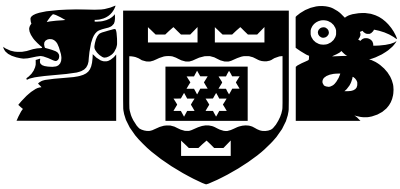
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Reputation Scraper - Social Media

Iain Walker

Supervisor: Kris Bubendorfer

Submitted in partial fulfilment of the requirements for
ENGR489 - Bachelor of Engineering.

Abstract

Online reputation has become increasingly important as more social and business interactions move online. This project is concerned with investigating how we can infer basic reputation metrics from non-traditional sources such as social media sites, including Facebook, Twitter, Google+, LinkedIn, and Slashdot. The method of gathering has been through web-scraping, since most social media sites do not (understandably) have an API to retrieve this information without logging in to the system. The project has looked at developing web-scrapers to retrieve publicly-available information on social media sites, and investigating how this can be used to infer basic reputation metrics.

Acknowledgements

I would like to thank my supervisor Kris Bubendorfer who was endlessly free with his experience and advice.

Contents

Figures

Chapter 1

Introduction

The aim of this project is to demonstrate the feasibility of a tool that gathers and provides data from social media websites, using a web-scraping approach. In this report I identify requirements and related work that will help me develop an approach to satisfy them...

FIXME

1.1 The Problem

In the digital age many social and business transactions are moving online. The size of Facebook and other Social Networking Sites (SNS) has grown exponentially with 1.05 billion monthly users on Facebook in December 2012 [15]. As such, defining who we should and should not treust becomes increasingly important, yet hard to resolve. Although SNS to some extent have inbuilt concepts of trust, using contact networks (Facebook, LinkedIn) and circles (Google+) for example, the concept of trustworthiness is more blurred. Tools to provide a snapshot, or hint of an individual's reputation and trustworthiness could be useful. Reputation is the global trust mechanism, whereas trust is one nodes impression of another in the network.

Reputation systems on trading websites have been extremely succesful, despite the potential for fraudulent activity []. While we might expect a self-aware adult to fairly easily spot equivilent 'fraudulent' activity or deceptive behaviour on social media, many examples have shown this to be untrue. Social crowdsourcing gone wrong has also been a high-profile issue, with incidents such as the Boston Bombings [].

Various strategies have been used to infer the personality or active traits of individuals online; these have been shown to be highly accurate. Applications on SNS have been used to give people an idea of their own personality. The results can then be compared against elements of the person's use of social media. This requires the express permission of the individual concerned to gain any meaningful data, when collecting information through a site's API. Which privacy concerns must be taken into account, publicly available information on an individual or company's social media account can be considered fair game. This information could be useful when judging trustworthiness. A new approach, that does not need permission to access information which is publicly-available, is needed.

1.2 Proposed Solution

A solution to the above issue is to use web-scraping strategies to anonymously retrieve reputation data. A better approach would be to retrieve information directly through an API, but few social networks provide sufficient APIs - and they all require express user permission to

work. This system could be used to give a general snapshot about an individual, by using these non-traditional data sources. There are many potential applications and users for this sort of application. Parents might wish to get a glimpse at their child's online friends, or a company might wish to automatically get some information about a potential employee for example.

1.3 Contributions

The major contribution of the project is to deliver a prototype that gathers online reputation data using web-scraping techniques.The prototype will be integrated with the GRAft reputation system [22], in order to link with a person's OpenID[32]. Users would be able to generate a reputation value for themselves, or for others they know. It aims to investigate alternative reputation sources, and look at how these can be applied in a practical sense.

Chapter 2

Background

This section covers the background research I performed for the project, and works related to my project. I also discuss how I was able to contribute to the research domain, with points of differences to these.

2.1 Web-Scraping Background

Web-scraping is a fairly well-understood problem, and there are existing architectures and frameworks to help facilitate web-scraping. Web-scraping or 'screen scraping' refers to the practice of downloading web pages directly through http requests, and parsing this unstructured data into something useful. Anything online that is accessible through a browser can be scraped. Hartley gives a light introduction to the concepts and strengths of web-scraping [8]. Web-scraping is largely used as an alternative to traditional API's on web pages, where these may not exist, or face a lack of support or development. Application's using web-API's which frequently change could also benefit from switching to a web-scraping alternative.

The primary limitation of web-scrapers is that major user interface changes will likely result in a broken scraper. In general site's structures do not change frequently enough for this to be a major issue. The typical argument however is that a site's structure can, without any warning, change at any time. This is an unavoidable issue when considering developing a scraper, and as such the regularity of user interface changes on a website should be considered carefully when deciding between using an API or a scraper. In this project, Facebook was ruled out as a scraping option due to its regular user interface updates. In general it is more sensible to inspect a site's API for applicability before jumping into constructing a web-scraper.

A counter to the limitations of web-scrapers is the frequency of API changes on modern social media sites. Past projects fetching data from Facebook have failed due to its too-often changed API []. In the course of this project, Twitter's API changed such that the possibility of fetching required data through this interface was rendered infeasible.

