ISSN:0975-9646

Similarity Measures of Web Repositories constructed by Focused Crawling from Database Driven Websites

J.Tamilselvan[1], Dr. A.Senthilrajan[2]
[1]*Assistant Professor, Department of Computer Science*
[2]*Director, Computer Centre*
[121]*K.S.Rangasamy College of Arts and Science (Autonomous)*
Tiruchengode, Tamilnadu, India
[2]*Alagappa University, Karaikudi, Tamilnadu, India*

Abstract: Intelligent systems require knowledge-rich resources. The most important tasks in information extraction from the web are webpage structure understanding as many web sites contain large collections of pages generated using a common template or layout makes it increasingly difficult to discover relevant data about a specific topic. Extracting data from such pages has become an important issue in recent days as the number of web pages available on the Internet has growing in exponential. Tools and protocols to extract all this information have now come in demand as researchers and web surfers want to discover new knowledge at an ever increasing rate. A web crawler also known as, a robot or a spider is a system for the bulk downloading of web pages, whereas the goal of a focused crawler is seeking pages that are relevant to a pre-defined set of topics. The topics are specified not using keywords, but using exemplary documents. Instead of collecting and indexing those accessible web documents which can answer all ad-hoc queries, a focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web. Since all search engines take their data fed using crawlers, it is critical to improve its working ability. As the size of data is huge, Common Crawlers are no longer applicable in real life. So there is need to develop a domain specific crawler builds on stock of existing algorithms. This led to considerable savings in hardware and network resources, and helps keep the crawl more up-to-date. This paper proposed a novel framework called StructWebNLP, which enables bidirectional integration of page structure understanding and text understanding in an iterative manner. We have applied the proposed framework to the judgments information system to extract text of judgments.

Keywords: ***Web Crawler, Text Extraction, Structured Web Data, Deep Web.***

1. INTRODUCTION

We observe an ever-increasing quantity of structured data available on the Web and, in the same instance, the diversity of the visual structures in which the data are stored. The prime samples of such structured data is the Deep Web, which refer to the content on the Web that is stored as databases and respond by querying HTML forms [Madhavan et al. 2007]. There is an enormous potential in combining and re-using this data in creative ways. We have studied the problem of automatically extracting database values from such a collection of web pages automatically with less human participation. We explain a working model of a web-scraping program developed using the Scrapy framework (written in Python). The program was framed to acquire an excellent deal of public archival of information presently available only in the form of web page HTML source code, and to acquire new data as it is added to the online repository. The study involves text extraction from the web pages that will give an outline of a program that scrapes and processes the previous collection of data. A crawler is a program that retrieves and save pages from the Web, commonly for a Web search engine. A crawler often has to download hundreds of millions of pages in a short period and has to constantly monitor and refresh the downloaded pages. In addition, the crawler ought to avoid putting too much stress on the visited Web sites and the crawler's local network, because they are fundamentally shared resources. A Web crawler is a program that downloads Web pages, for a Web search engine or a Web cache. Roughly, a crawler starts with an initial URL S_1. It first places S_1 in a queue, where all URLs to be retrieved are kept and prioritized. Spidering a website, link by link, will work for most of the websites. However, it can be a kind of tedious to examine each different kind of page to figure out the link structure. We can do a little survey and experimentation and may find a pattern in the site's URL that we use to save ourselves a considerable amount of time. The most obvious examples are sites that obtain information from database displays the content with a fixed template and dynamic tag paths. User selects URLs of his/her own interest and identifies the tag paths of each Data Region in a Web page or the tag paths of desired data, the only manual work we need to do is to traverse the HTML tree and collect x-path where relevant information kept.

A. Hypertext Crawler

The basic operation of any hypertext crawler is as follows. The crawler begins with one or more URLs that constitute a seed set. It picks a URL from this seed set, and then fetches the web page at that URL. The fetched page parsed, to extract both the text and the links from the page. The extracted text fed to a text indexer. The extracted links (URLs) added to a URL frontier, which at all times consists of URLs whose corresponding pages have yet to fetch by the crawler. Initially, the URL frontier contains the seed set; as pages fetched, the corresponding URLs deleted from

the URL frontier. The entire process may treat as traversing the web graph. In continuous crawling, the URL of a fetched page added back to the frontier for fetching again in the future if needed. This is a simple traversal of the web graph, which is complicated by the many demands on a practical web crawling system, the crawler has to fulfill certain features while fetching pages of high quality.

