International Journal of
Advances in Scientific Research and Engineering (ijasre)

E-ISSN : 2454-8006

DOI: 10.31695/IJASRE.2019.33651

Volume 5, Issue 12
December - 2019

A Domain-Specific Search Engine: A Case of University of Abuja

Yakubu Muhammed Sanusi[1] Aminat Ajibola[2*] and Amujo Oluyemi E [3]

[1-3]Computer Science Department

Faculty of Science, University of Abuja, Nigeria

ABSTRACT

Users need more effective ways of organizing and searching for information. It has become increasingly difficult for users to find information on the World Wide Web that satisfies their individual needs since information resources on the World Wide Web continue to grow. The reason is that the World Wide Web provides a bunch of information that is generic in perspective. Although, the generic search engine makes attempts to streamline searching to search-key by traversing and combing the whole bunch of stored information. Under these circumstances, the system turns out to be inefficient and many times the results do not tally to what users desire. This paper presents a system to find interesting textual content among tons of documents taking the department of computer science of the University of Abuja, Nigeria as a case study. The paper proposed a vector space ranking algorithm which is a content-based ranking method that allows the user to utilize the full co-occurrence matrix of all words in the corpus to bring out relevant material using a simple and structured query interface. The index structures of the system have been specifically designed to support the ranking scheme. One important aspect of the system is its flexibility and robustness that makes it adaptable to any domain rather than being tailored to a particular one.

Keywords: *Vector space ranking, indexing, information retrieval, domain-specific, query interface.*

1. INTRODUCTION

A domain-specific search engine is the one caters to a specific audience (or specific needs of a general audience) while offering high quality search within a particular domain [1]. This is contrary to generic search engine that adopted a "one size fits all" strategy in searching for information on WWW. By domain-specific, it could mean information to a specific audience such as students of computer science department that need study materials specific to their studies from the library; it could also mean specific information to a general audience such as students accessing system that provides only study materials meant for general studies like “Use of English”. Inkpen (2006) stressed that searching for information may take two ways: search engines such as Google search engine and directories organized by categories such as Yahoo Directories. Web search engines help users find useful information on the WWW. Web search engines collect data from the Web by “crawling”, specific search engines are based on focused crawlers, which collect only the documents related to the given topics of interest [2].

Most personal computer systems today provide hierarchical, directory-based naming that allows users to place each file along a single, unique path. Although useful on a small scale, having only one classification for each file is unwieldy for large data sets. When traversing large hierarchies, users may not remember the exact location of each file. To make these systems viable, they must automatically classify the user's files, and, in fact, this requirement has led most systems to employ search tools over hierarchical file systems rather than change their underlying methods of organization [3]. Users need more effective ways of organizing and searching for information. It has become increasingly difficult for users to find information on the World Wide Web that satisfies their individual needs since information resources on the World Wide Web continue to grow. The reason is because World Wide Web provides a bunch of information that is generic in perspective.

Another problem area posed by information searching and retrieval is that specific information to general audience and vice versa need not necessarily be on generic search engine or on the WWW. It could be implemented on the intranet or local area network, for efficiency and maintainability. This paper is therefore aim at developing a domain-specific search engine taking computer science department of University of Abuja as a case study. The proposed system adopted a full stack information retrieval technique which includes query development, indexing and ranking. The paper presents a content-based, vector space ranking method that accurately ranked documents based on the query term. Finally, in future, the search engine may be implemented on top of the eLibrary system and specifically address the needs of the students for easy searching and retrieval of course materials that is specific to that department. Domain-specific search engine has great benefits compare to generic search

engine such as efficient searching and retrieval of information, maintainability, easy updating and removal of information just to mention few. Furthermore, domain-specific search engine is applicable in many applications both online and offline such as Google desktop, application-specific search engine etc.

2.0 REVIEW OF LITERATURE

2.1 Information Retrieval (IR)

Mohamed et al (2014) described information retrieval (IR) as the task of representing, storing, organizing, and offering access to information items. It is different from data retrieval which is about finding precise data in databases with a given structure. IR is the field of study that has to do with finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). Information retrieval is fast becoming the dominant form of information access, overtaking traditional database style searching [5].

The term “unstructured data” refers to data which does not have clear, semantically overt, easy-for-a-computer structure. It is the opposite of structured data, the canonical example of which is a relational database. IR is also used to facilitate “semistructured” search such as finding a document where the title contains Java and the body contains threading. The field of information retrieval also covers supporting users in browsing or filtering document collections or further processing a set of retrieved documents.Given a set of documents, clustering is the task of coming up with a good grouping of the documents based on their contents. It is often approached by first manually classifying some documents and then hoping to be able to classify new documents automatically.