2.1.1 Legal and Privacy Issues

Companies such as Google and Facebook are widely known to use web-scraping on SNS and other sites for the purposes of web-indexing and advertising []. Despite this, other entities attempting to commercialise such aggregated data have been met with threats of hostile law action. Pete Warden discusses how his data-mining experiences on Facebook nearly resulted in legal action. In these cases, though, data-mining was intended to be used for commercial purposes.

By never signing into the SNS that I retrieve data from, I inherently do not agree to the user terms of service. I also followed general positive web-scraping practices where possible, in order to limit possible negative effects of my scrapers upon these sites. Examples include attemtping to adhere to a sites robots.txt, and place reasonable pauses between requests to reduce server burden.

2.2 Inferring Information from Social Media

Studies have shown how much information can be retrieved from SNS.

2.3 Personality from Social Media

Studies have shown how private details about personality traits can be inferred by behavioural analysis of individuals on social media. These studies have taken both a behavioural-based and textual sentiment-analysis approach to determine personality. Studies showed that real life personality traits can be predicted by looking at past behavioural patterns on Twitter, with high accuracy []. Using publicly-available information on Twitter pages, the authors were able to classify individuals into different Big-Five personality types []. This was verified against a 44-question personality test for 71 individuals, wih acucuracy of around 80%. Further research has been performed, looking at how Facebook behavioural patterns relate to personality type by asking questions on the MyPersonality Facebook app. Behavioural data was then gathered from the test subject's Facebook profiles, to check for correlations. Textual sentiment-analysis approaches have also been taken in order to determine personality type from social media profiles. Sentiment-based analysis of tweets has also been studied, analysing the use of emoticons on Twitter [].

While these studies showed how personality can be inferred from publicly-available data, they were made using applications which had to be given express permissions to access this data. In other words, although the conclusions drawn from these papers was that massive data-mining could be undertaken on these profiles as all the data is publicly available, they never used an approach that demonstrated the true feasiblity of this. Clearly as social media site API's are properly locked-down and require authorisation of the relavent users to retrieve data (see []), a web-scraping approach must be used. Such an approach allows applications to only require the same permissions as any human being accessing the same page, whilst also allowing the same data to be downloaded and aggregated.

A potential weakness in these studies is that personality on SNS may not translate well into actual personality. Although this may seem like a logical conclusion (e.g. Trolls, shallow internet relationships), studies have shown it to be false []. Quercia et. al. demonstrated how popular Facebook users in fact maintain meaningul relationships with their hundreds of friends, and not the superficial ones we may expect []. Again the Big Five personality model was used, and verified against popular Facebook users who used the MyPersonality app. Back et. al. demonstrated how Facebook profiles in fact are reflective of actual personality, and not a self-idealisation []. This study used manual inspection of profiles to make first impressions of individuals, which were compared against results from a Big Five personality test. Making similar predictions with computers would be an interesting and difficult challenge.

2.4 Quantifying Reputation on Social Media

Although the above studies investigate the concept of personality and how we can quantify this using social media, they did not expressly address the concept of reputation. Therefore we should discuss what reputation explicitly means on social media, and how sites currently help express user reputation.

Some sites explicitly reference user reputatation, a system commonly ustilised on social forums. StackOverflow is a question and answer site where users are able to post code-related programming problems. It has been identified that sites such as StackOverflow are largely dependant on the contributions of a small number of expert users, who provide a significant proportion of useful answers. As such, identitying users who have the potential to become strong contributors is of significant importance to the success of the website.

Action	Reputation Change
Answer is voted up	+10
Question is voted up	+5
Answer is accepted	+15 (+2 to acceptor)
Question is voted down	-2
Answer is voted down	-2 (-1 to voter)
Experienced Stack Exchange user	onetime + 100
Accepted answer to bounty	+bounty
Offer bounty to question	-bounty

Actions and Corresponding Reputation Change on StackOverflow [30]

Reputation in this context is entirely derivative of the actions of users. In the context of StackOverflow, a high user reputation score could be considered as a measure of expertise.

Sites such as Reddit use similar systems, where

2.5 Reputation Metrics Elsewhere

Reputation systems have had considerable success on trading applications such as eBay and TradeMe. Resnick et al looked at the success of eBay's reputation system and identified the three properties that a reputation system must enforce in order to add value to such a site [].

- Entities are long-lived, so that there is an expectation for future interacion
- Feedback about current actions is captured and visible to others.
- Feedback from the past guides current decisions: people must pay attention to reputations for them to be valuable.

This impacts my study as it guides the creation of reputation policies; reputation data I gather should conform to these concepts. Resnick's paper also identified weaknesses within current reputation systems online. There are three weaknesses present within current reputation systems which automated policies can help address.

- Fears of retaliation
- People not bothering to place feedback
- The assurance of honest feedback

2.6 Community Detection

2.7 Sentiment and Temporal Analysis

2.8 Related Systems

I will now discuss some systems which consider reputation data and apply it to access-control and service selection policies online. Firstly we will look at how reputation metrics can be used to assist with service selection in paradigms such as service-oriented-computing. Then the Generalised Recommendation Architecture and an OpenSocial access framework will be covered, in the context of access control solutions using reputation attributes as keys for access. Finally, the social-media based reputation system Klout will be discussed.

