Deep Web Technologies

[1] Sagar Kumar Choudhury, [2] Rajesh Kumar Padhi, [3] B.Giridhar, [4]Chandan Kumar Giri

[1, 2] B.Tech Student

[3, 4] Asst.prof in CSE dept

Abstract - With the proliferation of online repositories (e.g., databases or document corpora) hidden behind proprietary web interfaces, e.g., keyword-/form-based search and hierarchical/graph-based browsing interfaces, efficient ways of exploring contents in such hidden repositories are of increasing importance. There are two key challenges: one on the proper understanding of interfaces, and the other on the efficient exploration, e.g., crawling, sampling and analytical processing, of very large repositories. In this tutorial, we focus on the fundamental developments in the field, including web interface understanding, crawling, sampling, and data analytics over web repositories with various types of interfaces and containing structured or unstructured data. Our goal is to encourage audience to initiate their own research in these exciting areas.

***Keywords* – Web Technologies.**

1. Introduction

The tutorial shall begin with a series of real-world examples of deep web repositories hidden behind web interfaces (see Figure for a typical architecture). Specifically, a repository with structured data is Yahoo! Autos (*http://autos. yahoo.com*), while an unstructured one is the document corpus of Wikipedia.

We shall then use these examples to motivate the importance of efficient exploration over hidden web repositories. In particular, we shall show that many repositories only support a very restrictive set of search queries. To provide full (SQL) search support, one may need to *crawl* all elements from a repository and then execute the search locally. We shall also discuss the need of mining over hidden repositories. To support mining without incurring as many web accesses as crawling, one needs the ability to efficiently perform *sampling* and *analytical processing* over a hidden repository. We shall note that this tutorial focuses on deep web repositories with given URLs.

Resource discovery - i.e., how to find URLs of deep web repositories (e.g., for a given topic) - is an orthogonal problem.

Taxonomy of Web Interfaces: We shall describe four types of interfaces commonly present for web repositories: *keyword search* (e.g., Google), *form-like search* (e.g., Yahoo! Autos), *hierarchical browsing*

(e.g., Amazon's drop-down menu for product browsing), and *graph-based browsing* (e.g., Wikipedia).

Exploration Tasks: We shall describe three important tasks commonly desired for the deep web: crawling, sampling, and data analytics (e.g., the efficient processing of aggregate queries). We shall argue that while samples may also support aggregate (e.g., AVG) estimation, performing data analytics directly may be more efficient as its design can be made aligned with the specific aggregates to be estimated. On the other hand, sampling is more "versatile", as a collected sample may later support analytical tasks not yet known at the time of sampling.

2. Challenges

Our tutorial shall next discuss why the three tasks outlined above are difficult to accomplish over deep web repositories. We summarize two key challenges, one on understanding the interface - e.g., how to model web query interfaces and perform schema matching - and the other on the efficient exploration of data - e.g., how to determine which queries/browsing requests to issue, especially given the extremely restrictive input and output interfaces of a hidden web repository. We devote the rest of our tutorial to addressing the second (i.e., exploration) challenge. For the first one, we shall briefly review it and point audience to recent tutorials covering the topic.

3. Mining the Deep Web

In this first article in a series we introduce the deep web and tell you why, as a business or scientific professional you should care about mining its content. In later articles we will discuss in more depth some of the technical challenges to mining the deep web and how Deep Web Technologies and other companies are meeting those challenges.

The Internet is vast and growing - that's not news. Google does a great job of finding good information within it - that's not news either. What is news, and one of the dirty little secrets of Internet search engines, is that there's a huge collection of really useful content on the Internet that Google will never find - nor will any of its competitors, or any single search engine for that matter. We like to think that Google knows all, that if we click through enough of its search results we'll find whatever we need. This just isn't so. Beyond the 'surface web' of content that's continuously mined is the 'deep web'.