Information retrieval systems can also be distinguished by the scale at which they operate [5], and it is useful to distinguish three prominent scales as given below.

2.1.1 Web search

Categorized at the extreme, the system has to provide search over billions of documents stored on millions of computers. Distinctive issues are; needing to gather documents for indexing, being able to build systems that work efficiently at this enormous scale, and handling particular aspects of the web, such as the exploitation of hypertext and not being fooled by site providers manipulating page content in an attempt to boost their search engine rankings, given the commercial importance of the web.

2.1.2 Personal information retrieval

Categorized at the other extreme, as information retrieval tool integrated in consumer operating systems (such as Apple’s Mac OS X Spotlight or Windows Vista’s Instant Search). Distinctive issues here include handling the broad range of document types on a typical personal computer, and making the search system maintenance free and sufficiently lightweight in terms of startup, processing, and disk space usage that it can run on one machine without annoying its owner.

2.1.3 Enterprise, institutional, and domain-specific search

It is categorized in between the previous two, where retrieval might be provided for collections such as a corporation’s internal documents, a database of patents, or research articles on biochemistry. In this case, the documents will typically be stored on centralized file systems and one or a handful of dedicated machines will provide search over the collection.

2.2 Search Engine Implementation Approaches

2.2.1 Directory-based Approach

Craig & Gregory (2005) noted that most personal computer systems today provide hierarchical, directory-based naming that allows users to place each file along a single, unique path. Although useful on a small scale, having only one classification for each file is unwieldy for large data sets. When traversing large hierarchies, users may not remember the exact location of each file. Or, users may think of a file in a different manner than when they filed it, sending them down an incorrect path in the hierarchy.

2.2.2 Context-based Approach

Most users organize and search their data using context. A user, for example, may group files related to a particular task into a single directory, or search for a file by remembering what other files they were accessing at the time. These contextual relationships may be impossible to gather from a file's contents.

The popular Google search engine [6] uses the link structure in two primary ways. First, it uses the text associated with a hyperlink to guide content classifications for the linked site. Second, it uses PageRank, an algorithm that uses the link structure of the web to calculate the importance" of individual sites. Although successful for the web, these techniques face challenges in personal file systems because tagged, contextual links do not inherently exist in the file system. Craig & Gregory (2005) present a

DOI: 10.31695/IJASRE.2019.33651

paper that describes a searching tool called Connections, a tool that combines traditional content analysis with contextual relationships identified by temporal locality. Connections uses successor models to identify relationships between files, as they have successfully identified related files in other domains. The paper emphasizes that gathering context from temporal locality could enhance systems by providing additional contextual clues for classification. A few systems attempt to determine the user's current context to predict and pre-fetch potentially desired data.

In file systems, temporal locality can provide some of the contextual clues that are so readily available on the web. Almost all file systems use prefetching to hide storage latencies by predicting what the user will access next and reading it into the cache before they request it. Some prefetching schemes use temporal locality to correlate common user access patterns with individual user contexts. These systems keep a history of file access patterns using successor models: directed graphs that predict the next access based on the most recent accesses. If a sequence of accesses matches one of the stored successor models, the system assumes that the user's context matches this model, and prefetches the specified data.

Just as in current file system search, Mauldinn (1991) noted that early web search engines relied on user input (user submitted web page classifications) and content analysis (word counts, word proximity, etc.). More recently, two techniques have emerged that use the inherent link structure of the web to identify contextual links. The HITS algorithm presented by Kleinberg (1999) defines a sub-graph of the web using content search results, and then uses the link structure of the graph to identify the authority and hub nodes.

Another context-based approach seen in web search is personalization, using the user's current context to target search results. WebGlimpse proposed by Manber et.al (1997) took a first step toward personalized search using the concept of a neighborhood," the set of web pages within a certain hyperlink distance of a given page. Users could choose to search only within the current page's neighborhood, creating a directed search of potentially related pages based on the user's current context. More recent work has focused on targeting results to particular topics or interests, sometimes gathered from a user's recent activity. These systems remove results that do not relate to the user's current context, improving precision.