B. Features of Crawler

Distributed: The crawler should have the ability to execute in a distributed fashion across multiple machines.
Scalable: The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth.
Performance and efficiency: The crawl system should make efficient use of various system resources including processor, storage, and network bandwidth.
Quality: Given that a significant fraction of all web pages are of poor utility for serving user query needs, the crawler should be biased towards fetching "useful" pages first.
Freshness: In many applications, the crawler should operate in continuous mode it should obtain fresh copies of previously fetched pages. A search engine crawler, for instance, can thus ensure that the search engine's index contains a current representation of each indexed web page.
Extensible: Crawlers has designed to be extensible in many ways to cope with new data formats, new fetch protocols, and so on. This demands that the crawler architecture be modular.

C. Behavior of Crawler

The behavior of a Web crawler is the outcome of the combination of policies.

D. Selection Policy

A crawler always downloads just a fraction of the web pages; it is highly desirable that the downloaded fraction contains the most relevant pages and not just a random sample of the Web. Some selection policies are

a. Restricting followed links
b. Path ascending crawling
c. Focused crawling
d. Crawling the deep web

E. Focused Crawler

There are various uses of web crawler, but essentially a web crawler used by anyone seeking to collect database out of the internet search engines. Frequently web crawlers used to collect information about what is available on public web pages. Their primary purpose is to collect data so that search term on their site will quickly provide the surfer with relevant web sites. Linguistics may use a web crawler to perform a textual analysis that is, they may comb the internet to determine what words are commonly used today [1]. A standard crawler crawls through all the pages in breadth first strategy. When we want to crawl through some domain then it will be very inefficient. In Fig. 1 we show the general crawler crawling activity [2]. In Fig. 2 we can see that a focused crawler crawls through domain specific pages. The non-related pages of a particular domain not considered [2].

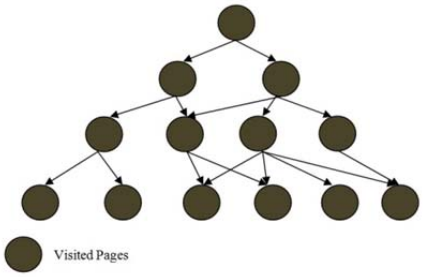

Fig. 1. Basic Crawling

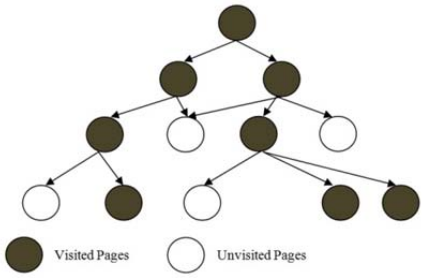

Fig. 2. Focused Crawling

2. RELATED WORKS

Chakrabarti, Berg and Dom [3] described a focused web crawler with three components, a classifier to assess the web page significance to the chosen topic, a distiller to discover the relevant nodes using few link layers, and a reconfigurable crawler that is governed by the classifier and distiller. They try to impose various features on the designated classifier and distiller: Explore links in terms of their sociology, extract specified web pages based on the given query, and explore mining communities (training) to improve the crawling ability with high quality and less relevant web pages.

Sk.Abdul Nabi et al [4] addressed domain based information system in which system crawl the information from the web and added all links to the database related to specific domain. Because of that searching space and searching time reduced and as a result, it improves the performance of the system. They use pattern-matching algorithm in which input is given as rank table of web page and then total rank is calculated. Web is very large and dynamic so searching the required relevant content from web is very difficult. Grouping the collection from the web is always challenging. Therefore, there is need to gather from broad range of domains.

Scott Deerwester et al [5] found a new method for automatic indexing and retrieval. The method has designed to overcome a fundamental problem that outbreaks existing retrieval techniques that try to match words of queries with words of documents. The problem is that users want to retrieve based on conceptual content, and individual words provide unreliable evidence about the conceptual topic. There are usually many ways to express a given content, so the literal terms in a user's query may not match those of a relevant document. In addition, most words have multiple meanings, so terms in a user's query will literally match terms in documents that are not of interest to user. The proposed approach use statistical technique to estimate the

latent structure, and get rid of the obscuring “noise”. A description of documents terms based on the latent semantic structure used for indexing and retrieval.

Radhika Gupta et al [6] developed a semi-deterministic algorithm and scoring system that takes advantage of the Latent Semantic Index scoring system for crawling web pages that belong to particular domain or specific to the topic. The proposed algorithm calculates a preference factor in addition to the LSI score to determine which web page needs to chosen for precision values as it builds a queue that is specific to a particular domain/topic that would not have been possible in breadth first algorithm and only LSI based information retrieval systems.