So, you're wondering, 'What is the deep web?' and 'Why haven't I ever heard of it?' In reality you've probably searched the deep web, maybe even surfed it, and never even realized it. The deep web is the collection of content that lives inside of databases and document repositories, not available to web crawlers, and typically accessed by filling out and submitting a search form. If you've even researched a medical condition at the National Library of Medicine's PubMed database www.ncbi.nlm.nih.gov/PubMed/ or checked the weather forecast at www.weather.com then you've been to the deep web. Three nice properties of deep web content are that it is usually of high quality, very specific in nature, and well managed. Consider the PubMed example. Documents cited in PubMed are authored by professional writers and published in professional journals.

They focus on very specific medical conditions. The National Library of Medicine spends money to manage and make their content available. Weather.com provides timely and specific reports of weather conditions for all of the United States and much of the rest of the world as well. Both collections share the three properties.

The deep web is everywhere, and it has much more content than the surface web. Online TV guides, price comparison web-sites, services to find out of print books, those driving direction sites, services that track the value of your stocks and report news about companies within your holdings - these are just a few examples of valuable services built around searching deep web content.

So, why doesn't Google find me this stuff? The answer is that Google isn't programmed to fill out search forms and click on the submit button. The problem is that there are no standards to guide software like the smarts behind Google in how to fill out arbitrary forms. In fact, computers don't 'fill out' and submit forms, they instead interact with the web server that's presenting the form, and send it the information that specifies the query plus other data the web server needs. Each web form is different and there are too many of them so Google can't know how to search them all. Plus, it currently takes a human to 'reverse engineer' a web form to determine what information a particular web server wants.

Standards are emerging to help with the content access problem and software will certainly get better at filling out unfamiliar forms but we have a long way to go before most of the deep web is accessing to the next generation of web crawlers. While filling out that web form is non-trivial it isn't the only barrier to accessing the deep web and it isn't even the hardest problem. Finding the best, or most relevant, content is harder. Within the deep web it means searching multiple sources, collating the results, removing duplicates and sorting the remaining results by some criteria that is meaningful to the person doing the searching. The problem of finding, aggregating, sorting and presenting relevant content is an involved one that we don't want to just gloss over so we will dedicate an entire article to discussing the issues.

As a professional you should care about What's in the deep web and about how to mine it effectively and efficiently. 'Why is that?' you ask. It's simple. In the worlds of business, science and other professional endeavors time is money. The slow and steady tortoise may win the race in fairy tales but it's going to get run over or left in the dust in today's competitive marketplace. The race to bring a new product to market, whether it be a new computer chip or a new drug, will be won by the company that can most quickly gather the most relevant information and intelligence and execute on it before its competitors do. A tool that can fill out forms on a number of web-sites with that high quality, specific and well managed content -- whether it be purchased, internal, or publicly available content -- then do the heavy duty processing to deliver the best of the best documents is worth its weight in gold. Such a tool will save you time and money and will make the best use of the content that you pay to acquire.

Imagine taking all of the intellectual property you possess or to which you have access and integrating its access into one simple to use form. Imagine further a system that knows what makes a certain document relevant to you as an individual. This system would be customized to scour your content plus all sorts of knowledge bases relevant to your needs and sift and sort information to present you with the very best of the deep web on demand. It would save you time. It would help you make money. This is the promise of deep web mining.

4. Challenges of the Deep Web Explorers

Web spiders these days, it seems, are a dime a dozen. Not to minimize the tremendous value that Google and other search engines provide, but the technology that gathers up or “spiders” web pages is pretty straightforward. Spidering the surface web, consisting mostly of static content that doesn’t change frequently, is mostly a matter of throwing lots of network bandwidth, compute power, storage and time at a huge number of web sites.

Merely throwing lots of resources at the deep web, the vast set of content that lives inside of databases and is typically accessed by filling out and submitting search forms, doesn't work well.

Different strategies and a new kind of "deep web explorer" are needed to mine the deep web. Surface web spiders work from a large list, or catalog, of known and discovered web sites. They load each web site's home page and note its links to other web pages. They then follow these new links and all subsequent links recursively. Successful web crawling relies on the fact that site owners want their content to be found and that most of a site's content can be accessed directly, or by following links from the home page. We can say that surface web content is organized by an association of links, or in HTML jargon, an association of <A HREF> tags. We should note that spidering is not without its hazards. Spiders have to be careful to not recrawl links that they've previously visited lest they get tangled up in their own webs!