2.2.3 Content-based Approach

The simplest content-based search tools (e.g., UNIX tools find and grep) as described by Craig & Gregory (2005) scan the contents of a set of files for a given term (or terms), returning all hits. To speed this process, tools such as locate and Glimpse use indices to reduce the amount of data accessed during a search. These tools work on the premise that the provided keywords can narrow the resulting list of files to a more human-searchable size.

Commercial systems such as Google Desktop and X1 desktop search leverage the work done in text-based information retrieval to provide more accurate, ranked search results. Although the exact methods of commercial systems are unpublished, it is likely they use techniques similar to cutting-edge information retrieval systems. These modern tools use probabilistic models to map documents to terms, providing ranked results that include both full and partial matches. Probabilities are generated using methods such as term frequency within a document, inverse term frequency across a collection, and other more complex language models.

2.2.4 Image Retrieval

Alex & Peter (2008) present a work, which considers a probabilistic model for content-based image retrieval which takes into account noise from user feedback. An algorithm based on binary search with noise is proposed and evaluated with experiments using both synthesized data and real data. The work extends these ideas and builds a practical search algorithm using high-level and low-level features.

Traditionally, content-based image retrieval with user feedback is considered a learning problem using data from user feedback and, with visual features most previous work assumes that no label describing images in datasets is available. Metric functions measuring similarity based on low-level visual features are obtained by discriminative methods. Long-term learning is used with training datasets from the feedback of different users. However, because of different perceptions about the same object, different users may give different kinds of feedback for the same query target. Short-term learning using feedback from a single user in a single search session can be used to deal with the different perceptions of objects. Weighting the importance of different low-level features is often used for short-term learning.

According to Alex & Peter (2008), suppose we have a database *D* of images x for image retrieval, and one can measure how close any given image x in the database is to a suitable image I, using a similarity function $\Phi(x, I)$. One also assumes that there is a limit N on the number of images presented to the user in each iteration, because of users' inability to handle a large amount of data.

DOI: 10.31695/IJASRE.2019.33651

Let *T* be the number of iterations required for the retrieval of a suitable image, and let St be the image subset presented in iteration t, $S_t \subseteq D$. As the number of user responses determines the retrieval costs, our objective is to minimize T. Thus, one may consider the following noisy feedback model as described by Alex & Peter (2008): image x ∈ S is selected with probability

$$P(x \text{ is selected}) = (1 - \alpha \cdot |S|)\frac{\Phi(x, I)}{\sum_{y \in S} \Phi(y, I)} + \alpha \quad (1)$$

where I is a suitable image, Φ is a similarity measure, and is a constant noise rate. Possible similarity measures are

$$\Phi(\mathbf{v}, \mathbf{I}) = \mathbf{v} \cdot \mathbf{I}, \quad (2)$$

And

$$\Phi(\mathbf{v}, \mathbf{I}) = \exp(-a||\mathbf{v} - \mathbf{I}||^2). \quad (3)$$

Figure 2.2 shows the algorithmic implementation of model 1 and 2 using weighted K-means for clustering with discounts given to the weights according to the user feedback as presented by Alex & Peter (2008).

Input: The images x in the database D, the similarity measure Φ, the relevance factor $\beta > 1$, and the number of images N to be presented in each iteration
Output: a suitable image I
Initialize all relevance weights, $w_x = 1$.
for $t = 1, 2, \ldots$ **do**
  Calculate cluster centers $c_1, \ldots, c_N \in D$ by weighted K-means, based on Φ and the weights w_x.
  Present images $c_1, \ldots, c_N$ to the user
  if *one of the images is suitable* **then**
    Stop
  end
  Let c_i be the image selected by the user as most relevant.
  For any image x which is more similar to c_i than to any other images $c_1, \ldots, c_N$, update the relevance weight, $w_x = \beta \cdot w_x$.
end

Figure 2.2: Algorithm for content-based image search
Source: [10]

2.2.5 File Type Identification

The identification of file types (e.g., ASP, JPG and EXE) is an important, but non-trivial, task that is performed to recover deleted file fragments during file carving [11 – 12]. File carving searches a drive image to locate and recover deleted and fragmented files. Since the file extension and magic numbers can easily be changed, file type identification must only rely on the file contents. Existing file type identification approaches generate features from the byte frequency distribution of a file and use these features for classification as explained [13]. The problem is that this process requires considerable time and memory resources because it scales with file size and the number of n-gram sequences.

