Evaluation and Development of Data Mining Tools for Online Social Networks

Dhiraj Murthy, Alexander Gross, Alex Takata, and Stephanie Bond

Social Network Innovation Lab, Bowdoin College, Brunswick, Maine
{dmurthy,agross,atakata,sbond2}@bowdoin.edu

Abstract. This chapter reviews existing data mining tools for scraping data from heterogenous online social networks. It introduces not only the complexities of scraping data from these sources (which include diverse data forms), but also presents currently available tools including their strengths and weaknesses. The chapter introduces our solution to effectively mining online social networks through the development of VoyeurServer, a tool we designed which builds upon the open-source Web-Harvest framework. We have shared details of how VoyeurServer was developed and how it works so that data mining developers can develop their own customized data mining solutions built upon the Web-Harvest framework. We conclude the chapter with future directions of our data mining project so that developers can incorporate relevant features into their data mining applications.

Keywords. Data Mining, Online Social Networks, Web Content Extraction, Web-Harvest 2.0

1 Introduction

The practice of data mining and web-content extraction is an important and growing field. Many disciplines are looking at ‘big data’ and ways to mine and analyze this data as the key to solving everything from technical problems to better understanding social interactions. For example, large sets of tweets mined from Twitter have been analyzed to detect natural disasters [1, 2], predict the stock market [3], and track the time of our daily rituals [4]. As our use of blogs, social networks, and social media continues to increase, so does our creation of more web-based hyperlinked data. The successful extraction of this web-based data is of considerable research and commercial value.

Data mining often goes beyond information retrieval, towards a meta-discovery of structures and entities hidden in seas of data. As our social interactions become increasingly mediated by Internet-based technologies, the potential to use web-based data for understanding social structures and interactions will continue to increase.

Online social networks are defined as ‘web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system’ [5]. Individuals interact within online social networks through portals such as Facebook which create social experiences for the user by creating a personalized environment and interaction space by combining knowledge of one users’ online activity and relationships with information about other networked individuals. It is through data mining algorithms that Twitter, for example, determines recommendations for users to follow or topics which may be of potential interest. One way to study social networks is by examining relationships between users and the attributes of these relationships. However, data on a blog, Facebook, or Twitter is not inherently translatable into network-based data. This is where data mining becomes useful. Social networks typically only provide individual portal access to one’s egocentric network. Put in the language of social network analysis (SNA), the visible network is constructed in relation to ego (the individual being studied) and relations of ego, known as ‘alters’, are seen (e.g. Facebook friends). However, in a restricted profile environment, the alters’ relationships are not revealed. In order to understand network structure (which is key to a systems perspective), the researcher must use methods like data mining in order to gather information about all users and interactions by iterating over the data. A variety of different types of tools have been developed to collect this type of information using the text based framework of the web. These tools were created for a wide array of purposes. The majority of these tools have been commercially released. Some of these tools can be used to construct profiles of individuals based on data from multiple sources. Given issues of privacy, ethical uses of these tools should be strictly employed.

Despite the existence of a variety of tools, the simplicity and robustness of them varies widely. There are many types of networks and online communities that could qualify as a subject of network-based research. Many of these virtual organizations and networks often share key elements and structures that are common across social network technologies. These could include users, groups, communications, and relationship networks between these entities. Also unlike the simple data that is subject of most data mining projects, network analysis is not merely focused on generating lists of entities and information. Social networks are more organic in their growth and place emphasis on relational attributes. SNA seeks to understand how individuals and groups within networks (termed ‘cliques’) are connected together.

The Social Network Innovation Lab (SNIL) is an interdisciplinary research lab dedicated to understanding online social networks, social media, and cyberinfrastructure for virtual organizations. Research at the SNIL often involves the need for tools that are able to extract social network-based data for analysis from varied online social communities. The SNIL currently has projects which require data mining of popular microblogging services, shared interest forums and traditional social networks. As part of our ongoing research, we have begun to investigate and develop our own custom data mining tools. As part of this project, we have researched existing tools, developed a conceptual framework for general data mining of online social networks save document, and built and tested prototype implementations of the toolkit while acquiring data for use in current ongoing projects.

In this chapter, we will consider a variety of common methodologies and technologies for generic data mining and web content extraction. We will highlight a number of features and functionalities we see as key to effective data mining for social network analysis. We will then review several current data mining software tools and their goodness of fit for data mining online social networks. The remainder of the chapter discusses our development of a data-mining framework for online social networks. Specifically, we introduce our initial development work in extending the Web-Harvest 2.0 framework to incorporate some of the important identified needs of data mining for online social networks. This is followed by a case study of some of our initial results and discoveries in the use of our pilot technology to acquire data from an actual online virtual community which is organized around social network technologies. The remaining sections summarize what we have learned through this process, and lays out a course for future development.

