## **Final Report -Implementation Project**

## Srinath.V.S

# Steam Games Information retrieval System

Youtube Link : <https://youtu.be/unx1oISj5cE>

About the system :

This is an Information Retrieval system based on the Steam games dataset on Kaggle.

The system works well on the game dataset and is somewhat fast, when the actual size of the number of features is taken into question, ( there are 6.4 billion elements in the data, 80,000 documents, 80,000 terms in the documents.).

The query is processed in about 5-7 seconds and the results are then shown via web scraping from the url for the required fields.

Readme file has all the steps needed to run the program.

Process Used to Retrieve the Data :

**Indexing Process : ( done in Indexer.py )**

Initially, the data is cleaned to remove duplicates and then is used to create inverted list structure with terms and document frequencies.

The inverted list is used to determine the idf values of the terms and is also stored as a pickle dump.

The inverted list is further used to create a sparse document-term matrix.

The intermediate data is stored in a folder Stored\_Data to be referenced later.

Sample Code to clean the Data :

def clean\_steam():  
 # Getting data from the csv files using pandas  
 data = pd.read\_csv(r'BaseData/steam\_data.csv')  
 data\_des = pd.read\_csv(r'BaseData/text\_content.csv')  
  
 # cleaning all data, removing duplicates and incomplete rows  
 data.drop\_duplicates(subset='url', keep='last', inplace=True)  
 data\_des.drop\_duplicates(subset='url', keep='last', inplace=True)  
 data\_des.drop\_duplicates(subset='desc', keep='last', inplace=True)  
  
 # removing rows that have null elements in identifying columns  
 data = data.dropna(subset=['name'])  
 data\_des = data\_des.dropna(subset=['full\_desc'])  
  
 # create dataframes to merge  
 url\_frame\_left = pd.DataFrame(data=data, columns=['url', 'name', 'categories', 'developer', 'publisher'])  
 url\_frame\_right = pd.DataFrame(data=data\_des, columns=['url', 'desc'])  
 url\_frame = pd.merge(url\_frame\_left, url\_frame\_right, how='inner', on='url')  
 url\_frame.drop\_duplicates(['url'], keep='last', inplace=True)  
 filename = 'Stored\_Data/clean\_frame'  
 outfile = open(filename, 'wb')  
 pickle.dump(url\_frame, outfile)  
 outfile.close()

The document lengths are then estimated for the document-term matrix and stored separately in the aforementioned folder.

**Processing Query : (done in Indexer.py, SearchUI.py )**

The user’s input is entered on the UI generated by the SearchUI.py and submit is hit, the UI is then deleted and the query processing part takes over.

The user’s input is modified to include a synonym of the word provided it is already not present in the list of words to be evaluated using nltk’s wordnet corpus. This extends the query and adds more words that were previously not present in the query.

The modified input from the user is then passed to a function that traverses the inverted list structure and then tries to find relevant documents to the query, this shortens the retrieval process. ( this cut down on the time taken to retrieve the documents in order of similarity from 30 to 5 seconds).

The parsing through the inverted list helps provide the term frequencies of the values in the query. These term frequencies are then multiplied by their corresponding idf values from the stored list in the indexing process to get the actual weights when divided by the length of the query vector.

The cosine similarity is then evaluated between the query and each relevant document and the results are returned. The processed queries return urls to the actual game on steam.

The actual code used :

def process\_query\_reduced(input\_query,  
 train,  
 path\_to\_idf='Stored\_Data/idf\_list',  
 path\_to\_terms\_dta='Stored\_Data/terms\_data',  
 path\_to\_document\_lengths='Stored\_Data/document\_lengths',  
 path\_to\_doc\_mat='Stored\_Data/document\_term\_matrix.npz',  
 path\_to\_inverted\_list='Stored\_Data/inverted\_list'):  
 # Fro time logging  
 start = time.perf\_counter()  
  
 # Parsing of the input query  
 stop\_words = nltk.corpus.stopwords.words('english')  
 stemmer = PorterStemmer()  
 word1 = word\_tokenize(input\_query)  
 word1 = [word for word in word1 if word not in stop\_words]  
 print("Query words are : ", word1)  
 wordlist = []  
  