In a work by Irfan et.al, (2011), a content-based file type identification schemes was proposed which typically use a byte frequency distribution with statistical machine learning to classify file types. Contrary to most algorithms that analyze the entire file content to obtain the byte frequency distribution, a technique that is inefficient and time consuming, the paper proposes two techniques for reducing the classification time. The first technique selects a subset of features based on the frequency of occurrence. The second speeds up classification by randomly sampling file blocks. Experimental results demonstrate that up to a fifteen-fold reduction in computational time can be achieved with limited impact on accuracy. Experimental tests of the two proposed techniques involve six classification algorithms: artificial neural network, linear discriminant analysis, k-means algorithm, k-nearest neighbor algorithm, decision tree algorithm and support vector machine.

Given a training set with instances and class-label pairs (x_i, y_i) where i = 1, 2, . . . , m and $x_i \in R^n$, $y_i \in \{1, -1\}^m$, the function φ maps the training vector xi to a higher-dimensional space using a kernel function to find a linear separating hyperplane with a maximal margin.

Since the SVM is a binary classifier, the one-versus-one approach is used for multiclass classification. Thus, r(r − 1)/2 binary classifiers must be constructed for r file types. Each binary classifier was trained using data corresponding to two file types. The final classification was determined based on the majority vote by the binary classifiers.

DOI: 10.31695/IJASRE.2019.33651

2.2.6 Document File Retrieval

There are various systems available for retrieval of Audio, Video and for Image. Shubham et.al, (2017) argued there was no currently available system like file retrieving using content based search. The work introduces the system for higher levels of retrieval of files/folders which needs only the contents of files/folders. The work targets not only for use of simple features like docs file, text file, but also use the conversion of PDF to docs, PDF to texts.

In content-based file retrieval (CBFR), the retrieval system typically contains two mechanisms: similarity measurement and multidimensional indexing. Similarity measurement is used to find the most similar files. Multidimensional indexing is used to accelerate the query performance in the search process. CBFR involves the subsequent four parts in system realization, data collection, build up feature database, search in the database, arrange the order and deal with the results of the retrieval [15].The technology is relevant to a number of application areas. The examples show the power this capability can bring to a user working in these areas.The selection Features is one of the significant aspect of File Retrieval architecture to better capture users Intention. It will display the files from the database which are more attention to the user CBFR gives better performance than the Existing system. Therefore challenge in visible of theCBFR structure is time saving complexity and efficient system [15].

2.2.7 Domain-specific approach

Mohamed et.al, (2014) presented a domain-specific web search engine which is justified by the fact that there is a huge quantity of text, audio, video, and other documents available on the Internet, on about any subject. The paper describes the proposed domain specific search engine based on Depth Limited Search (DLS) and Vector Space Model (VSM) algorithms with specialized content of crawled pages. The methodology used in the paper suggests domain-specific search engine using focused crawler depends on DLS and VSM algorithms. This leads to information retrieval accurately to resolve the content quality problem.

A domain-specific search engine cannot and will not be used if the search engine does not contain crawled content of its entire domain. Despite restricting itself to one domain, if it is not able to serve every query in that domain, it is a useless task to build one. Hence crawl diversity is one of the crucial factors that impact the quality of domain-specific search. To a large extent, crawl diversity depends on the choice of seed URLs (the list of URLs that the crawler starts with). Though significant effort has gone into building various crawling strategies, not enough research has been done in choosing good quality seed URLs.

Pattisapu et.al, (2014) proposed a seed selection for domain-specific Search in which they automate this process using URLs posted on Twitter. In the work, the problem of diverse seed selection is posed as a graph search problem. They form a graph where each vertex is a URL shared on Twitter. Two vertices are connected if they are "similar". An algorithm is proposed to get a set of diverse seed URLs from a Twitter URL graph. They compare the performance of the approach against the baseline zero similarity seed selection method and find that the approach beats the baseline by a significant margin.

3.0 RESEARCH METHOD

The method begins with a notion that users start with information needs, which they translate into query representations. Similarly, there are documents, which are converted into document representations (the latter differing at least by how text is tokenized, but perhaps containing fundamentally less information, as when a non-positional index is used). Based on these two representations, a system tries to determine how well documents satisfy information needs.