2.8.1 Service Oriented Computing

Service Oriented Computing (SOC) is a computing paradigm in which software applications are constructed based upon independant component services with standardised interfaces [39]. An emerging issue with this concept is how to select appropriate services based upon the application's needs. In the traditional model, service providers publish descriptions of their services in the service registry that is used by service clients to discover and select services. Najafi et al. identity four issues with this model [31].

- Service descriptions provided by the service provider may not be accurate or trusted
- Service descriptions generally do not capture specific performance requirements for different client's needs, in different contexts
- Less well-known services do not get an opportunity to show their features
- Service features vary with different measures, and are obtained under different situations, thus cannot be fairly compared.

Proposed solutions have suggested approaches such as promoting competition between webservices, using an independant entity to evaluate the different web services against given criteria.

The Generalised Recommendation Architecture (GRAft) is an example of such a framework that could store such reputation data.

2.8.2 Generalised Recommendation Architecture

The Generalised Recommendation Architecture (GRAft) [] is a proposed distributed architecture that supports the collection and storage of reputation information for entities, whether these be individual users or systems. It uses the OpenID [] infrastructure in order to maintain long-term identity for users. To ensure validity of reputatin data, information within GRAft is duplicated over the nodes (using a distributed hash-table) which comprise the network.

Entities within GRAft are uniquely identified using their OpenID. The system is able to maintain a history of the entities actions, including recommendation and transactional history. In this fashion, entity 'profiles' are established, with recommendation data being pulled from multiple sources.

Although GRAft is not a system currently used in industry, its draft paper discusses how it could be applied in practice. The domain of a scientific wiki is one such example.

This wiki consists of scientific articles, which can be read and edited by members. Using a two-tier access policy, read access to a page was is given by co-authorship. Users who had previously worked with the owner have a 1st-degree relationship; those who have worked with a person in the set of 1st-degree owners are 2nd degree owners, and so on. Once the user is granted access depending on this degree of co-authorship, editing rights are calculated by the users Hirsh-Index (H-Index) []. Another policy is then responsible for assigning editing permissions based on this value. This example demonstrates how access rights can be automatically granted for all users who fulfil requirements defined by the page owner. The group is never explicitly defined in terms of usrs; and as the author begins work with others, they can be automatically granted access to resources. This flexible policy comes closer to reflecting ad-hoc and informal scientific collaborative interactions, and is based on trust information that comes closer to real-life reputation.

```
$rb->logicalAnd(
 $rb['hindex']->greaterThanOrEqualTo(1),
 $rb->logicalAnd(
   $rb['degree']->greaterThanOrEqualTo(0),
   $rb['degree']->lessThanOrEqualTo(3)
  )
)
```

2.8.3 Open Social

Zhang et al. [] proposed an Open Social based group access control framework. Open Social is a framework for deploying cloud-based social applications. It was used in this context as it provides an API useful for retrieving social connection data. It also supports OAuth [], a protocol that allows users to grant third-part access to their protected resources.

The social trust scheme proposed in this paper consists of a multi-tenancy access control model, which can be applied to a scientific team-management scenario. Firstly the authors consider how trust in this context is a complex human-to-human relationship, developed through scientific collaboration. The authors argue that information about such relationships can be captured through data embedded on Online Social Networking (OSN) sites. This enables friend-of-a-friend trust to be computed, enabling transitive data as also seen in GRAft(which proposed using degree of co-authorship as a trust attribute).

2.8.4 Social Media Data Access Control

How do social media sites control access

How effective have social media sites been at controlling access, and how well do users feel that their information is safe?

2.8.5 Klout

Klout [] is a social media reputation system that generates an 'impact score' out of 100, based on how influential you are on various social media sites. In order to gather data from sites, you must create a Klout account and grant access to your various social media accounts. Klout then interacts with these sites APIs to fetch the relevant data. This site is useful to me as it both demonstrates how such information can be used to generate influence information, as well as how such a system can become popular. I extend this system by using the alternative approach of web-scraping. By doing so, users would be able to look at the reputation data of other users, as well as themselves.

2.8.6 Karma

2.8.7 Discussion

2.9 Discussion

Chapter 3

Methodology

3.1 Project Management Approach

The project was structured around a loose waterfall approach. At the start of the project, a long-term plan and major milestones were outlined. More detailed plans were added and target dates adjusted as the year progressed. Aspects of agile development practices were also used during the implementation and design stages. Throughout the project I had joint weekly meetings with Dr. Kris Bubendorfer, Ferry Hendrikx and Filip Dimitrievski. In these meetings, 30-45 minutes in length, we were able to outline progress achieved during the week, any issues encountered,and solidify project direction. Having meetings with Filip and Ferry present was beneficial, as there were aspects of the project which Filip and I were able to collaborate over. Ferry, co-designer of the GRAft system was also able to give useful technical feedback, with his experience in the development of web-scraping tools.