 # Extending query to also include one synonym of the word from wordnet  
 for word in word1:  
 syns = wordnet.synsets(word)  
 if len(syns) > 0:  
 if len(syns[0].lemmas()) > 1:  
 print("Syns is : ", syns[0].lemmas()[1].name())  
 if syns[0].lemmas()[1].name() not in word1:  
 wordlist.append((syns[0].lemmas()[1].name()))  
  
 # including the synonyms to the actual list of words  
 word1.extend(wordlist)  
  
 # applying porter stemmer to lemmatize the words  
 word1 = [stemmer.stem(word) for word in word1]  
  
 # converting all words to lower case for uniformity  
 word1 = [word.lower() for word in word1]  
  
 # debug statement  
 print("query words are : \n", word1)  
 if train:  
 # opening all the files to the corresponding saved data in idf, inverted list and document matrix  
 infile = open(path\_to\_idf, 'rb')  
 infile1 = open(path\_to\_terms\_dta, 'rb')  
 infile2 = open(path\_to\_document\_lengths, 'rb')  
 infile3 = open(path\_to\_inverted\_list, 'rb')  
  
 # loading the data in the files to required variables  
 idf = pickle.load(infile)  
 doc\_lengths = pickle.load(infile2)  
 terms, data\_points = pickle.load(infile1)  
 terms = list(terms)  
 inverted\_dict = pickle.load(infile3)  
 query\_row = np.zeros(shape=(1, len(terms)), dtype=np.float32)  
 doc\_list = []  
  
 # convert the saved sparse matrix to a dense matrix  
 doc\_mat = scipy.sparse.load\_npz(path\_to\_doc\_mat)  
 doc\_mat.todense()  
 doc\_mat = doc\_mat.toarray()  
  
 # find reduced documents by ID based on items present  
 for word in word1:  
 # searching for the word from the query in the inverted term-document dictionary  
 if word in inverted\_dict.keys():  
 inverted\_docs = inverted\_dict[word].listvals  
 # Getting document indicies from the term-document dictionary and saving these in a relevant list  
 for doc, freq in inverted\_docs:  
 doc\_list.append(doc)  
 doc\_set = set(doc\_list)  
 doc\_list = list(doc\_set)  
 if len(doc\_list) == 0:  
 # if no data is present, then return a None result  
 print("Nothing is present")  
 return []  
 for word in word1:  
 if word in terms:  
 query\_row[0][terms.index(word)] += 1  
 print(query\_row)  
 for elem in range(len(query\_row[0])):  
 # tf x idf step for the query to find the weights  
 query\_row[0][elem] = query\_row[0][elem] \* idf[elem]  
  
 # finding the length of the query vector  
 query\_length = find\_Length\_vector\_for\_query(query\_row)  
  
 # dictionary to store cosine similarities  
 cosine\_sim = {}  
  
 # finding cosine similarities between relevant documents and query vector  
 for row in doc\_list:  
 dot = np.dot(doc\_mat[row], query\_row[0])  
 sim\_val = dot / (doc\_lengths[row] \* query\_length)  
 cosine\_sim[row] = sim\_val  
  
 # sorting cosine similarities by value of similarity  
 everything = sorted(cosine\_sim.items(), key=lambda a: a[1], reverse=True)  
 if len(everything) >= 10:  
 # Getting the top ten elements  
 reverse = everything[:10]  
 else:  
 # Getting the resulting set, if results size is < 10  
 reverse = everything  
  
 # # debugging statement meant to check the values  
 # for element in reverse:  
 # print(element)  
  
 result\_list = []  
 for k, v in reverse:  
 # Appending urls to the result to be returned  
 result\_list.append(data\_points[k].url)  
 print(data\_points[k].url)  
  
 # logging time for conclusion of search and indexing  
 end = time.perf\_counter()  
 diff = end - start  
  
 # Time taken for the search to conclude  
 print("Time taken = ", diff)  
  
 # returned results  
 return result\_list

**Displaying Results : (done in SearchUI.py and WebScraper.py**

The results are displayed in the form of a UI template that can be cycled through using the next and previous buttons. When Load is clicked, the url is used to scrape the website and get all the data for the required fields. This is done using WebScraper.py.