3.1 System Development Plan

Our corpus (document collection) is a file containing collection of descriptions of book/articles. To simplify our work and focus on the core algorithms, we will not write the crawler to get the file because the proposed system is not to be implemented on the World Wide Web. The file contains approximately 100 articles from various fields ranging from computer science to arts, of course more can be added later. The document looks XML like structure in the collection. Every document lies between <book> and </ book > tags, where the docID, title, and text is separated by the corresponding tags. We will write our own routine to parse this structure by using regular expressions. We will assume that these special tags won't appear in the body of the articles. However, the articles may contain non-alphanumeric characters and capital letters that we will pay special attention to while building the index. For example we don't want to index code, Code, and CODE differently. They are basically all code. When we search for one of these terms we would expect results containing any of those variations. Additionally, we would not like to index stop words such as 'the', 'a', 'an' because they appear in almost every document and they don't give us very much information about the document or the query.

3.2 Search and Rank Algorithm

Considering the fact that a good search engine does not attempt to return the document that best match the input query, a good search engine tries to answer the underlying question using certain algorithm. The factors in the algorithm consist of "hard

DOI: 10.31695/IJASRE.2019.33651

factors" but not like generic search engine like google.com. The factors considered here are in-site factors such as title, description etc. to explain in detail the applied vector space algorithm, the following parameters are considered;

- **Token:** A token is any single term in a text. It is the basic atom of search which is a whitespace-delimited substrings determined by tokenizer. A document is aN_d-length finite sequence of tokens.

$$d = (t_1, \dots, t_{N_d}) \qquad (1)$$

- **Query terms (Qt)**: each separate term in the query.
- **Term Frequency** (TF): This measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

$$TF(t) = \frac{\text{No of times term t appears in a document}}{\text{Total No of terms in the document}} \qquad (2)$$

- **Inverse Document Frequency (IDF):** this measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$$IDF(t) = \log\_e(\frac{\text{Total No of document}}{\text{No of documents with term t in it}}) \rightarrow IDF(t, D) = \log(\frac{N}{n_t}) \qquad (3)$$

- **Tf-idf** stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.
- **Document Frequency** (**df**): the number of times a term appears in a document.
- **Total document (Tt)** incorpus, for simplicity in our experiment was taken to start with three documents of varied length.

Other parameters that will be computed are;

- **Wt,q** = weight for term in query
- **Wt,d** = weight for term in document
- **Product** = Wt,q * Wt,d
- **Score** = Sum of the products

3.3 Vector Scores Computing

The representation of a set of documents as vectors in a common vector space is known as the *vector space model* and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering. We first develop the basic ideas underlying vector space scoring; a pivotal step in this development is the view of queries as vectors in the same vector space as the document collection.

In a typical setting we have a collection of documents each represented by a vector, a *free text query* represented by a vector, and a positive integer *K*. We seek the *K*documents of the collection with the highest vector space scores on the given query. We now initiate the study of determining the *K*documents with the highest vector space scores for a query.

float Score[N] 0
Initialize Length[N]
for each query term t
do calculate $w_{t,q}$*and fetch document list for t*
    for each pair(d, $tf_{t,d}$*) in document list*
    do Scores[] += $wf_{t,d}$ *x* $w_{t,q}$
Read the array Length[d]

DOI: 10.31695/IJASRE.2019.33651

For each d
do Score[d] = Score[d]/ Length[d]
return Top K components of Scores[]

Algorithm 3.1: Vector Score computing algorithm

3.4 System Design

The proposed system is illustrated in Figure 3.1 with the components which correspond to software modules developed independently but interconnected.