This approach of combining aspects of waterfall and agile was reasonably effective as a project management approach. Agile methods tend to work best in a team environment, assisting with the coordination of team members. However the method of small, focused sprints contributing to the larger project were excellent in maintaining focus and direction. The waterfall element in turn outlined the more concrete and long-term goals of the project, which was useful with monitoring and controlling and ensuring my work did not fall behind.

These target goals were met in the majority of cases. I did experience delays with finishing my implementation and evaluation, as new goals such as community detection were added late in the project. Reasons for this late discovery and delay were in part due to the exploritory nature of the project, as discussed in the design approach.

3.2 Design Approach

Requirements analysis and design were completed through a combination of research and prototyping. As the project was focused on an exploration of understanding reputation data on social media, a large portion of time was given to background research. This will be covered in depth in chapter 3.

Later in the design phase, I constructed a set of prototypes to evaluate the feasiblity and alternatives for a web-scraping solution. These covered preliminary scrapers for Twitter, Facebook, LinkedIn and Slashdot. The benefit of developing these was to both give me a better understanding of technologies involved with performing these functions, and to produce some meaningful effort early in the project, as these scrapers could be potentially useful later. There were some pitfalls in this approach however. It is noted that some effort

in prototyping can go to waste - and this was the case here. We eventually declared web-scraping Facebook largely infeasible, due to its user interface's constant state of flux. Also during prototyping, Twitter's API changed significantly, rendering much effort lost.

The weekly meetings with my supervisor allowed opportunities to obtain feedback on design choices, as well as suggestions where there was room for improvement.

3.3 Project Complexities

The complexity of the system stems largely from the aggregation of unstructured data from a variety of sources - a difficulty often encountered in web-crawling applications. The codebase itself is not overly complex. However, what contributed to the primary difficulty of prototype development was the development of code in the aggregation of data from disparate locations, and debugging often unclear and unexpected errors from various web requests.

Understanding the data gathered was a time-expensive challenge. The process of understanding and defining reputation data on social media was the task that occupied most of my time on the project.

The time constraint of 300 hours impacted the project at all stages. The implementation and evaluation components were particularly impacted - limitations had to be placed upon the scale of data collected from scrapers in order to compensate for time. In addition, the selected prototyping methodology was often expensive in terms of time, due to the necessity of revising code and collecting more relevant data, as I achived greater understanding of the problem domain.

Understanding reputation data, and writing policies to describe reputation effectively.

Debugging and collection of data, and asserting that this data is avalid. Ensuring that websites were not overloaded, resulting usually in scrapers being blocked. Recovery from detection, and how scrapers can respond. Construction of useful policies, and data analysis and aggregation.

Chapter 4

Requirements Analysis

To satisfy the goals of the project, the following issues need to be addressed. The requirements may be logically split into two divisions; web-scraping requirements and access policy requirements.

4.0.1 Functional Requirements

R1. Policies must consider data from reasonable time periods, to generate a shadow of the future.

R2. Aggregation of social media data into consistent and readable format

R3. Metric of reliability based on amounts of data collected.

4.0.2 Non-Functional Requirements

R4. Resistance to User-Interface Change

R5. Reasonable performance - expectation that policies and scrapers could be used as part of wider application.

R6. Accuracy of data collected - content should not be missing or incorrect

R7. Resistance to Blocking Detection - my scrapers should not be blocked.

R8. Extensibility and Social Media Portability for future use of framework

4.0.3 Discarded Requirements

R8 Aggregation of Social Media data, for storage in GRAft.

4.1 Policy Construction

The first and most significant requirement is to generate a set of policies to assist with generating a snapshot of the reputation information of an individual.

4.2 A reliability Metric Based on the Amount of Data Considered

From the beginning of the project I acknowledged that it may not be feasible to scrape the profile or data of an entire user

4.3 Social Media Platform Portability

The ability to build scrapers for new sites on top of existing architecture I develop.

4.4 Maintainability and Resistance to User-Interface Changes

Along with portability of my scrapers, they need to be somewhat resistant to change at the user interface level. An oft-repeated downfall of web-scraping is that changes to interfaces may occur at any time, without warning. This is in contrast to changing APIs, which generally give some significant warning and phasing out period of functions. Often changes to the layout of pages will break crawlers, resulting in a need for frequent re-builgs on sites that go regular user interface change. As such my system must be designed in a manner that is as un-reliant on layout-specific information as possible. In cases where change will unavoidably break my scrapers, they should be designed in a fashion which allows for easy identification and solving of issues.

4.5 Scraper Performance

This requirement refers to the speed and accuracy of my web-scrapers. The web-scraping tools developed must perform with sufficient speed to generate snapshots of an individual's reputation, in a reasonable amount of time. In order to create policies that are useful for future works, these scrapers must be able to gather and make conclusions about an individual in a matter of seconds or minutes. Data gathered must be accurate and represent what is actually displayed on a site.