2 Web-Content Extraction Technologies

The nature of online social networking sites is such that the information and data that constitutes the network and its entities are by necessity distributed over a vast array of unique and dynamically generated page instances. Even considering only a basic set of common SNS features (user profiles, friend lists, discussion boards), it is easy to see how social network-mediated data exponentially grows. In order to study virtual communities as social networks, the researcher needs to transform this sea of distributed data into data formatted for network analysis software (most commonly UCINET for smaller networks and pajek for larger networks). In the absence of direct access to the database systems that drive social networks or a site-provided API, one must utilize other means to capture SNS data for research.
The field of online data extraction or web scraping has existed in one form or another since the advent of web based information. The majority of information on the Internet is circulated in the form of HTML content, which wraps data in a nested set of tags that specify how data needs to be visually rendered in the browser. This is suitable for making data easily read and understood from the screen or through printing, but not so useful when clean organized machine-readable datasets are desired. Most online data extraction tools take advantage of the fact that the HTML is itself a structured data interchange format, albeit one focused on the display of information. These tools leverage the HTML format to create language parsers, which can extract the simple content of the page in an organized way while discarding the irrelevant material. Generally, most online data extraction technologies can be classified into several categories with a few hybrids.

2.1 Formats, Conventions, Utilities, and Languages

Technologies in this class are low-level constructs that often derive from some sort of published standard grammar. This grammar may then be implemented in whole or in part by other higher-level technologies. They often simply define a way in which data can be ordered, searched, manipulated, or transformed. For instance, the XPath standard defines a format for finding and isolating pieces of information from a structured XML document. Similarly, regular expressions are a format for performing advanced searches and manipulation on unstructured strings of characters. XSLT is a language defined to assist in the transformation of one type of structured XML document into another (e.g. transforming an HTML document into a simpler RSS feed or vice-versa). Without well-defined standards for interacting with various types of data, extraction would be much more difficult, yet because of the low level nature of these structures is would be hard to use them in isolation to perform any kind of advanced extraction project without constructing a broader framework for their application to a set of data.

2.2 Libraries

Data extraction libraries often perform the job of wrapping one or more lower level data manipulation/extraction constructs into an organized framework within the context of a specific programming language. These libraries then manage the implementation of a given construct within a framework useful for further development within a given programming language. Development libraries leave the end goals completely open to the developer. Depending on the time, investment, and goals of the developer, development libraries can be used to create anything from simple one-off scripts to creating higher level applications with many advanced features.

2.3 Web-based APIs and CLUI

Web-based API and command line user interfaces often provide a kind of standardized abstraction layer to certain sets of web content. These typically wrap development libraries with their exact nature dependent on the hosting server and application. Furthermore, they will generally apply and be structured around the content available for one data source (e.g. a particular website, web-enabled technology, or application). Examples of such tools include Google’s OpenSocial API framework which is actually just an open standard for a set of API features that any developer could implement for their social networking site. Other examples include APIs provided by most large popular social networking sites like Twitter and Facebook.

2.4 Applications

The vast majority of data extraction solutions take the form of applications. Applications make use of a large set of extraction technologies and development libraries and wraps then in a interface designed around a set of desired functionality. Depending on that set of functionality and the level of expertise expected of the user by the developer, there can be a wide range of different types of data extraction applications, each potentially best suited toward certain sets of tasks and users. This spectrum extends the range from self-adapting, learning, fully GUI based extractors for non-technical users to applications for advance data extraction that may require some knowledge of programming or data extraction utilities. Many applications fall into this category. Some of the most common are Helium Scraper, Djuggler, Newprosoft, Deixto, and Web Harvest.

2.5 Enterprise Suites

This class of data extraction solutions is characterized by providing very high level, multi-featured, and advanced software solutions, often delivered in a suite of highly specialized applications. The implementations of these software packages are often fully private (as the code is often developed from the ground up based off the companies' own proprietary development libraries). Like many enterprise solutions these software products are often so powerful, advanced, and feature-rich that special training, and or ongoing technical support from the company itself are required to use these tools effectively. This support and training often come at significant additional cost beyond the original software license. Though potentially expensive, this support may allow the client to obtain custom solutions to their specific needs which would be developed for them by the company's developers in response to the client's specific needs. Pentaho and QL2 are two examples of enterprise solutions.