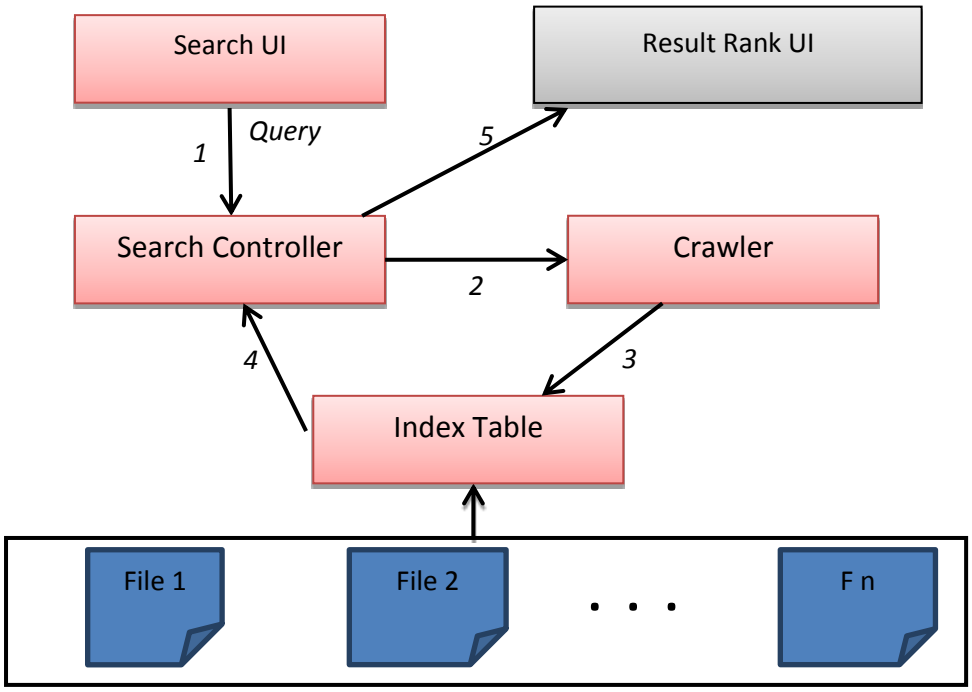

Figure 3.2: The design of the proposed Domain specific search engine

At every point in time the system count the number of documents make up the corpus to update the index Table. The content of each document is crawled to search for the query term. If found, the number of the query term found in each document is noted. The result of the query is ranked, co-ordinated by search controller and sent to the result user interface.

3.5 System Implementation Plan

The implementation of the proposed system is carried out in module in consideration of the components. The components and the corresponding programming language are briefly discussed below;

- **User Interface**: - This comprises the Form element and the search result interface. The form element allows user to enter the search terms and the container element that shows the searched results. User interface is developed with HTML, CSS and a bit of PHP.
- **Search Controller File:** - This is a JavaScript file that collects the search query from the Search Form, pre-processed the query, send the query to crawler file and finally receive the result of the search from the index file. The algorithm 3.1 discussed above is implemented to compute the ranking of the documents that contain the search term. The list of the document is displayed according to the rank.
- **Page File:** - This is a JavaScript Object Notation (JSON) file that contains elements for each search domain such as id, title, description and location.
- **Lunr**: - This is crawler file written in JavaScript. The function is to scan document path.

  Index File: - This is the place where the documents are stored in an inverted tree

DOI: 10.31695/IJASRE.2019.33651

4.0 EXPERIMENT AND RESULT DISCUSSION

To experiment with the algorithm described above, we have three documents named doc 1, doc 2 and doc 3. These documents are word document prepared at random. The aim is to show that some document with more contents may not be relevant. Therefore, vector space score is adopted to compute the score of document in order to determine its relevance based on the term.

For simplicity, word is used to represent token

Doc1: 3 page - 817 tokens

Doc2: 4 page - 1179 tokens

Doc3: 2 page - 1540 tokens

The score for the documents would be as follows:
score("information, and, approach", Doc1) = 16 + 28 + 2 = 46
score("information, and, approach", Doc2) = 8 + 32 + 5 = 45
score("information, and, approach", Doc3) = 7 + 8 + 7 = 22

Doc1 and Doc2 are in this case closer related to the query. In this example the term "and" gains the most weight, but is this fair? It is a stop word, and we like to give it only a little value. We can achieve this by using inverse document frequency (tf-idf), which is the opposite of document frequency (df). Document frequency is the number of documents where a term occurs. Inverse document frequency is, well, the opposite. As the number of documents in which a term grows, idf will shrink.

IDF can thus be calculated by dividing the total number of documents you have in your corpus by the number of documents containing the term and then take the logarithm of that quotient. The detail report of the study shown in Table 4.1

Table 4.1: IDF computation reports

Doc1	**Query**			**Document**			**Product**	**Score**
Term		**idf**	**Wt,q**	**tf**	**Wf^2**	**Wt,d**		
information	1	0.805912798	0.805912798	16	256	0.89443	0.72083032	0.72083032
and	1	0.614428097	0.614428097	8	64	0.44721	0.274780599	0.274780599
approach	1	1.74287549	1.74287549	7	49	0.39131	0.682007913	0.682007913
							Total Score:	0.995610919
Doc2	**Query**			**Document**			**Product**	**Score**
term		**idf**	**Wt,q**	**tf**	**Wf^2**	**Wt,d**		
information	1	0.805912798	0.805912798	28	784	0.65850	0.530697291	0.530697291
and	1	0.614428097	0.614428097	32	1024	0.75258	0.462404267	0.462404267
approach	1	1.74287549	1.74287549	8	64	0.18814	0.327911869	0.327911869
							Total Score:	0.993101558
Doc3	**Query**			**Document**			**Product**	**Score**
term		**idf**	**Wt,q**	**tf**	**Wf^2**	**Wt,d**		
information	1	0.805912798	0.805912798	2	4	0.37139	0.299308499	0.299308499
and	1	0.614428097	0.614428097	5	25	0.92848	0.570482167	0.570482167
approach	2	1.74287549	3.48575098	7	49	1.29987	4.531013949	4.531013949
							Total Score:	0.869790666