Hong-Wei Hao et al [7] developed the improved topic relevance algorithm for focused crawling. Firstly, they implement a prototype system of the focused crawler. Second, experiments on Chinese text corpus shows that using latent semantic indexing outperforms using TF-IDF (term frequency-inverse document frequency) for hyperlink topic relevance prediction and pages topic relevance calculation. Third, in real crawling experiments on the prototype system, the crawler using TF-IDF has high performance with the accumulated topic relevance increasing quickly at the beginning of the crawling, however the crawler using LSI can find more related pages and TF-IDF, they proposed TFIDF+LSI performs the same crawl task and demonstrates the combination advantage of TF-IDF and LSI . However, due to the limitation of LSI and using only anchor text and other factors, topical crawler using TFIDF+LSI may still cause topic drift.

Ahmad Pesaranghader et al [8] proposed improved measure called Term frequency-Information Content (TF-IC) to prioritize terms in a multi-term topic. Through conducted experiments, we compare our measure against both Term frequency-Inverse Document frequency (TF_IDF) and Latent Semantic Indexing (LSI) measures applied in focused crawlers. Experimental results indicate superiority of measure over TF-IDF and LSI for collecting both more relevant web pages of general and specialized multi-term topics.

M. Diligenti et al [9] presented a focused crawling algorithm that builds a model for the context within which topic relevant pages occur on the web. Because of the major problem in focused crawling which perform appropriate credit assignment to different documents along a crawl path, such that short-term gains are not pursued at the expense of less obvious crawl paths that ultimately yield larger sets of valuable contents. This context model can capture typical link hierarchies within which valuable pages occur, as well as model content on documents that frequently co-occur with relevant pages. The algorithm shows significant performance improvements in crawling efficiency over standard focused crawling.

3. SYSTEM ARCHITECTURE

The focused crawler has four main components: A **URL Iteration** continuously set the URL parameter value (Path tag) by incrementing the parameter values to obtain pages, a **Page Downloader to** download pages from web, a C**lassifier** to make relevance judgments and to decide on pages crawled, and a D**istiller** to determine a measure of centrality of crawled pages.

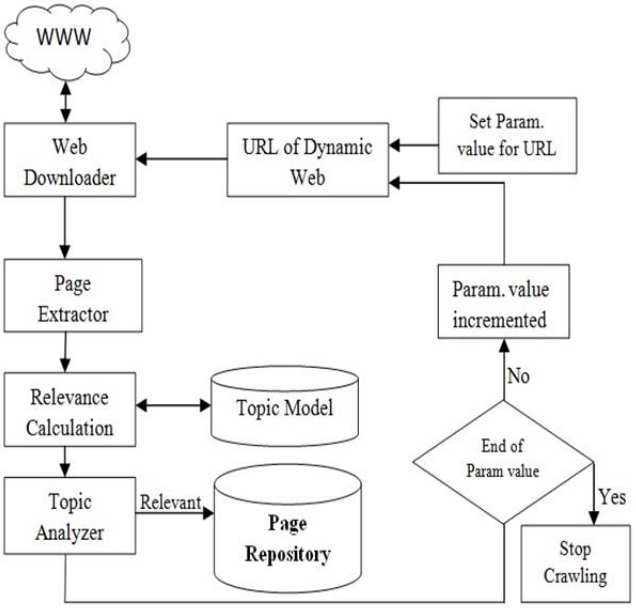

Fig. 3. Block Diagram of Focused Crawler

Here we briefly outline the basic processes. Fig. 3 shows the architecture of focused crawler.

A. URL Iteration

URL Iteration contains the address of a dynamic website and initialized with starting parameter value. The classifier analyzes whether the content of parsed pages related to topic or not. If the page is relevant, add the content extracted from it, to the page repository, otherwise discard it. There are a large number of URLs pointing to the same page the crawler is likely to be trapped [10]. A way to alleviate this problem is to limit the number of pages that a crawler can get from a domain name. In this paper, the solution is that URL Iteration has carried for crawling k (for example, k =2000) pages from same domain. Advantage of this approach is that the load of crawler to web servers reduced substantially.

B. Page Downloader

The page downloader fetches URLs from URL queue and downloads corresponding pages from the internet. In order to download a page, the page downloader contains a HTTP client used to send the HTTP request and read the response. The client needs to set the timeout to ensure that it will not take much time to read large files or wait for response of web servers.

Important consideration of a crawler is the Robot Exclusion Protocol [11]. The protocol provides a mechanism that a web server administrator can define file access policy used to indicate which files the crawlers should not be access. The method used to exclude robots from a server is to create a file on the server, and this file must be accessible via HTTP on the local URL "/robots.txt". Before a crawler download pages from a server, it must first obtain the appropriate robots.txt file and check whether the

downloading allowed or not. The files are cached to improve efficiency, which can avoid re-downloading the file when downloading pages from the same server. Our focused crawler follows the Robot Exclusion Protocol.