4.6 Ability to Resist Blocking Detection and Recover from Failure

A challenge presented in web-scraping is the ability to resist detection as a robot by webservers. Webservers do not look well upon robots, and will attempt to block aggresive crawlers. As such my scrapers need to implement strategies to avoid detection, and when blocked or detected, take appropriate action. A balance has to be achieved between performance of scrapers and detection by webservers - normally a scraper attempting to retrieve masses of results will be detected and blocked extremely rapidly.

4.7 Privacy Protection

This requirement refers to the fact that data I gather should be anonymised to a reasonable extent, such that actual personal details should not be traceable.

Maintain reasonable privacy of data I collect. Discuss how the information I am gathering is publicly avaialble on social media anyway, and users should have reasonable expectation that their data will be accessed.

4.8 Discarded Requirements

Aggregate data for storage into GRAft - because of community effort required. Business model would need to be constructed, out of scope for the project.

Visualisation component of the project- discarded as was able to source external tools to visualise and represent data.

Chapter 5

Scraper Design and Implementation

In this section I discuss the design and implementation of the web scraping components of the project. I highlight important decisions made during the course of the project, as well as steps taken to implement the system based on requirements detailed in chapter four.

5.1 Architecture

Two potential architectures were considered when designing the web-scraper components.

- A database storage approach
- A text/XML storage approach

Having received feedback on the database approach, the more straightforward and suitable text/XML design was ultimately used.

5.1.1 Database Storage Architecture

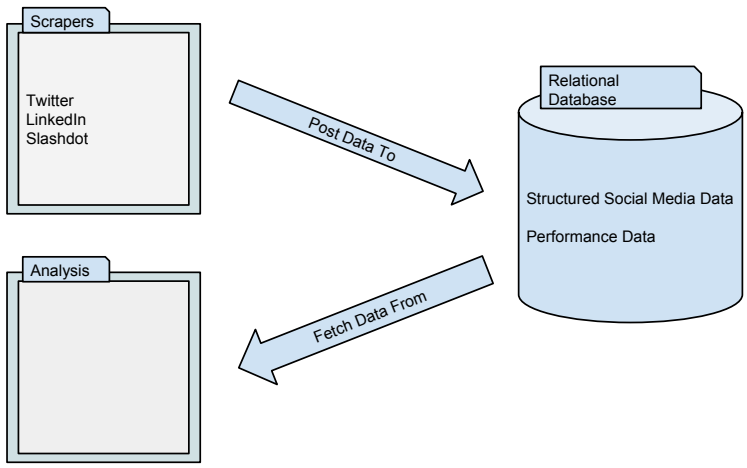


Figure 5.1: Proposed Architecture for Database Storage Solution

A possible approach to meeting the requirements of the project was to store fetched data in a relational database. In this design, the scrapers would gather data over an extended period of time, writing retrieved information into a database. Analysis components would then fetch data from the database as necessary, saving results in seperate tables within the same schema.

This approach provides long-term extensibility benefits, as per requirement R.4. All of the advantages that come with database storage would have been present in this solution - fetching of specific profiles for example would have been much more practical through a database query.

However this design was deemed to be over-engineered for the purposes of the project. Given that data retrieved is in HTML, JSON or XML format there is a large impedance mismatch between retrieved data and relational database tuple format. This is compounded when retrieving data from the store for analysis - many third party analysis tools require input data in a variety of text forms. The analysis components were also largely contingent on the fetching of a whole profile, rather than specific elements, lessening the need for querying against specific elements in my data storage structure.

5.1.2 XML Storage Architecture

The alternative and ultimately selected architecture involves the scraper and analysis components interacting with a shared XML storage structure. Again in this structure, scrapers run over time whilst saving data to a common XML-based file system. Analysis components could then annotate and enrich the original dataset.

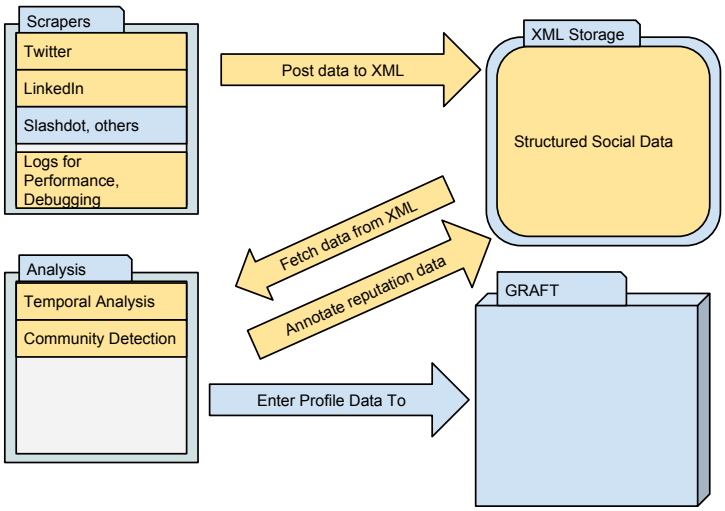


Figure 5.2: Architecture for XML Storage Solution. Implemented Components are Highlighted Yellow

This design largely simplifys the scripting and scraper construction. Little setup was required during prototyping to construct schemas against which to save data. An informal XML approach also allowed for straightforward manipulation of data structures, and additions to what was being stored. Performance of storing to files is also known to be much faster and less server-taxing than the use of a database.