If spidering the surface web is not an impressive achievement then what makes Google's technology so highly touted? In the case of Google and of other good search engines what's impressive is not the ability to harvest lots of web pages (although Google currently searches over four billion pages) but what the engine does with the content once it finds it and indexes it. Because the surface web has no structure to it good search technology has to make relevant content easy to find. In other words, a good search engine will create the illusion of structure, presenting related and hopefully relevant web pages to a user.

Google's claim to fame is its popularity-based ranking. It structures content by presenting first to the user web pages that are most referenced by other web pages. The deep web is a completely different beast. A web spider trying to harvest content from the deep web will quickly learn that there are none of those <A HREF> links to content and no association of links to follow.

It will realize that most deep web collections don't give away all of their content as readily as surface web collections do. It will quickly find itself faced with the need to speak a foreign language to extract documents from the collection. This need is definitely worth meeting since the quantity and quality of deep web content is so much greater than that of the surface web.
Deep web explorers approach content searching in one of two ways, they either harvest documents or they search collections on the fly. A deep web explorer may attempt to harvest content from a collection that doesn't support harvesting but, for reasons cited below, the effort will likely not be very fruitful. Dipsie and BrightPlanet are harvesters. They build large local repositories of remote content. Deep Web Technologies and Intelliseek search remote collections in real time.

Harvesting and real time search approaches each have their pluses and minuses. Harvesting is great if you have adequate infrastructure to make the content you've collected available to your users and if you have a sufficiently fat network pipe plus enough processing and storage resources to get, index and save the content you've obtained. Harvesting isn't practical if the search interface doesn't make it easy to retrieve lots of documents or if it's not easy to determine how to search a particular collection. If the collection doesn't support a harvesting protocol then harvesting will not retrieve all documents. Additionally, not having the network bandwidth and other resources makes harvesting impractical. And, if a collection is constantly adding documents then either the collection is going to somehow identify new content or you're going to waste lots of resource retrieving the documents already in your local repository just to get a few new documents.

OIA, the Open Archives Initiative, is an example of a harvesting protocol. OIA describes a client-server model useful for aggregating multiple collections into a single centralized collection. The server tells the client, among other things, what documents are new in its collection and the client updates its repository with them.

Deep Web Technologies' (DWT) Distributed Explorit application implements the other approach, the real-time search approach, which also has its pluses and minuses. A tremendous plus is that most deep web collections lend themselves to real-time searching even if they don't lend themselves to harvesting. This is because by not implementing a harvesting protocol the content owner doesn't have to do anything to its documents to allow them to be searched; it doesn't need to generate metadata or otherwise structure its content. An on-the-fly search client uses the simple HTTP protocol to fill out and submit a web-form that initiates a query against the content database. The client then processes (parses) the content returned and displays search results to the user. DWT's Distributed Explorit does multiple simultaneous real-time searches against different collections then aggregates the results and displays them to the user. The minuses of the harvesting approach become pluses in real-time searching. That entire infrastructure you needed to retrieve, store, refresh and index remote content and to then provide access to it disappears.

Minuses of real-time searching are the ongoing demands placed on the remote collection, the reliance on the availability of the remote content, the vulnerability of depending on search forms that change or break, and the inability to rank documents in a homogenous and effective way. (Search engines are notorious for ranking poorly or not at all and even collections that do rank documents in a relevant way can't deal with the fact that their well-ranked documents will likely be aggregated with documents from other poorly ranked documents.)

Now that we've tapped into the vast content of the deep web we quickly discover that we're drowning in content and not all of it is so relevant. What's a web explorer to do with so many documents? We'll explore this question next time.

5. Crawling

In this part of the tutorial, our focus is on discussing the crawling of a deep web repository *after* its interface is properly understood. We shall start with illustrating the motivations for crawling, and then discuss existing crawling techniques for repositories with search and browsing interfaces, respectively.

