Mechanics of Search: Assignment 2

MARCEL AGUILAR GARCIA, National University of Ireland, Galway, Ireland

This report shows the implementation of a search engine focused on animals pictures. Scrapy has been used to download around 1500 images from Wikipedia animal pages. The search engine uses the ranking function BM25 based on image annotations from HTML content and YOLO. The results are presented in a simple website using Python and Flask.

Additional Key Words and Phrases: information retrieval, BM25, image search engine

Link to Image Searcher: http://marcelgarcia.pythonanywhere.com/
User: dcumarcelaguilargarcia
Password: dcuassignment2

INTRODUCTION

Animal Image Search Engine is an image search engine with access to more than 1500 animal related images. All images have been downloaded from from a list of 68 Wikipedia animal pages (e.g. cat, fish, lion, etc.) using Scrapy. A Python script annotates all images by using HTML content and labels returned from computer vision-based tools (YOLO). BM25 implemented in MoS Assignment 1 is used to return the 10 most relevant images to the user query. A simple website with this search engine has been published in Pythonanywhere using Python and Flask.

The project is structured as seen in Figure 1. Scraper folder contains a Scrapy project with a spider, wikiscraper.py, that downloads all images from a list of urls into static folder. After downloading all images, executing collection_generator.py will create a collection of documents with all images annotations (using HTML content and YOLO) in file/collection folder. Finally, a user can enter a query into the main page query.html from which app.py will find all the relevant documents from collection folder using BM25 and return the top 10 results. The user will be redirected to successful.html where will be able to see these images.

ir.py contains three main classes that are used through all the process. ImageHTMLContentRetrieval contains all the methods required to annotate images. TextProcessor is used to pre-process files from the collections (e.g lowercasing, stemming, etc.). Finally InformationRetrieval is used to compute the inverted index from the collection of documents and return the relevant documents using BM25.

Author's address: Marcel Aguilar Garcia, National University of Ireland, Galway, Galway, Ireland.

Manuscript submitted to ACM

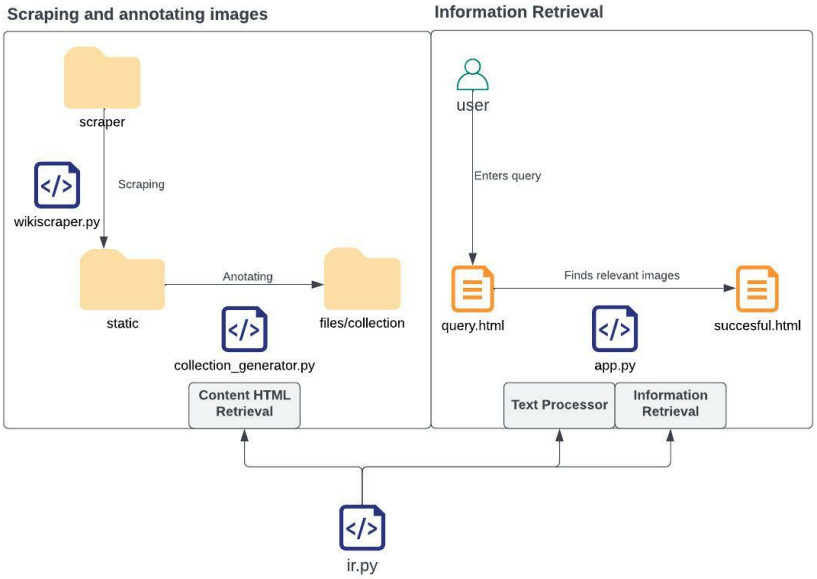

Fig. 1. Project Structure

ANNOTATION

collection_generator.py creates the folder **collection** with one text document per image. The annotation process consists of two main steps:

(1) Annotate HTML Content
(2) Annotate recognised labels from YOLO

In the first step, *alternative text*, *title*, and *parent text* are extracted from the HTML code where the image was downloaded from. If the image contains none of these elements, the *image name* is used instead. All this information is saved as a text file in **file/collections**.

In the second step, YOLO is used to recognise additional information from the image. YOLO contains specific animal labels such as horse, giraffe, dog, etc. so it is expected that it should improve the engine performance. On top of this, if the same label is identified multiple times in an image, that word is added as many times into the related document (not just once). For example, if YOLO recognises five dogs in a picture, the document will contain the word 'dog' five times consecutively at the end. This is aligned with the fact that the BM25 implemented uses tf-idf and therefore is sensitive to the word frequency.

INDEXING

As I have used the algorithm from Assignment 1 the indexing is the same as described in the previous assignment. In this particular case I am using **inverted index**. As implemented, BM25 retrieves only documents that contain at least one of the words from the query. Therefore, the efficiency for the algorithm to find this documents is essential. While a

document-term matrix would provide the same results, the time performance of the search engine would decrease. Additionally, as the collection of images used for this project is not going to change, I have decided to save the inverted index locally so it does not have to be re-calculated for every search. If new images were to be added in the future, I would run the annotation script again and re-upload the updated index.