Summary

	Scores (higher = better)		**Query Terms**	**Document Frequency**	**Doc1 term frequency**	**Doc2 term frequency**	**Doc3 term frequency**
Doc1	0.9956109	**Term 1**	information	16,260,000,000	16	28	2
Doc2	0.9931016	**Term 2**	and	25,270,000,000	8	32	5
Doc3	0.8697907	**Term 3**	approach	1,880,000,000	7	8	7

DOI: 10.31695/IJASRE.2019.33651

In Table 4.1, Doc1 has 0.9956109 which is the highest score. This means the relevancy of the keyword "information, and, approach" is very high in Doc1. Vector space model is adopted for the ranking which helps to solve the problem arise whereby one document contains much more content than another document, without being more relevant. Figure 4.1 shows a sample of the JSON Page file that consists of information of each page as already discussed in Section 3.8.The source codes for the system are attached to the Appendices.

Figure 4.2 below shows the sample of result of searching term "information" using the proposed system. The result shows that the term exists in three documents and the rank is the order displayed.

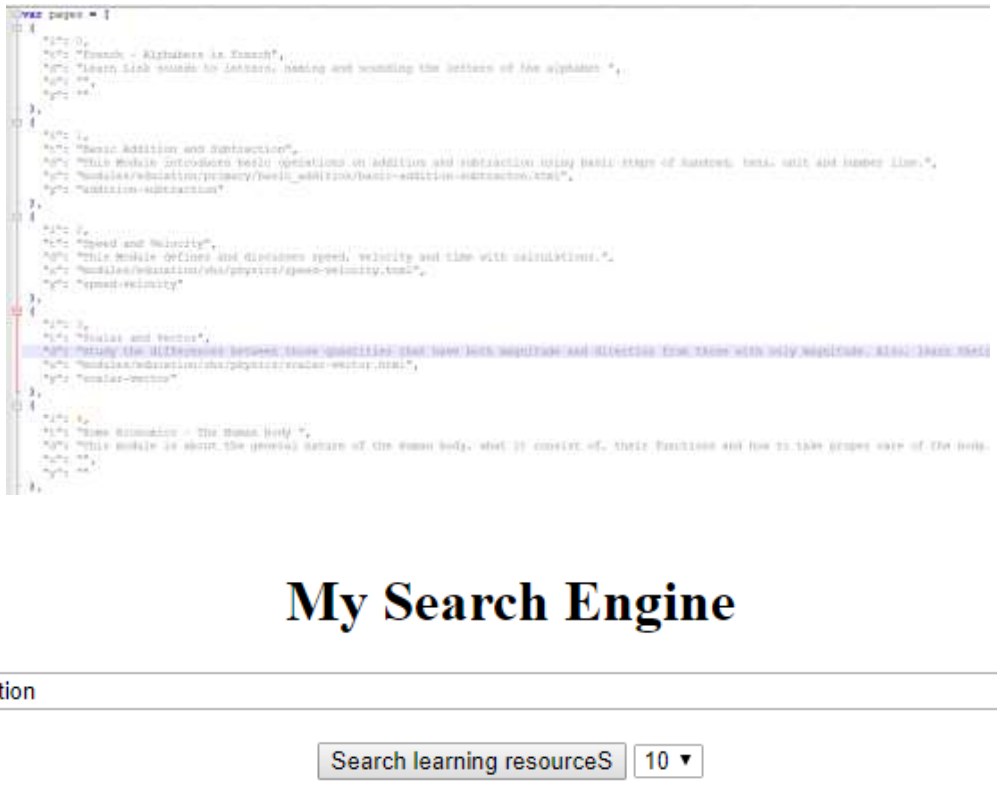

Figure 4.1: JSON Page file

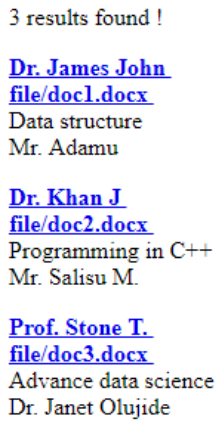

Figure 4.2: Search result

5.0 CONCLUSION

In this work, it has been shown that the algorithm and techniques used in popular search engine for the web can also be applied to develop a domain specific search engine that can be used even for offline documents while still provides same functionality such as ranking and searching. It has also been shown that a good search engine should not just provide document that contains a search term but also provide the document based on how relevant the term is to the document.Vector space model is adopted for the ranking which helps to solve the problem arise whereby one document contains much more content than another document,

DOI: 10.31695/IJASRE.2019.33651

without being more relevant. It is based on statistical computation of a value; the higher the value the higher the rank of the document.

Since the algorithms used by the popular search engine such as Google is not made public, it therefore recommended intensive use of domain-specific search engine since the algorithms are open for improvement by the members of users.

Furthermore, it is believed that by further utilizing distributional features of data and the scoring functions, efficiency of the system may be improved drastically. Also,it is noted that the computational load could be reduced by approximating the results instead of computing exact scores and ranks for the documents.

Finally, it is important to point out in this work is that the indexing is done manually. This means that any time the Page file is updated, the index builder has to be run to re-generate the index file. This process is not efficient considering indexing large file. herefore, the future work will see to automate the process of indexing.