3 Considerations for Data Mining of Online Social Networks

3.1 Data extraction specification language.

One of the most important features for any professional level data-mining application for research is the implementation of a robust and dynamic data query specification language. This language should include the ability to define functions, execute loop, and conditional branching. Many basic data mining application just use a GUI to allow one to specify the desired extractions but this will always have it's limitations. Some tools used a command based query language like SQL to scrape data, but a better alternative is if the specification language is robust enough to emulate a programming language ipso facto. Conditional branching, loops, and functions, as well as the ability the define and access global and local variables are all needed qualities for successful data mining of complex structures like social networks.

3.2 Flexible I/O.

At its simplest level, data extraction centers around taking one kind of information as input and translating, manipulating or filtering it into another format more appropriate for ones' research objectives. In order to allow for the most possible types of automated data extraction and manipulations, data mining tools should be able to both read in and output to a large number of potential data formats. Ideally, the tool should be able to take input from and make output to all of its supported formats. Common formats that will provide the most amount of use within a data mining tools kit are various kinds of structured text files like HTML/XML, delimited text, or JSON. Additional important features include being able to read and write to different types of databases, APIs, or even the ability execute local system commands. The power and usability of the tools will increase the more ways it is able to take input and give output.

3.3 API Interfacing.

There are many different types of social networks from small shared-interest communities to large global networks. The administrators of these networks often recognize the importance for allowing different ways for people to access their data and they often provide third parties to develop applications which further develop or enhance participating in the network. These Application Programming Interfaces (APIs) often provide alternate methods for requesting information from a site. As opposed to simply requesting a page and extracting data from it, APIs allow developers to make a special kind of request to the API and return just the raw data one is looking for. Any serious data mining tool for the analysis of social networks should be created in such a way as to allow content extraction from both traditional web pages as well as APIs.

3.4 Job Scheduling.

There are several types of job scheduling features that would be most useful in data mining of online social networks.

Now. This option would immediately execute a job. This is the most basic type of scheduling operation.

Later. This option allows a user to schedule a job for a specific time. This could be useful to extract data from a site during low traffic hours, or for a situation where it is known that new information will be posted or made available at a specified time.

Chain. The chain option would allow one job to be scheduled to start once another had completed. This is very useful for when one data extraction task is dependent upon the completion of one or more other tasks. With this option the whole flow could be specified in advance and sent to the server application as a single project.

Recurring. Recurring jobs are quite valuable in data mining of online social networks. The vast majority of social network data presents difficulties in terms of mining due to the fact that social networks can be in continuous flux. For research purposes, it becomes necessary to capture a snapshot of data as it exists at a specified point in time. A feature to consider would be to include data extraction tasks that update the data at regular intervals.

3.5 Concurrency.

Most web-content extraction tools acquire data by essentially creating a virtual agent to make automated requests from a web host. This is the same way a web browser works. The web site is requested by the browser and the host send a file containing information (i.e. in HTML) needed to display the web page on the requester's browser. In data mining this data is simply grabbed and parsed in a variety of methods to obtain data. Most tools just use one agent to go to each page in sequence. By creating multiple page request agents, a data mining tool could make multiple concurrent requests to the same web host (for different information). This allows the user to take advantage of the scalability of the hosting server. For large jobs this feature would play a key role in speeding up the acquisition of data, and should be a key component of any data mining tools for online social networks.

3.6 Progress Management.

More often than not, the analysis of social networks requires large amounts of data. This is because these networks are most often user-agent based and each user will generate some amount of activity. Networks can be analyzed effectively by collecting the activity of individual users and their connections (egocentric networks). Though, it is often useful to collect data on all users (or at least from large subsets). This enables comparative analysis and discovery of connections between subnetworks (e.g. users who act as bridges from one group to another). In data mining tasks, data extraction is often limited by the speed that the hosting server allows clients to access data. Aside from concurrency, there is often little that can be done if the social network you are mining is slow or very large. If the network is both, it could take hours or perhaps days to complete a data extraction tasks. This is why we identify progress management as an important feature for the data mining of online social networks. Wherever possible, ideal tools should attempt to keep track of the progress of data extraction tasks as well as expected time to completion. This feature will be of great value to those who are charged with managing one or more data extraction projects by giving them the information they need in order to be prepared for when data will be ready for post-processing.