The primary disadvantage to the XML approach is difficulties associated with selecting individual slices of data in XML. Technologies such as xQuery exist for querying against XML, but were deemed unecessary for the project. This is due to entire profiles being used for input into analysis components, rather than selected components only.

5.2 Technology

The web-scraper components were written entirely in Ruby. The development of screen scrapers is largely independant upon language choice; however Ruby was selected due to its large number of libraries suitable for scraping, and straightforward scripting nature. Ruby also has a significant OpenSource following in comparison to many more conventional languages like Java. This was considered necessary early in the project, when looking at a model for future maintenance of the scrapers, especially for the discarded requirement of aggregating data for GRAft.

Alternatives to Ruby were considered; Java, which does not have the same open source following as Ruby, as well as a larger degree of lower-level network coding for web-scraping software; and PHP, which was rejected due to time constraints limiting personal capabilities to learn the language to a competent level during the project. PHP would have given the advantage of being more consistent with past works, however [**?**], as well as being the same language as my policies are described in.

The frameworks upon which my scrapers were built included:

- Nokogiri: a widely used HTML and XML parsing framework, allowing for straightforward interpretation of raw HTML documents.
- Mechanize: an extension of the Nokogiri framework, that simulates browser actions on web-pages.
- Rest-Client: Ruby's most popular HTTP client.

Alternative frameworks were considered, and their advantages and reasons for not being selected in the final model will now be covered.

5.2.1 Extend scrAPI

scrAPI is a Rubyforge project, allowing for fast implementation of web scrapers. The core benefit of scrAPI is allowing data to be fetched from HTML pages using CSS selectors. In addition it hides processes such as the actual fetching of pages. It is a much higher-level framework than Nokogiri and Mechanize.

Extending scrAPI was ultimately discarded, due to its heavy reliance on CSS files remaining constant. Any change to stylesheet files would likely have broken the scrapers. Arguably these stylehseets are less likely to change than layout manipulations (for example consider xpath on HTML as an alternative); however on sites such as Twitter and Facebook large design teams frequently make changes to these files. Given that a key requirement of the project was to make scrapers resistant to user-interface change, this resulted in scrAPI being deemed unfit for purpose.

5.2.2 Extend A Browser-Automation Model

Browser automation options were considered as an alternative architecture. Frameworks such as Watir allow users to simulate user interaction with web pages, by driving an actual browser instance.

A brower automated-scraper would likely have had little problem with site detection or blocking, due to the use of a real browser - providing genuine user agents, downloading css selectors and such. Despite this, performance would have been significantly poorer using this option. As browser automation scripts are generally highly dependant on site layout and interface naming conventions, this would have violated the requirement of resisting user interface change (R5).

5.2.3 Use the API

Although the project title from the outset was 'Reputation Scraper', we investigated the use of site's APIs before settling on the scraping option. A basic application interacting with the Twitter API was constructed early in the project. Twitter's API version 1.0 was actually highly suitable for the needs of the project. However the changes in REST API v 1.1 rendered this infeasible. The earlier API allowed fetching of up to 800 statuses from any public profile, along with more public data (see appendix for more information). Version 1.1 however implemented more requirements for authentication, and limited functions such as downloading tweets to the individual's account only. Facebook's developer API is even more restricted. On Facebook, permissions must be obtained from the appropriate users before fetching data from their profile. LinkedIn's policy is similar.

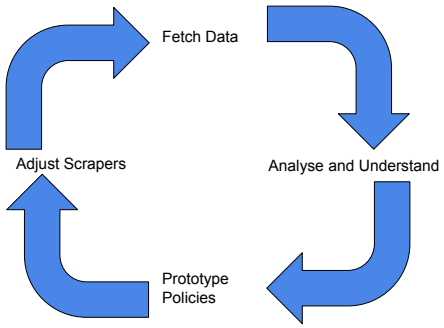


Figure 5.3: Scraper Development Feedback Loop

5.3 Systems Design and Structure

Developing the project solution consisted generally of a cyclic, iterative prototyping approach as detailed in figure 5.3. This generally consisted of four phases; Implementing and adjusting scrapers, fetching data, analysing the data, and prototyping policy concepts as a result of this analysis. The first scraper developed was for Facebook.