REFERENCE

1. Pattisapu N. P., Ajay D., Krish P., Sai P., Dharmesh K., Vasudeva V. (2014). Seed Selection for Domain-Specific Search.International World Wide Web Conference Committee (IW3C2).ACM 978-1-4503-2745-9/14/04.
2. Inkpen D. (2006).Information Retrieval on the Internet.[Web log post] Retrieved February 1, 2018, fromhttp://www.site.uottawa.ca/~diana/csi4107/IR_draft.pdf
3. Craig A. N., Gregory R. G. (2005). Connections: Using Context to Enhance File Search. SOSP'05, October 23?26, 2005, Brighton, United Kingdom. Copyright 2005 ACM 1¬59593¬079¬5/05/0010
4. Mohamed. M. E., Elsaeed. E. A, Doaa. M. H. (2014). Building Domain Specific Search Engine Based on DLS and VSM Algorithms. (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3), 2014, 4546-4551
5. Christopher D. M., Prabhakar R. and Hinrich S. (2009).An Introduction to Information Retrieval. Cambridge University Press Cambridge, England
6. Brin S. and Page L. (1998). The anatomy of a large-scale hypertextual web search engine.Computer Networks and ISDN Systems, 30(1) 7:107-117.
7. Mauldin M. L. (1991).Retrieval performance in Ferret a conceptual information retrieval system.ACM SIGIR Conference on Research and Development in Information Retrieval, pages 347{355. ACM Press, 1991.
8. Kleinberg J. M. (1999). Authoritative sources in a hyperlinked environment. Journal of the ACM, 46(5):604{632. ACM, September 1999.
9. Manber U., Smith M., and Gopal B. (1997). WebGlimpse: combining browsing and searching. USENIX Annual Technical Conference. USENIX Association, 1997
10. Alex P. L. and Peter A. (2008).An Efficient Search Algorithm for Content-Based Image Retrieval with User Feedback. Department fur Mathematik und Information technologieMontanUniversit¨atLeoben Franz-Josef-Straße, ustria.
11. Garfinkel S. (2007). Carving contiguous and fragmented files with fast object validation, Digital Investigation, vol. 4(S1), pp. 2–12, 2007.
12. Roussev V. and Garfinkel S. (2009). File fragment classification – The case for specialized approaches, Proceedings of the Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, pp. 3–14, 2009.
13. Li, K. W., Wang, S. S. and Herzog B. (2005).Fileprints: Identifying file types by n-gram analysis, Proceedings of the Sixth Annual IEEE Information Assurance Workshop, pp. 64–71, 2005.
14. Irfan A., Kyung-Suk L., Hyun-Jung S. and Man-Pyo H. (2011).Fast Content-Based File Type Identification.
15. Shubham B., Pranmay J., Soumadwip D. and Dipali T. (2017).International Journal of Innovative Research in Computer and Communication Engineering, An ISO 3297: 2007 Certified Organization. Vol. 5(2).

DOI: 10.31695/IJASRE.2019.33651