Search Interfaces: We shall identify two main prerequisites for efficient crawling over search interfaces: One is how to generate "legitimate" values for populating into input fields (e.g., query phrases as keywords). The other is how to input values such that each combination returns a large number of distinct elements.

Since solutions to both depend upon the repository's content, most existing techniques feature a bootstrapping process which starts with a small number of probing search queries, then uses the returned results to refine the selection of input keywords or attribute value combinations to quickly achieve high coverage. reduced to the traversal of vertices in a tree (for hierarchical browsing) or a graph (for graph-based browsing). The common technique is breadth-first search (aka *snowball* method). While the technique itself is relatively straightforward, we shall point out to the audience that the main challenge is the *comprehensiveness* of crawling, as the graph is not necessarily connected. We shall discuss techniques used by existing crawlers to address this issue. We shall conclude this part of the tutorial with discussions of the system-related issues (e.g., using a cluster of machines for crawling) that apply to both types of interfaces.

6. Sampling

In this portion, we discuss sampling techniques which aim to draw representative elements (e.g., documents, tuples) from an online repository while minimizing the number of web accesses.

We shall start the discussion with motivating applications for sampling, and then review existing techniques for keyword search, form-like/hierarchical browsing, and graph browsing interfaces, respectively
Keyword-Search Interface: We begin by showing that a key problem facing the "sampling" process in many existing techniques is that the returned elements have an unknown but often significant *skew*, i.e., certain elements are sampled with much higher probability than others. We shall then discuss a skew-correction technique through *rejection sampling*.

Form-like Search or Hierarchical Browsing Interface: Skew reduction remains a challenge here. In particular, the main source of skew is the *scoring function* used by the interface to determine which top-*k* elements to return. We shall discuss two ideas of skew removal: One is to avoid the influence of scoring function by finding queries that return <*k* elements. The other idea assigns a one-to-many mapping from queries to elements in the repository, such that even if a highly scored tuple is returned by more queries, it can only be sampled from one.

Graph Browsing Interface: We shall describe two types of existing techniques for sampling over a graph browsing interface: (1) the early work which uses BFS/snowball sampling to produce sample elements with an unknown skew; and (2) the random walk based techniques which has roots in the theory of finite Markov chains to produce known (and thus removable) skew over connected graphs.

7. Data Analytics

We shall now discuss analytics techniques for online repositories. We shall first argue that the key enabler for data analytics is the ability to approximately answer aggregate queries over an online repository, and then describe a few motivating examples of aggregate queries. After that, we shall discuss *bias* and *variance*, two complementary measures for the accuracy of aggregate estimations, and then review the existing techniques for the three types of interfaces, respectively.
Keyword-Search Interfaces: We shall focus on two types of data analytics techniques over keyword search interfaces. One is a two-step process which first calls upon the above-discussed sampling techniques to produce sample elements, and then use the sample to extract aggregate information for analytics. The other type of technique directly estimates aggregates without the middle step of sample generation. A key advantage here is that unlike in the sampling case where many retrieved elements may have to be rejected for skew removal, all retrieved elements may be used, albeit in a weighted fashion, for aggregate estimations.

Form-like Search or Hierarchical Browsing Interfaces: We shall first demonstrate that a direct estimation of aggregates over form-like or hierarchical browsing interfaces avoids the costly process of rejecting elements for eliminating sample skew.

Then, we shall explain why SUM and COUNT queries can be easily estimated without bias, while doing so for

AVG queries is extremely difficult if not impossible. After that, we focus on *variance-reduction* techniques for improving estimation accuracy. Before concluding this part, we shall briefly discuss a few recent works which have the exact opposite objective – i.e. to *prevent* aggregate queries from being estimated (accurately) through a form-like interface, in order to protect the privacy of aggregate information for repository owners.
Graph Browsing Interfaces: We shall start by arguing that data analytics over graph browsing interfaces is closely related to the problem of graph testing, as the latter assumes an access cost to learning whether an edge exists in the graph, resembling the web access cost for a graph browsing interface, and aims to learn certain (aggregate) information of the graph while minimizing the access cost. Nonetheless, we shall argue that the cost models of real-world interfaces are much more diverse than what have been studied in graph testing, leading to vastly different solutions and calling for further research on the cost models. We shall then discuss the existing work for aggregate estimation using random walks, random BFS, etc.