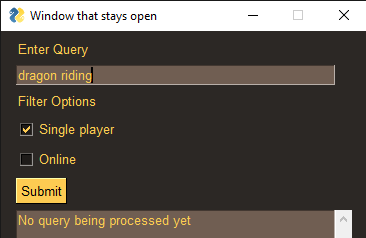
The data can be cycled through using the next and previous buttons. The indices of the data can also be seen in the terminal outputs as debugging statements.

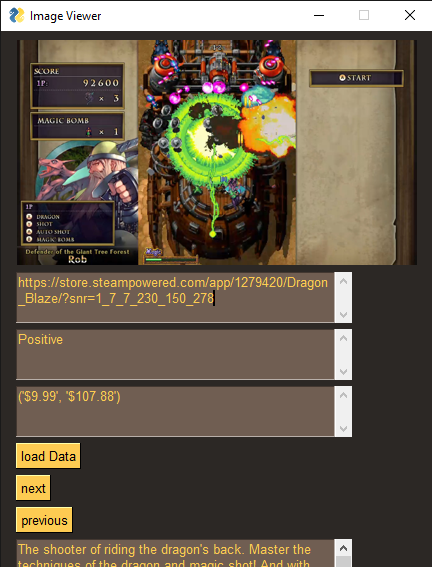
If no results are found, a pop up window shows that no relevant documents were found, then the user can enter a new query.

A sample function to scrape data from the web :

# returns description from the steam store page  
def get\_description(src):  
 source = requests.get(src)  
 soup = bs4.BeautifulSoup(source.content, 'html.parser')  
 description = soup.find("meta", property="og:description")  
 if description:  
 description\_text = description["content"]  
 print(description\_text)  
 return description\_text  
 else:  
 return "No description found"

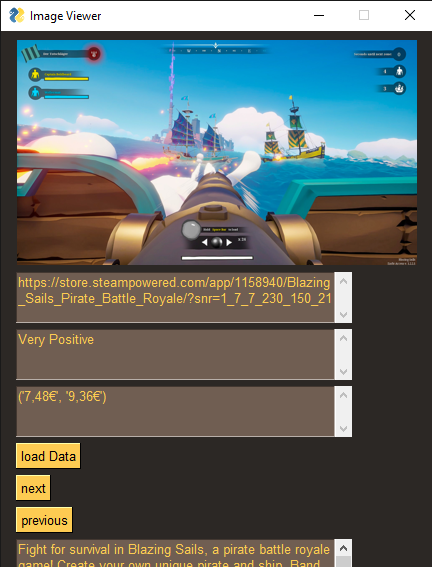
One example of a query and its processed output is shown below :





Testing the System :

Input query was : Sail a ship with friends and Online was ticked.



We ran the testing part on the test dataset provided at : <http://ir.dcs.gla.ac.uk/resources/test_collections/cacm/>

We weren’t sure how to evaluate the correctness of the system, so we attached the results here. The test results can be obtained by running Test.py in the submitted source code of the project. The test documents needed to be differently processed and we were unable to process if the test document in the same way as the source query.

The test dataset can be found in the TestData folder in the zipped file.

**TestCase Outputs for first two queries :**

**Query:**

.W

What articles exist which deal with TSS (Time Sharing System), an

operating system for IBM computers?

.N

1. Richard Alexander, Comp Serv, Langmuir Lab (TSS)

**Results:**

Query words are : ['.W', 'What', 'articles', 'exist', 'deal', 'TSS', '(', 'Time', 'Sharing', 'System', ')', ',', 'operating', 'system', 'IBM', 'computers', '?', '.N', '1', '.', 'Richard', 'Alexander', ',', 'Comp', 'Serv', ',', 'Langmuir', 'Lab', '(', 'TSS', ')']

query words are :

['.w', 'what', 'articl', 'exist', 'deal', 'tss', '(', 'time', 'share', 'system', ')', ',', 'oper', 'system', 'ibm', 'comput', '?', '.n', '1', '.', 'richard', 'alexand', ',', 'comp', 'serv', ',', 'langmuir', 'lab', '(', 'tss', ')']

[[0. 0. 0. ... 0. 0. 0.]]