C. Relevance Calculation

A similarity/distance measure must be determined before clustering. The measure reflects the degree of closeness or separation of the target objects and should correspond to the characteristics that are believed to distinguish the clusters embedded in the data. In many cases, these characteristics are dependent on the data or the problem context at hand, and there is no measure that is universally best for all kinds of clustering problems.

4. EXPERIMENT

The main components of web search engines are systems that assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries. A related use is web archiving, where large sets of web pages periodically collected and archived for future use. Further use is web data mining, where web pages analyzed for statistical properties, or where data analytics performed on them. Finally, web-monitoring services allow their clients to submit standing queries, or triggers, and they continuously crawl the web and notify clients of pages that match those given queries.

Organization of Experiment

We briefly describe how we have organized the experiment. For each input collection of web pages that we used we present the following steps as part of achieving the experimental results.

1. *Source Pages:* Define the source pages in the collection.
2. *Detecting URL values:* Find the path tags that used as parameter values for fetching pages.
3. *Extracted Template:* The template deduce by our system for the collection.
4. *Extracted Schema:* The schema deduce by our system.
5. *Extracted Data:* The data encoded in each page that extracted by our system.
6. *Defining URL:* Set the path tags (URL values) for the sites along with base URL.
7. *Manual Schema:* We deduce the schema manually by using the semantics of the information in the pages for evaluating the system.

Scrapy Code

```
import scrapy
    class JudisTxt(scrapy.Item):
        title = scrapy.Field()
        link = scrapy.Field()
        desc = scrapy.Field()
```

We start by modeling the item to hold the site's data obtained from judis.nic.in. As we want to obtain the name, url and description of the sites, we define fields for each of these three attributes.

```
import scrappy
import os.path

class JudisSpider(scrapy.Spider):
   name = "judis"
   allowed_domains = ["judis.nic.in"]
   start_url = [
      "http://judis.nic.in/supremecourt/imgst.aspx?filename=1"
   ]
   def parse(self, response):
       for i in range(1,2000):
          filename = response.url.split("/")[-2] + str(i)+".html"
          with open(filename, 'wb') as f:
               uname = str(i)+".html"
               save_path = 'D:/jdocs/'
               completeName = os.path.join(save_path, uname)
               f.write(response.body)
```

The above code runs the spider with name *judis* that will send requests to the judis.nic.in domain then retrieves the page of which contains text document starting from the first page and reads subsequent pages. When the filename (URL parameter value) is incremented with the continuous values for *filename* it saves the continuous pages from one to two thousand in the prescribed local storage device location.

To extract textual information from the pages, we use the XPath.

```
response.xpath('//textarea()').extract()
```

4.1 Data Sets

This work experiments by giving the seed URL as http://judis.nic.in The Judgments Information system consists of the Judgments of the Top Court and several other Courts has 30,000+ datasets (Judgments). The path tag (URL parameter value) is incremented subsequently with the continuous values. It fetches the web page present at that URL by using the above given scrappy code. The fetched pages are then parsed, to extract both the text and the links if any from the page.

4.2 Evaluation

In order to measure the similarity of documents in the web repository, Top 400 documents returned by the above-mentioned algorithm will be used. Then a count of different URLs present will be prepared. This will be indication for site recommendation. In addition, pages of spam sites also identified. Minimum number of overlapping document, more relevant page, less traffic consume less bandwidth and most updated page storage are to be considered as far as this type of continuous ordering crawlers produce mere common contents as the crawler fetches from the same site of different contents. We use entropy as a measure of quality of the clusters.

4.3 Result Analysis

Text document clustering plays an important role in providing intuitive navigation and browsing mechanisms by organizing large amounts of information into a small number of meaningful clusters. Clustering method has to embed the documents in a suitable similarity space. While

several clustering methods and the associated similarity measures have been proposed in the past, there is no systematic comparative study of the impact of similarity measures on cluster quality. This may be because the popular cost criteria do not readily translate across qualitatively different measures. In this paper we compare four popular similarity measures (Euclidean, cosine, Pearson correlation and extended Jaccard) in conjunction with different types of vector space representation (boolean, term frequency and term frequency and inverse document frequency) of documents. Clustering of documents is performed using generalized k-Means; a Partitioned based clustering technique on high dimensional sparse data representing text documents. Performance measured against a human-imposed classification of Topic categories. We conducted a number of experiments and used entropy measure to assure statistical significance of results. Cosine, Pearson correlation and extended Jaccard similarities emerge as the best measures to capture human categorization behavior, while Euclidean measures perform poor.