5.4 Facebook Crawler

A Facebook scraper was prototyped early in the project, and allowed for experimentation with many of the technologies that were used later. Firstly, strategies for authenticating against Facebook had to be developed. While public profiles do exist, these are primarily for company pages or politicians, which is not what the project aimed to soley investigate. Authentication was implemented using a dummy Facebook account, through the Mechanize Ruby framework. Mechanize allows for form data on pages to be populated and submitted through either HTTP or HTTPS. Once authentication was achieved, an auth token was

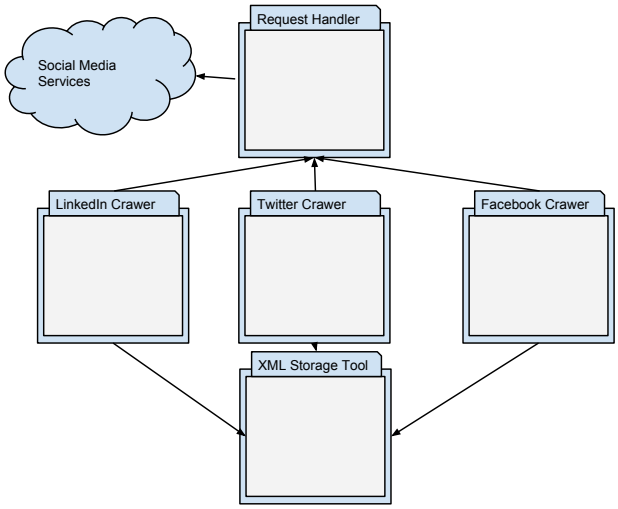


Figure 5.4: Pipes-and-Filters Class Structure

passed as an attribute of futher requests. New auth tokens are obtained after an adjustable time period, or on a new scraper run.

Having obtained the necessary authentication tokens, profile pages could be fetched. The planned approach was to crawl through friend networks via friend lists, and scrape profiles in this manner. Downloaded profiles would then be parsed using Nokogiri, and relevant data extracted via xpath expressions. Posts, likes, friends are examples of information identified as useful to collect. However as mentioned in chapter 3, this scraper was eventually discarded as infeasible. Scraping large amounts of content reliably through Facebook was infeasible within the project's scope due to its highly dynamic content, and frequently changing interface. Maintaining and updating a scraper to fetch accurate data from Facebook during the project would have been too time-consuming and distracting, and as a result was removed from the project scope. However the technologies experimented with whilst developing this prototype were useful later when developing the more robust Twitter scraper.

5.5 Twitter Crawler

The Twitter Crawler users a Google Twitter search function to search for profiles to fetch at the top level. An input text file of random names delimited by lines is accepted by the scraper. These names are looped through in order, with the scraper fetching the profiles of these top ten results returned through Google. Bias is introduced through this approach, since only the top ten search results are analysed.

Due to this bias, alternative approaches were considered. A potentially better scientific approach to randomising the selection of profiles to scrape would have been to crawl through an already-generated set of random profile names. However searches did not reveal any such databases to exist, excluding 'celebrity-exclusive' ones which would have been significantly worse than my current approach. I also considered crawling via embedded links on Twitter, starting from any profile and following retweet links through to new profiles. However this introduces new complexities - how will the algorithm detect infinite loops (potentially huge loops), or detect repeat profiles? As the project was ultimately about reputation and not scraping algorithms, the simple name-searching approach was selected.

Once the ten profile names are selected (for the current input name in the text file), the application will perform the scraping of each of these profiles in order. The definition of a Twitter profile changed throughout the project, as more information was chosen to be analysed. The final schema of stored data for the standard Twitter scraper is contained in appendix 1.1. Essentially all tweets are captured where possible, along with associated retweets, favourites, and the profile names of users who retweeted this item. Profile metadata is also stored, such as number of followers, number following, and total number of tweets.

Technical difficulties exist with fetching such profiles and metadata, which we will now discuss. Firstly, a Twitter page does not immediately display all tweets for a user for obvious latency reasons. To view more than the initial tweets, users must scroll through an 'infinite scrolling' window which requests more data through AJAX calls. To simulate this interaction, I used Google Chrome's 'network capture' feature to inspect requests to the page during the scrolling function. The URL is of the following form:

```
/timeline/with_replies?include_available_features=1&include_entities=1&max_id=12345
```

The JSON returned is then parsed to retrieve relevant information. As a Twitter profile may contain retweets from other profiles, checks have to also be taken at this stage to ensure the content is originally generated by the user of interest. This is as simple as checking the user name who posted the content originally. In order to increase the performance of extra tweet fetching, various tweet window sizes were experimented with. This was ultimately inneffective at maximising speeds, reasons for which are covered now.

5.5.1 Parallel Tweet Fetching

Although tweet content can be fetched from the base profile page, useful data such as retweet and favourite count is initially hidden. To view tweet detail, a further request must be sent, to mimick the user interaction of clicking on the tweet. Fortunately this process can be parallelised for each tweet within a tweet window. In the final twitter scraper a thread pool of 15 threads is created, one for each potential tweet in the window. Each thread then fetches and parses each tweet individually. To ensure parallel performance is not lost, any tweet-fetching process that fails here will simply be thrown away. Parallelisation of tweet fetching resulted in huge performance gains, which will be discussed further in chapter 6. Parallel fetching was not considered earlier in the project as I expected the resulting simulataneous requests to end in more frequent detection by sites.

5.5.2 Detection Avoidance and Recovery

Operating on the University network likely resulted in less chance for blocking detection by Twitter, due to the large IP range allocated to Victoria University. Regardless, steps were taken to limit detection rates. Multiple strategies for avoiding detection were considered, and several employed simultaneously. The Request-Handler package handles the implementation of these strategies.