3.7 Playing nice.

When setting up large data extraction tasks, the operator of the software tool might be tempted to create large number of page request agents which generates a large amount of traffic on the hosting site. This not only is considered bad ‘netiquette’ [6], but has ethical and legal considerations. If one’s data mining project is part of academic research, the relevant Institutional Review Board (IRB) should be consulted as well to confirm ethical compliance with human subjects. A large volume of page requests with a web host could degrade the quality of the experience of other users of the site. Also it could result in the web host banning all requests from your IP address if the host believes your requests are malicious. Furthermore, many SNS have specific policies or terms of service in place that would dictate how much data can be requested per agent. It is in the best interest of the data extractor to ‘play nice’ and follow and conventions whether explicit, or implicit about not trying to request too much data from a host. If in doubt, contact the host whose data you intend to mine. Any data mining tool kit for online social networks should implement some kind of standard limiting, but also provide the ability to create custom guidelines depending on the known Terms of Service (TOS) of a web host or for when the user knows it is acceptable to request large volumes of data. The idea is to be able to set the tools to get data as fast as possible, but not “too fast” for the given host.

3.8 Client-Server Paradigm.

Extracting data from the web can require significant processing power as well a bandwidth. Many types of data extraction projects may be ongoing and most users would not want their computer constantly running potentially resource expensive scripts. This is why for data mining, the ideal solutions for tools is to use a client server paradigm (where each user simply submits their jobs to a server for handling). That way, the designated server can handle all the heavy processing and high data load while the clients' machine remains free for use. The server application just needs to notify the client when the data is final collected data is available. The client side application gives a lot of flexibility to the user requesting certain extraction jobs. They can use the client to log onto the main server and manage all there running jobs, no matter where they are physically located. The server should provide the client with options such as checking job progress, creating new jobs, aborting running jobs, changing scheduling, changing the extraction specification. Also, this provides the ability for multiple users with different data extraction needs to utilize one centralized server. If the server was designed to be appropriately powerful and scalable, then a powerful open research service could be provided.

4 Review of Existing Data-Mining Tools to Mine Online Social Networks

After the evaluation of several commonly available tools and technologies for online data extraction, we determined that Web-Harvest 2.0 was an the best fit for the needs of our project (which included mining data from two online life science communities of practice which used social networking technologies).

4.1 Common Data-Mining Tools

Among the tools considered were Helium Scraper, Newprosoft, Happy Harvester, Djuggler, Rapid Miner, Deixto, and Web-Harvest. Based on our evaluation it was determined that Helium Scraper, Newprosoft, Happy Harvester, and Djuggler, were all excellent GUI based scraping applications. However, these tools also shared the same limitations. All four tools were single operating system applications that only allow scraper configurations to be defined within the context of the application. They also have no ability to be controlled or configured from the command-line. Their source code is not open source and script could not be written against their various executables. When taken in consideration with our project goals (which required modifications for large scrapes to be conducted with minimal impact on the host), it became clear that these tools could not be leveraged to achieve our desired functionality. Rapid Miner is one of the leading open-source applications for data mining and analytics with solid data extraction capabilities. Rapid Miner was evaluated as a potential fit for our project's needs. It is open-source, cross-platform, uses XML-based configuration files which can be developed through the interface or written directly, and the code base can be scripted against both in application wrapper interfaces as well as from the command line. The issue is that Rapid Miner is such a powerful tool that it has a very steep learning curve and includes features which would not be needed for our project needs. Using Rapid Miner would prevent us from being able to develop a fast lightweight utility in a reasonable amount of time. DEiXTo is another web extraction technology that was potentially able to meet most of our project's needs. DEiXTo is a single platform GUI-based web extraction application built on top of an open source Perl-based scraper utility. DEiXTo also uses a XML based configuration language that potentially allows configurations to be defined outside of the GUI. The Perl module that forms the backbone of DEiXTo's extraction technology could also be scripted against on any operating system or code framework which supports the Perl scripting language. Though the DEiXTo file format (.wpf) is unduly complicated and not well documented. This means that most .wpf files must be developed within the GUI application, which is single platform and closed source. Also DEiXTo is limited in the way output can be written only to specific file formats and in specific ways. While the features and options available in DEiXTo would allow us to accomplish our project goals, it was determined that another tool, Web-Harvest 2.0, also had all the capabilities we needed for our project, but was easier to work with, more configurable, and more open in terms of input and output capabilities.