In this work seed points are statically chosen, but efficiency can be improved if seeds selected are random or run the code more than once to check the efficiency. As shown in Table 1, Euclidean distance performs worst while the performance of other measures is quite similar. From our results it is observed that Boolean representation with Pearson measure has non-zero clusters. Hence the overall entropy for Boolean representation table shows NaN values for other measures as some of the clusters are empty. On an average, the Jaccard and Pearson measures are slightly better in generating more coherent clusters, which means the clusters have lower entropy scores.

Table 1: Entropy Results of Different Vector Space Representations Using Crawled dataset

	Cosine	Jaccard	Euclidean	Pearson
Boolean	NaN	NaN	NaN	0.07
Freq. Count	0.15	0.11	0.28	0.07
TF-IDF	0.07	0.08	0.28	0.07

The Euclidean distance proved as an ineffective metric for modeling the similarity between documents. The Jaccard and Pearson's coefficient tend to outperform the cosine similarity.

4.4 Performance Evaluation Measures

A human or crawler must identify web sites containing form interfaces that lead to deep web content. [12] The first step in resource selection is to model the content available at a particular deep web site, e.g., using query-based sampling [13]. Finally, a crawler must extract the content lying behind the form interfaces of the selected content sources. A query consists of a single data value submitted to the form interface, which retrieves the set of content items directly connected to the data. Each query incurs some cost to the crawler, typically dominated by the overhead of downloading and processing each member of the result set, and hence modeled as being linearly proportional to result cardinality.

5. CONCLUSION

In this study, we found that all the measures have significant effect on Partitioned clustering of text documents except for the Euclidean distance measurer. Pearson correlation coefficient is slightly better as the resulting clustering solutions are more balanced and is nearer to the manually created categories. The Jaccard and Pearson coefficient measures find more coherent clusters. Considering the type of cluster analysis involved in this study, we can see that there are three components that affect the final results—representation of the documents, distance or similarity measures considered, and the clustering algorithm itself. In our future work our intension is to apply semantics knowledge to the document representations to represent relationships between terms and study the effect of these similarity measures comprehensively.

REFERENCES

[1] Gautam Pant, Padmini Srinivasan1, Filippo Menczer, "*Crawling the Web*", Department of Management Sciences, The University of Iowa, Iowa City IA 52242, USA.

[2] Debajyoti Mukhopadhyay, Arup Biswas, Sukanta Sinha, "A *New Approach to Design Domain Specific Ontology Based Web Crawler*", West Bengal University of Technology, pp.70091.

[3] Chakrabarti, Soumen, Martin van den Berg, and Byron Dom. "*Focused crawling: a new approach to topic-specific Web resource discovery*", Elsevier, 1999.

[4] Sk.Abdul Nabi, Dr. P.Premchand, "*Effective Performance of Information Retrieval by using Domain Based Crawler*", Vol. 4, No.7, 2013.

[5] Scott Deerwester, Susan T. Dumais, George W. Furnas and Thomas K. Landauer, Richard Harshman, "*Indexing by Latent Semantic Analysis*", 41(6):391-407, 1990.

[6] Radhika Gupta, AP Nidhi, "*Focused Crawling System based in Improved LSI*", Volume 2 Issue 9, September 2013.

[7] Hong-Wei Hao, Cui-Xia Mu, Xu-Cheng Yin, Shen Li, Zhi-Bin Wang, "*An Improved Topic Relevance Algorithm for focused Crawling*".

[8] Ali Pesaranghader, Ahmad Pesaranghader, Norwati Mustapha, Nurfadhlina Mohd Sharef, "*Improving Multi-term Topics Focused Crawling by Introducing Term Frequencyy-Information Content (TF-IC) Measure*", September 2013.

[9] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles and M. Gori, "*Focused Crawling Using Context Graphs*", NEC Research Institute, Princeton, NJ 08540-6634 USA.

[10] Allan Heydon and Marc Najork. Mercator: A scalable, extensible Web crawler. World Wide Web, 1999, 2(4):219–229.

[11] A Standard for Robot Exclusion[EB/OL]. http://www.robotstxt.org/wc/norobots.html.

[12] L. Barbosa and J. Freire, "*An adaptive crawler for locating hidden-web entry points*", in Proceedings of the 16th International World Wide Web Conference, 2007.

[13] J. Callan and M. Connell, "*Query-based sampling of text databases*", ACM Transactions on Information Systems, vol. 19, no. 2, pp. 97–130, 2001.