195 : What is a Code?

325 : Numerical Solution of the Polynomial Equation (Algorithm 30)

1410 : Interarrival Statistics for Time Sharing Systems

339 : Real Exponential Integral (Algorithm 20)

289 : Critical Path Scheduling (Algorithm 40)

Time taken = 0.3851186999999925

**Query :**

.W

Intermediate languages used in construction of multi-targeted compilers; TCOLL

.N

3. Donna Bergmark, Comp Serv, Uris Hall (intermed lang)

**Results:**

Query words are : ['.W', 'Intermediate', 'languages', 'used', 'construction', 'multi-targeted', 'compilers', ';', 'TCOLL', '.N', '3', '.', 'Donna', 'Bergmark', ',', 'Comp', 'Serv', ',', 'Uris', 'Hall', '(', 'intermed', 'lang', ')']

query words are :

['.w', 'intermedi', 'languag', 'use', 'construct', 'multi-target', 'compil', ';', 'tcoll', '.n', '3', '.', 'donna', 'bergmark', ',', 'comp', 'serv', ',', 'uri', 'hall', '(', 'interm', 'lang', ')']

[[0. 0. 0. ... 0. 0. 0.]]

852 : A Method of Comparing the Time Requirements of Sorting Methods

205 : Macro Instruction Extensions of Compiler Languages

2652 : Reduction of Compilation Costs Through Language Contraction

1134 : Some Effects of the 6600 Computer on Language Structures\*

2252 : A Method for Incrementally Compiling

Time taken = 0.4114666999999912

More Sample Runs of the System :

**Query :**

returned values are : {'Query': 'assassin', 'Single player': True, 'Online': False, 'Show': [], 'Process': 'No query being processed yet\n'}

Query is : assassin Singleplayer

Query string is : assassin Singleplayer

Query words are : ['assassin', 'Singleplayer']

Syns is : assassinator

query words are :

['assassin', 'singleplay', 'assassin']

[[0. 0. 0. ... 0. 0. 0.]]

**Results :**

https://store.steampowered.com/app/687640/Infinity\_Assassin\_VR/?snr=1\_7\_7\_230\_150\_2636

https://store.steampowered.com/app/1013810/The\_Legendary\_Assassin\_KAL/?snr=1\_7\_7\_230\_150\_2089

https://store.steampowered.com/app/335460/An\_Assassin\_in\_Orlandes/?snr=1\_7\_7\_230\_150\_813

https://store.steampowered.com/app/1344500/The\_not\_so\_silent\_assassin/?snr=1\_7\_7\_230\_150\_2102

https://store.steampowered.com/app/311560/Assassins\_Creed\_Rogue/?snr=1\_7\_7\_230\_150\_60

https://store.steampowered.com/app/603460/King\_and\_Assassins/?snr=1\_7\_7\_230\_150\_1440

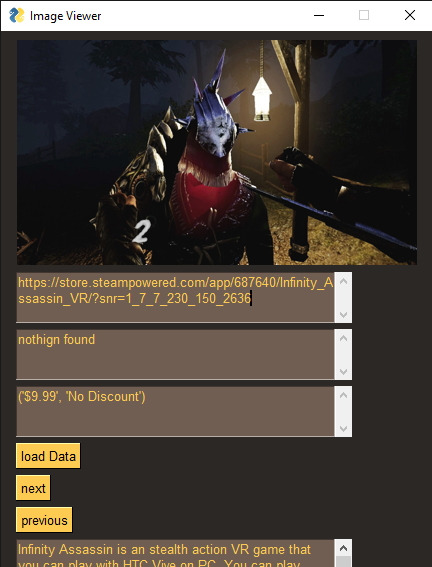
https://store.steampowered.com/app/551550/ASSASSINATION\_BOX/?snr=1\_7\_7\_230\_150\_1410

https://store.steampowered.com/app/918870/Legend\_of\_Assassin\_Egypt/?snr=1\_7\_7\_230\_150\_2119

https://store.steampowered.com/app/929690/Legend\_of\_Assassin\_Jungle/?snr=1\_7\_7\_230\_150\_2097

https://store.steampowered.com/app/925610/Legend\_of\_Assassin\_Siberia/?snr=1\_7\_7\_230\_150\_2134