4.2 Web-Harvest

Web-Harvest 2.0 is a hybrid technology that consists of a GUI based application wrapped around a Java open-source development library. This library, in turn, implements several of the most common and powerful extraction utility formats such as XPath and regular expressions. The Web-Harvest 2.0 platform also defines syntax for defining custom data extraction workflows. This was ideal for several reasons. First, the graphical user interface allowed for quick start-up of development using the features of the GUI to easily debug, learn, and understand how to develop complex workflows in the Web-Harvest scraper configuration format. Defining workflows via this format is, in many ways, better than coding library solutions that require workflows to be defined in the context of that code base. This is because the configuration syntax is just a simple standard which can be written with any text editor. This frees the developer from the additional nuances of any specific high-level programming language. Furthermore once tested these workflows could be easily shared with others and passed to the development package, which could execute the scraper configurations through code. The fact that the core of Web-Harvest is an open-source data extraction engine allowed for our project to wrap this engine in our own lightweight code. We discarded the overhead of a GUI in favor of a lightweight command-line interface implemented with a client-server pattern. We were also freed from the limitations placed on the extraction engine by the GUI, by taking advantage of the Java programming language to develop our own features not present in the GUI or the engine itself. This included multiple simultaneous extractions as well as timed and repeated extractions. Furthermore all the configuration files we had previously developed could be simply passed to this engine and executed in the same manner as through the GUI. Web-Harvest 2.0 was a good fit because of its hybrid nature. Most pure application based scrapers are not extendable, and few define a configuration format. This causes the developer to have to work within the confines of what the application allows. Pure development package-based based extraction tools can have a steep learning curve and can be difficult to debug. Relying purely on data extraction utilities and standards like XPath, and regular expressions requires that an entire framework be built around them in order to execute complex dynamic extraction workflows. This can be time-consuming and resource intensive. Given the remit of our project, Web-Harvest served as an ideal solution in terms of features and the ability to separate between code and UI (which allowed us to quickly develop our own tools using the power of the Web-Harvest engine). Few if any other tools would have efficiently allowed us to work in this way.

5 Extension of Web-Harvest for Data Mining of Online Social Networks

After a review of existing data mining tools and a consideration of the desired features of data mining for online social networks, it was determined the best course of action was to develop extensions and a application wrapper for the open-source Web-Harvest 2.0 data extraction engine. Web-Harvest 2.0 already incorporates many of the basic functionalities identified as important in mining online social networks. Because the code is open source, we saw Web-Harvest as a good place to begin testing and developing a truly social network-centric data mining tool. Our plan centered on taking the extraction features that existed and wrap the code base within a multi-threaded client-server model. Once the base extraction modules were wrapped in this way, we could focus on added additional management feature to the wrapper like scheduling, process, and progress management.

5.1 Related Work

The data mining literature regarding either Web-Harvest or extensions to Web-Harvest is minimal. Web-Harvest has been used successfully as a basic scraper based in the literature. One such example was in a study by Nagel and Duval [7]. They used Web-Harvest in their study in order to collect large amounts of information from a website. For their study, they only needed a simple web scraper, and used Web-Harvest in its original form to mine publication data from Springer, an academic publisher. They used the software to collect data including titles, authors, affiliations, and postal addresses.

Katzdobler and Filho use Web-Harvest extensively [8]. They combined Web-Harvest with JENA a tool used to build semantic web applications as well as an ontology which described what type of information they wanted to extract. The ontology is then accessed by the jena api and Web-Harvest extracts the information from the site. However, manual creation of the configuration file and manual startup of Web-Harvest is needed.

TagCrawler is a program written using Web-Harvest and is is one of the few cases of Web-Harvest being directly extended [9]. The creators of TagCrawler desired a web crawling tool which would be able to retrieve information from tagging communities. TagCrawler is a web crawler that focuses on “retrieving data from tagging communities.” While the end goal of the project was not related to our project, their use of Web-Harvest as a base and building off of it has shown that this was a successfully deployed method.

5.2 Voyeur Server Project

This program is designed as an extension to the existing Web-Harvest 2.0 framework. It uses Web-Harvest’s existing functionality in terms of scraping and use of configuration files, but adds on several layers of additional features. In the development of this extension, we tried to take into consideration all the features we identified in Section 3 as being key considerations for the data mining of online social networks and attempted to push and adapt the Web Harvest 2.0 engine to better fit this model.

Project Overview. Web-Harvest is an excellent basic web scraper. The Web-Harvest framework has been able to satisfy many of our scraping needs. These include a robust query specification language with capabilities to import and export data to a number of important formats including MySQL database integration. Our most important needs include the ability of the scrape tool to be able to repeat upon finishing a scrape and to run on a specific date. Additionally, it was important to us that the tool be able to run concurrent scrapes. This is not only more efficient, but allows users to collect sets of scraped data for analysis rather than a single variable/page. We needed to be able to limit a scrape, check the status of all running scrapes, stop a running scrape, and update a scrape. Our tool adds all of these features by building on top of the Web-Harvest source code.