8. Proposed Methodology

The work is having three major parts first is web surfer search engine second is training of dataset last is extracting relevant data after mining from large data warehouse.

8.1 Web Surfer Search Engine

In this proposed method, search engine will provide only relevant link to the customers. At the time of crawling, the word gets divided and the sentence will be split into tokens and each token is assigned with consecutive numerical value.

Meanwhile, each word is matched up in web and then frequency check will decide whether that particular keyword exist in web or not

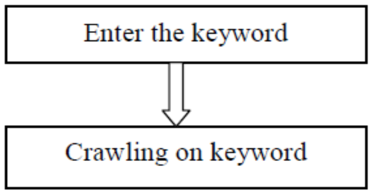

Figure 1

8.2 Training Data Set

In this stage the set of data will train and find out all the related links and then store it into a large data warehouse. The training data set is the major part of this stage. Training data set has a vital role in the identification of useful data.

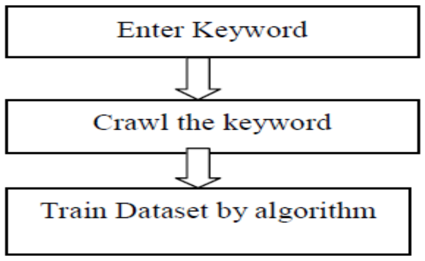

Figure 2

8.3 Extracting Relevant Data after Mining

This is the final and most important step of our work in which the relevant data is extracted from a huge set of data stored in data warehouse. The focus will be on mining the data from data warehouse and fetch the relevant links in to the search engine**.**

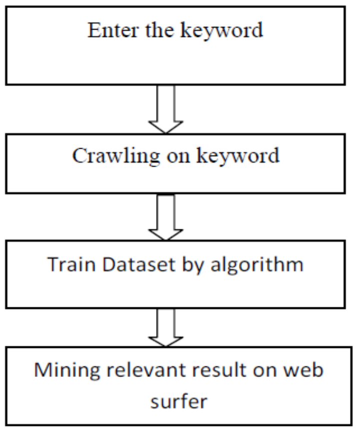
Enter the keyword
Crawling on keyword
Train Dataset by algorithm
Mining relevant result on web surfer

Figure 3

9. Proposed Outcome

At this stage the proposed outcome is keyword which user will enter the keyword in search engine that keyword goes to the web, fetch all the related links and stored in to the large data warehouse. Then from there the essential data will be extracted with the help of data mining techniques and user will get the useful data as output.

10. Conclusions

We shall summarize how the challenging problems of crawling, sampling and analytics over hidden web repositories require expertise in traditional query processing, IR, social networks, data mining as well as algorithms. We shall conclude by identifying open challenges.

References

[1] Claudia Elena Dinuca, Association and Sequence Mining in Web Usage, Economics and Applied Informatics, 2011.

[2] Hsinchun Chen, Xin Li, Michael Chau, Yi-Jen Ho, Chunju Tseng, Using Open Web APIs in Teaching Web Mining, ACM, 2009.

[3] Sachin Pardeshi, Ujwala Patil, Central web mining services–public and free access log files, WJST, 2012.

[4] B.Naveena Devi, O.Sreevani, Dynamic Modelling Approach for Web Usage Mining Using Open Web Resources, IJEST, 2010.

[5] Sanket Nagone, Bharat Kapse, Mayur Bhagwat, Ecommerce Application using Web API and Apriori Algorithm of Data Mining, IJCA, 2011.

[6] Claudia Elena Dinuca, The process of data pre-processing for Web Usage Data Mining through a complete example, Annals of the "Ovidius" 2011.