Time taken = 4.642127499999997





2)

**Query :**

{'Query': 'best way to search for games ', 'Single player': False, 'Online': True, 'Show': [], 'Process': 'No query being processed yet\n'}

Query is : best way to search for games Online

Query string is : best way to search for games Online

Query words are : ['best', 'way', 'search', 'games', 'Online']

Syns is : mode

Syns is : hunt

Syns is : online

query words are :

['best', 'way', 'search', 'game', 'onlin', 'mode', 'hunt', 'onlin']

**Results :**

https://store.steampowered.com/app/961680/Rampage\_Online/?snr=1\_7\_7\_230\_150\_2855

https://store.steampowered.com/app/934550/Duck\_Hunting\_Challenge/?snr=1\_7\_7\_230\_150\_1843

https://store.steampowered.com/app/1330790/Hunting\_Unlimited\_2/?snr=1\_7\_7\_230\_150\_1464

https://store.steampowered.com/app/965640/Horror\_Legends/?snr=1\_7\_7\_230\_150\_657

https://store.steampowered.com/app/731890/Horror\_Hunt/?snr=1\_7\_7\_230\_150\_629

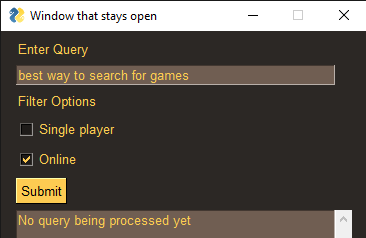
https://store.steampowered.com/app/545920/Hunting\_Unlimited\_4/?snr=1\_7\_7\_230\_150\_391

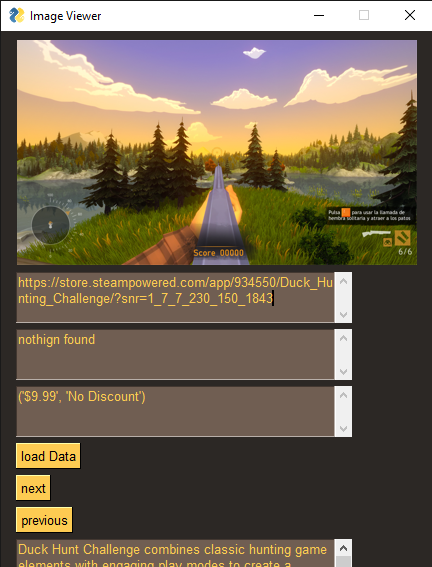
https://store.steampowered.com/app/12690/Hunting\_Unlimited\_2010/?snr=1\_7\_7\_230\_150\_287

https://store.steampowered.com/app/988340/Mad\_Hunting\_Simulator\_VR/?snr=1\_7\_7\_230\_150\_1375

https://store.steampowered.com/app/1350280/Jonas\_Willy\_Online/?snr=1\_7\_7\_230\_150\_1669

https://store.steampowered.com/app/401190/Dinosaur\_Hunt/?snr=1\_7\_7\_230\_150\_504





Inferences :

The program performed reasonably well on the dataset given the limiting nature of cosine similarity and the size of the dataset.

It could be improved by improving the speed and adding additional features like also including a window to check the phrases present in the query and search for phrases instead of just the words.

The system achieved the goal of showcasing a Cosine Similarity based IR system that can be used to return a set of games based on any input query.

Extending the query and adding synonyms made the output more prone to noise and error, due to difference in contextual meanings of a word, which caused wildly different results with respect to whether or not the synonyms of a word were to be included in the query.