Project Structure and Development. Our program is organized around a client-server model with the server able to create any number of threads. This program is designed to run though the terminal with one window for the client and one window for the server. The server will first connect to a port on a server and once it has, the client can then connect and begin communicating. The server and client are able to communicate by using various readers and writers. The server uses a buffered reader to receive information from the client and an objectoutputstream to send information to the client. The client can receive information throug an object input stream and it send information with a printwriter.

When both the client and server are running, the user can enter input into the client. The client then sends the input to the server, which determines what the user has inputted. If the call is for a new scrape, the server will take the inputted information and create a new thread, which calls the web-harvest scrape code. This thread is then stored in an ArrayList in the server class. The thread also receives information regarding any date limit or repetitions and then performs all the necessary procedures. When the user inputs a command that will affect any existing scrape thread, the server will retrieve that specific thread from the ArrayList and pass in a command to the Thread. In general, the VoyeurClient receives inputs from the user and relays them to the VoyeurServer. The VoyeurServer then relays information and commands to the VSServerThread. If the command sent to the VSServerThread returns information, the Server receives the information and sends it back to the Client which prints it.

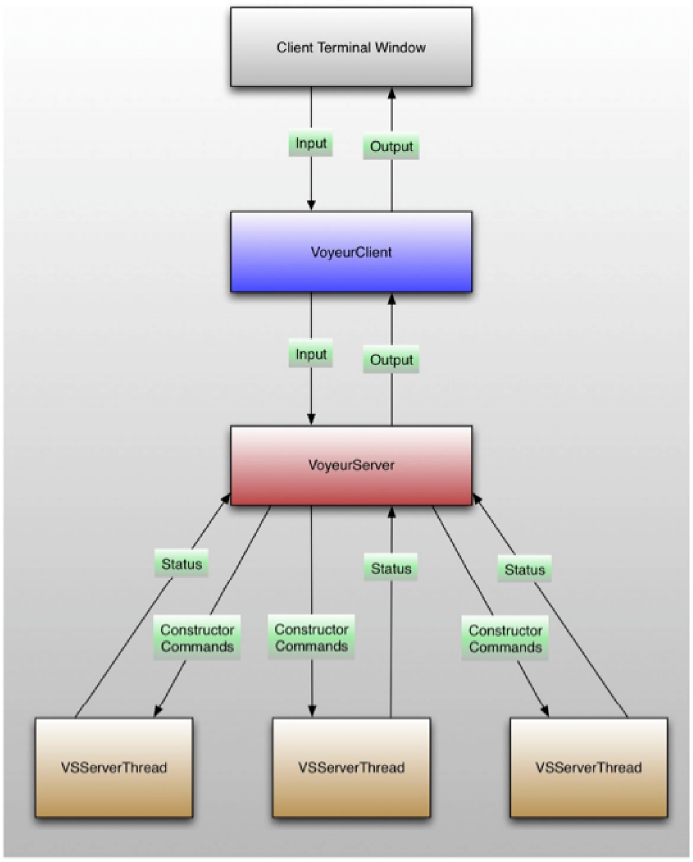

Fig. 1. Scematic of data flow in Voyeur Server

One of the first problems we attempted to tackle was enabling the program to run concurrent scrapes. We decided to base the program in the client server model, so we were able to use server threads. Each time we wanted to create a new scrape, the server would call a new thread and store the created thread in an ArrayList that enabled it to be accessed later. Each thread runs independently of all other threads. Since all of the actual scraping logic occurs in the thread, this enables the user to run multiple concurrent threads.

When beginning a new scrape of a file, the user has 3 pieces of information they can enter. First, they must enter the file extension of the scrape file they wish to use. Second, they can enter a date, which will be the start date of the scraper. Third, if they want the scrape to repeat, they can enter information regarding how frequently it would repeat. When this information is inputted, the server parses through the information provided and determines what scrape options have been inputted. With this information the server creates a new thread object and sends in the users choices regarding the start date and the repetition. When created, the thread stores the date and repeat frequency if present. Additionally, there are booleans for both the start date option and the repeat option. These are set according to what information is inputted. Once the thread has been created, the `run()` method of the thread begins. First, the important Web-Harvest classes are created.