The inverted index provides a link between each words from the vocabulary and the documents where they occur. Additionally, I have added the term frequency of each word per document. Figure 2 shows an example of inverted index in which word 1 occurs twice in Document 5, once in Document 8, twice in Document 50, and four times in Document 98.

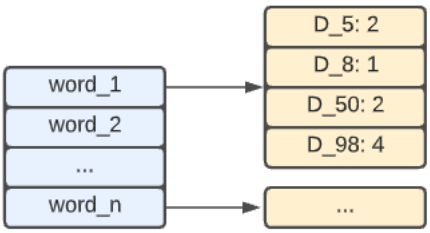

Fig. 2. Example inverted index with document term frequencies

RETRIEVAL

I have chosen a simplistic approach for the user interface with just one search bar for the query and one button 'search' to perform the query. As the collection of images used is quite limited, I have decided to just returned the first ten ranked images. All images returned are then populated to the user in a grid.

Fig. 3. UI Search Engine

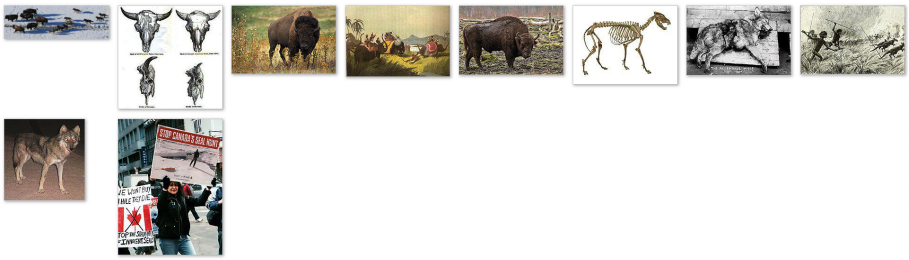

Fig. 4. UI Results

The ranking function used is the one of Assignment 1. My motivation to use BM25 from Assignment 1 is to have an image searcher that has been fully implemented by me. BM25 combines the weight w^{RSJ}, the term frequency derived from the 2-poisson model, and the document length normalisation into one single weight. The pseudo-code for the BM25 implementation is:

Algorithm 1 BM25 ranking

```
b ← 0.6
k ← 1.5
N ← len(collection)
for word in query_processed do
    if word in inverted_index then
        tf ← inverted_matrix[word][doc_id]
        dl ← docs_length[doc_id]
        avdl ← mean(docs_length.values)
        n ← len(inverted_matrix[word])
        bm25_score+= (tf/(k((1-b)+b(dl/avdl))+tf))log((N-n-0.5)/(n+0.5))
    end if
end for
return bm25_score
```

Note that for the purpose of this assignment I have set $k = 1.5$ and $b = 0.6$.

The search engine populates just the first ten documents ordered by BM25 ranking.

EVALUATION

First, the search engine has been tested with simple one word queries such as 'cat' or 'panda' in where, in most of cases, all images returned were considered relevant. As expected, animals that had not been included were unlikely to have images returned (e.g. duck). I noticed that the images for some queries such as 'dog' were not as accurate as I would expect. One of the hypothesis is that YOLO miss-labeled some images, for example in this case, miss-labeled wolfs as dogs.

By looking at a sample of pictures I performed more complex queries such as 'Wolves hunting bison' which surprisingly returned the first two images completely relevant (20%) and most of remaining results could be related to one of the words from this query. Finally, and concerned than the current implementation is not using n-grams, I decided to compare results of queries from "Killer Whale" vs "Whale". The first query had 77.7% of relevant images while the second one had 100% relevant images with only one 'Killer Whale' (12.5%) followed by other types of whales.

As well it is important to mention that while I did not test YOLO labeling specifically, I have tested that words such as 'person' return pictures with people. Additionally, after performing few searches I realised that when searching for 'donuts' we can see an image related to Elk patties which YOLO has labeled incorrectly.

CONCLUSIONS

I am quite satisfied with the performance of the search engine. Especially, I would like to highlight the ability of the search engine to return relevant results for queries with a certain degree of complexity. On top of this, saving the inverted index locally, has allowed the search engine to have a great time performance as the results are generally returned in milliseconds.

There are few things I would like to improve. To start with, the images downloaded by Scrapy have small size and low resolution. In consequence, the images provided to the user may not have good quality and YOLO has probably miss-labelled few of them, leading to incorrect results (e.g. images of wolfs when searching for 'dog'). Finally, this has been the first website I have created and, while I believe that the User Interface is good enough, it should be improve, for example, by allowing the user to open a specific image from the list of results.

In the future, apart from improving image quality, I think it would be interesting to use an open-source search tool (e.g. Lucene, MG4J, Solr, ElasticSearch, Lemur, Terrier, etc) and compare the performance with the information retrieval used from Assignment 1.