine. We can see in the diagram above that we have a user interface with a search field where you must type in your query. The top results are then displayed after the text from the query is compared to each image description using the BM25 similarity. The amount of results to be presented is determined by the user; for example, if the user requests 10 photos for a query, he will receive them.

To test the search engine's functionality, we ran the following three queries: The search results for each query are shown in figures 11, 12, and 13.

(1) Query 1: 'Green Apple on a table'
(2) Query 2: 'Fruit Basket'
(3) Query 3: 'Fruit Salad'

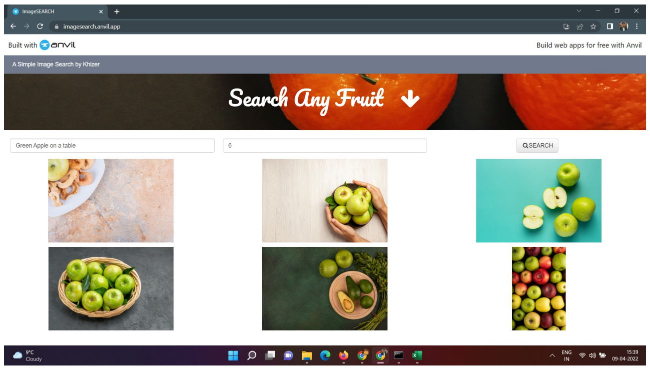


Figure 11: Query1 : "Green Apple on a table"

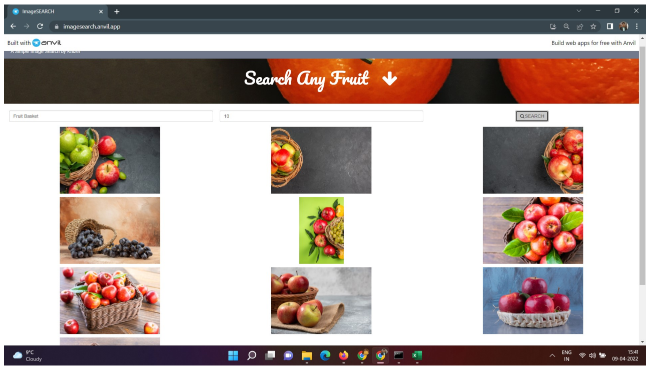


Figure 12: Query 2: "Fruit Basket"

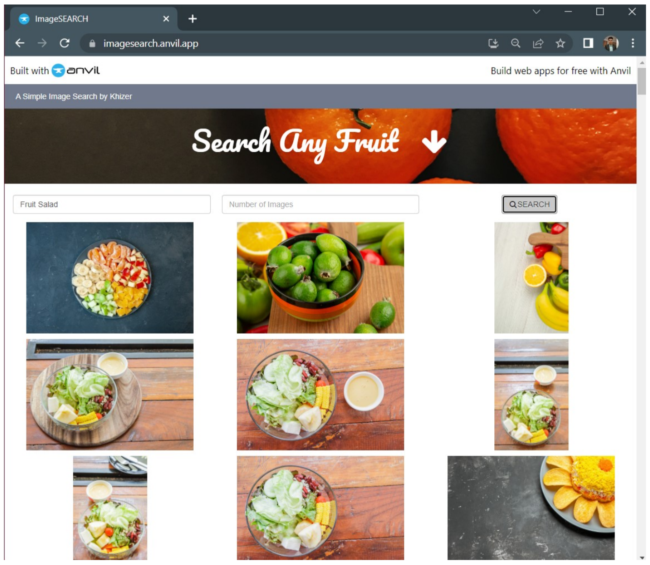

Built with anvil
Build web apps for free with Anvil
A Simple Image Search by Khizer
Search Any Fruit
Fruit Salad
Number of Images
SEARCH

Figure 13: "Query 3: "Fruit Salad"

6 CONCLUSION

To summarize, this paper proposes a simple image retrieval system that returns relevant photos from a collection of images, similar to a modest replica of an image search engine. We scraped images from an image website and retrieved information about each image using the html tag properties of the <img> tag. For cleaning the annotated text or image descriptions, various text pre-processing techniques were used. The BM25 score was then utilized to determine the degree of similarity between the user query and each image description in order to determine the top photos for the query. Finally, to do our search, we designed a simple user interface.

We can add more features in the future, such as advanced search, which allows the user to type in the type of search results he wants, as this was a simple rip-off of Google image search. More image annotation technologies can also be used to extract more picture data, allowing us to gain more image description and, as a result, create better results.

REFERENCES

[1] Scrape Beautifully With Beautiful Soup In Python BY MOHIT MAITHANI, PUBLISHED ON DECEMBER 4, 2020 IN DEVELOPERS CORNER, https://analyticsindiamag.com/beautiful-soup-webscraping-python/#:~:text=Beautiful%20Soup%20is%20a%20Python,hierarchical%20and%20more%20readable%20manner.

[2] What Is Image Alt Text? https://elementor.com/resources/glossary/what-is-image-alt-text/?utm_source=google&utm_medium=cpc&utm_campaign=11138809851&utm_term=&gclid=CjwKCAjwur-SBhB6EiwA5sKtjg_pd6XEfRduO8Tfw_5w7eFmANuSKgTn_JISkB46kTjyGpqCcR8SvxoCAxoQAvD_BwE

[3] Kadhim, Ammar. (2018). An Evaluation of Preprocessing Techniques for Text Classification. International Journal of Computer Science and Information Security,. 16. 22-32.

[4] Khyani, Divya B S, Siddhartha. (2021). An Interpretation of Lemmatization and Stemming in Natural Language Processing. Shanghai Ligong Daxue Xuebao/Journal of University of Shanghai for Science and Technology. 22. 350-357.

[5] Indexing Shared Content in Information Retrieval Systems Andrei Z. Broder1,?, Nadav Eiron2, , Marcus Fontoura1, , Michael Herscovici3 , Ronny Lempel3 , John McPherson4 , Runping Qi1, , and Eugene Shekita5

[6] D. Cutting and J. Pedersen. 1989. Optimization for dynamic inverted index maintenance. In Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '90). Association for Computing Machinery, New York, NY, USA, 405–411. DOI: https://doi.org/10.1145/96749.98245

[7] Robertson, Stephen Zaragoza, Hugo. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval. 3. 333-389. 10.1561/1500000019.

[8] Mechanics of Search: Multimedia Retrieval Dublin City University a course on FutureLearn.

[9] An Introduction to Scraping Images With Python by Nikhil Tomar 14th June 2020, https://idiotdeveloper.com/an-introduction-to-scraping-images-with-python/