Web-Harvest is organized such that all of the scraping functionality can be accessed through two of its classes. First, an instance of the ScraperConfiguration class can be created with the file extension as the parameter. Second, you can create an instance of the Scraper class, which takes an instance of the ScraperConfiguration as a parameter. When a scrape is to be called you call the execute function of the Scraper Class.

Within the `run()` method, there is a loop which contains all the code for calling scrapes. Which type of scrape is called depends entirely on which booleans are set by the constructor. This setup easily enabled the implementation of a start date limitation as well as the option to repeat. Within the infinite loop, there are a series of if else statements that check for the status of the date and repeating Boolean variables. The date is checked through use of the Java Calendar class. When the date Boolean is true the thread enters a loop which checks if the current time, (checked by a Calendar time feature), is equal to the time passed in by the constructor, when it is the scraper execute method is called and the program exits the loop.

VoyeurServer Features. As mentioned previously, the key parts of VoyeurServer included a command line user interface, job scheduling, process management, progress management, and database access.

Repeating Scrapes. There is a Boolean that designates whether or not the program should repeat. This section of the code uses the Calendar class to check whether or not the current time is equal to the time at which the scrape should repeat. So first the program creates an instance of the Calendar class that is altered by the time period the user wishes to repeat over. Next, it checks whether the current time is equal to the time at which it should repeat. Finally, when that time is reached the scraper executes. It will continue to repeat because it will continue to infinitely loop within the run() method.

Individual Scrapes. The final scrape option is a single stand-alone scrape. This option also has a Boolean associated with it that when true will cause a single stand-alone scrape. If while passing through the infinite loop the date and repeat variables are false and the single scrape variable is true, the program will run one single scrape.

Updating scrapes. The next problem we approached was how to organize the program such that the user would be able to update the initial parameters of a scrape after it has been created. When creating the scrape, it was important that the thread object be stored and be able to be accessed later. Once that was in place, the user could then call methods of the thread that would change the parameters of the scrape. If the user wanted to change either the date or repeat status of the scrape a method would be called in the thread that would switch the Booleans associated with the change. Since the thread is set up to always be looping, a change in the Booleans is all that was necessary to update the parameters of the scrape.

Status Check. The final change we made is the thread status. When called, the status returns the filename, the elapsed time of the scrape, the scrape finish time, the time of the next scrape, the current status, and the number of variables scraped.

Of these, the filename, scrape finish, and next scrape are simply stored variables. If a scrape is running, elapsed time is calculated by finding the difference is between the current time and the time the scrape began. If the scrape has finished, the difference between the finish time and the start time. The total elapsed time of the scrape is converted from milliseconds to days, hours, minutes, and seconds by taking a series of divisions and mods of the difference. The state of the scrape is determined by the three Boolean variables date, repeat, and single.

The most difficult task was regarding the number of variables scraped. This is more or less about the progress of the scraper. Each time the scraper scrapes a piece of information from a web site, there is a section of code in the scraper file that increments a variable in a database. When the Server calls the status method, the thread accesses that database and pulls the number. All of these variables are then concatenated into a string and returned to the Server. The server sends this string to the client which parses through this string and pulls out all the variables and prints them in an organized manner.

6 Experimental Results: A Case Study

We used this tool to gather information from an online life science community of practice. This virtual community consisted mainly of users and their communication across a wide array of social network-mediated interactions, including profiles, blogs, and forums. This section details how we were able to use our tool to acquire the information we needed for our research.

Our needs included capturing information on the users within this community and their communications with one another. Our eventual goal was to use this data to study patterns of trust choose one development of scientific collaboration online. The community we studied did not provide any API. Therefore, we had to rely on traditional web content extraction methodologies. We were able to use the Web-Harvest specification language to develop separate jobs to collect user data including profile information as well as collect posts. This is a fairly basic task. We wanted to limit the bandwidth consumption of our requests to not affect service of the site. Therefore, we limited concurrency. However, we were able to successfully collect user and post information simultaneously. At first we were collecting data into text files to be reviewed, evaluated, and codded manually. As our research developed, we were able to further update and modify our data collection job to collect information directly to a database. We had previously developed an application to assist in the coding and classification of the community's data based on this database. Being able to execute web content extraction to interface directly with some of our downstream research applications represents an extremely powerful and desirable workflow for network analysis. This is an area which we envision our future work to follow.

Although this research is preliminary and its remit has not been to test all the features we have identified as being important to data mining online social networks, our experience in developing the VoyeurServer tools has been positive and represents what we believe to be an important step towards the further development of this and/or other data mining tools specifically for online social networks. It is important to begin developing these domain specific solutions so that good open source options are available to researchers. In its absence the market will likely be left to be served primary by the existing companies whose tools focus much more on the domains of marketing and business knowledge. These types of solutions will never be ideal for pure research and could lead to a period where it becomes difficult for researchers to obtain this kind of information.

7 Future Work

Despite initial success in using Voyeur Server for mining data for network analysis, there are still many potential capabilities of Voyeur Server that have yet to fully tested. In Section 3 we outlined key features for data mining of online social networks. Currently, the Voyeur Server extension of Web-Harvest only implements these features in a basic way. Further testing and development would determine whether this extension has a future as a general research tool or whether it suggests that extending Web-Harvest 2.0 is perhaps less preferable than starting from scratch to develop a data mining toolkit for online social networks. If one is considering developing custom wrappers for Web-Harvest, we suggest considering these as possibilities of extended functionality:

- Coded modules for common APIs to ease the use of the extraction specification language for API related tasks.
- Coded modules for specific database tasks. VoyeurServer currently relies on raw SQL statements.

- Develop specification files for projects as opposed to per file scrapes. Incorporate timing and progress monitoring options.
- Smart or automated concurrency as opposed to having to design your individual jobs or project for concurrency.
- Develop ratelimiting features including self awareness of requests per second and bandwidth of incoming data. Ability also then to impose limits on itself to guarantee it does not exceed some bandwidth limits. It would also be useful to be able to set these values within a preference file or as a module. Limits for various sites could be defined and reusable.

Our continued work seeks to develop VoyeurServer in the following ways:

- High levels of concurrency.
- Investigate the feasibility of a broad-based public research server providing network-structured extraction as a service.
- Investigate high per-thread resources. Experience suggests that VoyeurServer is memory intensive. Our solution will need to make large numbers of jobs for various clients more efficient.
- Improved Interface for Web-Harvest backend (incorporating client server features)

We have shared these improvements so that data mining developers can be aware of issues we currently face and some possible solutions. This will enable designers and developers to learn from our development challenges.

8 Conclusion

This chapter has reviewed various data mining tools for scraping data from online social networks. It has highlighted not only the complexities of scraping data from these sources (which include diverse data forms), but also introduces currently available tools and the ways in which we have sought to overcome these limitations through extensions to existing software. After reviewing data scraping tools currently on the market, we developed a tool of our own, VoyeurServer, which builds upon the Web-Harvest framework. In this chapter, we outlined the challenges we faced and our solutions. We also included future directions of our data mining project. Concrete methods for developers to develop data mining solutions of online social networks using the Web-Harvest framework are provided.

9 References

[1] Doan, S., Vo, B.-K., and Collier, N., "An Analysis of Twitter Messages in the 2011 Tohoku Earthquake", 2011,

[2] Hughes, A.L., Palen, L., Sutton, J., Liu, S.B., and Vieweg, S., "“Site-Seeing” in Disaster: An Examination of on-Line Social Convergence", 5th International ISCRAM Conference, 2008

[3] Bollen, J., "Twitter Mood as a Stock Market Predictor", computer, 44(10), 2011, pp. 91 - 94.

[4] Golder, S.A., and Macy, M.W., "Diurnal and Seasonal Mood Vary with Work, Sleep, and Daylength across Diverse Cultures", Science, 333(6051), 2011, pp. 1878-1881.

[5] Boyd, D.M., and Ellison, N.B., "Social Network Sites: Definition, History, and Scholarship", Journal of Computer-Mediated Communication, 13(1), 2008, pp. 210-230.

[6] Morzy, M., "Internet Forums: What Knowledge Can Be Mined from Online Discussions": *Knowledge Discovery Practices and Emerging Applications of Data Mining: Trends and New Domains*, IGI Global, 2011. , pp. 315-336.

[7] Nagel, T., and Duval, E., "Muse : Visualizing the Origins and Connections of Institutions Based on Co-Authorship of Publications", *2nd International Workshop on Research 20 At the5th European Conference on Technology Enhanced Learning Sustaining TEL*, 2010, pp. 48-52.

[8] Katzdobler, F.-J., and Filho, H.P.B., "**Knowledge Extraction from Web**", in (Editor, 'ed.'^'eds.'): Book **Knowledge Extraction from Web**, Retrieved from: http://subversion.assembla.com/svn/iskm/FinalDocumentation/FinalReport.pdf, 2009

[9] Yin, R.M., Tagcrawler : A Web Crawler Focused on Data Extraction from Collaborative Tagging Communities, University of British Columbia, 2007.