The first and most straightforward strategy is to include random pause times between requests. The goal of this approach is to reduce potential server burden, and lessen the possibility of an aggresive scraper being permanently blocked due to its flooding the site. However this approach imposes a very low ceiling for performance and tweet fetch rates. Resources such as [] recommend a ten to fifteen second delay between requests, but such a wait was infeasible for the needs of the project. As such, pauses between requests were only implemented in certain special cases; failed requests and at the end of a profile.

The second strategy employed is to send requests that spoof a browser user-agent, in order to appear as a legitimate browser instance at the server end. Before scraping a new tweet window, a new user-agent string is randomly selected from a list.

A third strategy that was considered but not implemented was the downloading of CSS and other markup, in order to behave as browser-like as possible. This was discarded because of questions of its feasiblity, as well as uncertainty as to the level of potential benefits from constructing such a solution.

I rarely encountered rate limiting at University, and was never blocked outright on the University machines. Early in the project, when prototyping a Facebook scraper I was briefly banned from Facebook at home. 503 errors reveal detection on Twitter. 500 errors were encountered throughout the project also, but these were due to coding errors.

5.5.3 Dealing with Authentication

The original scraper did not require authentication, since the majority of Twitter profiles are viewable to individuals who do not have a Twitter account, or are not signed in. However to fetch the names and profiles of users retweeting posted content, an authentication feature had to be added. Fortunately the authentication strategy employed by the Facebook prototype was able to be re-used for this component.

To fetch retweet-name data, more features had to be added. All other data fetched was publicly viewable, whereas this data is only available to users who have authenticated with Twitter.

5.5.4 Dealing with Poorly Formed Markup

A concern earlier in the project was the scraping of content containing poorly formed HTML or JSON markup. Thankfully the sensitivity of HTML parsers to poor markup is largely library-dependant. This was not well-documented in the gems (a gem is the library equivalent in Ruby) I found, so experimentation with various options had to suffice.

5.6 LinkedIn Scraper

The LinkedIn Scraper was developed towards the close of the project, and as a result data analysis of what was collected using this tool was not carried out.

5.7 Code instrumentation

The code was instrumented using a Ruby logger system written for the application. Given complexities and difficulties debugging errors on web-scraping applications, logging had to be performed to a very fine level of granularity. Any action changing system state such as fetching a page triggers the logging mechanism. The logger would then take note of the timestamp and write to the appropriate file the nature of the action. For example, if the scraper sent an HTTP request to retrieve a given URL, it would record the timestamp and URL requested.

The logger would write to the appropriate log file based on the nature of the supplied action. Because these scrapers were running over long periods of time, using traditional IDE debugging tools was not effective at detecting errors. As a result, I used multiple debugging files with different purposes in an attempt to catch these errors. The debug.log captured

all interaction information at a basic level. Error.log captures error information that is non-fatal to scraping an individual's profile, e.g. on Twitter. Commonly these errors were due to application logic flaws, such as performing operations on null entities. As a result the error.log assisted greatly in identifying these edge cases. Finally the failure.log was used to record fatal exceptions that would prevent me from scraping a profile. Occurences such as 404, 500 or 503 responses from servers are examples that would result in a record being written to the failure log. The failure log would write system state at the time of failure to a high level of detail, sometimes even writing the entire HTML document before the failure to disk. This again assisted with debugging, when reviewing how a particular run had gone.

To limit the performance overhead of writing to these logging files, a buffered approach was taken in order to achieve the least impact.

Chapter 6

Data Discussion and Policy Construction

In this section I discuss the construction of my reputation policies for use on Twitter. In particular the development of analysis tools for achieving greater understanding of the Twitter dataset created will be discussed, and discussion of hypotheses generated from this data and how we experimented with these.

6.1 Social Media Selection

Until now, the reasoning behind my social media selection has been largely neglected. In order to gather useful data, I looked at the various social media sites in the context of gathering reputation data, and seeing what could be fetched.

In order to scrape useful information, I needed to look at the various social media sites in the context of gathering reputation data, and seeing what could be selected.

- Facebook: Publicly available information on Facebook includes data such as a user's Likes, Posts to Walls, and Friend networks.
- LinkedIn: The primary business social network on which professionals can display information pertaining to their career and skills. Reputation data is fairly concrete - endorsements, skills, groups, and recommendations.
- Twitter: The social networking site allowing posts of up to 140 characters. Information available is number of followers, number of profiles the person is following, number of posts, and post data itself. Post data includes retweet and favourite count, the content itself, as well as the people who retweeted the tweet itself.
- Google+: Google+ is similar to Facebook in style, and holds a large user base, but with much lower daily use than Facebook.
- Slashdot: Slashdot is a social media and news network targeted at a 'geek' audience. Slashdot is one of few social media sites that uses a positive/negative rating system on posts. This inbuilt reputation system could be interesting to look at further, when porting data between social media platforms. It is also unique from the standpoint of identifying 'trolls' - for which the website is widely known.

This basic feature analysis was the foundation for data exploration on the social media sites scraped.

6.2 Collection Method

Data discussed was collected entirely from the Twitter Scraper I implemented. This was collected during implementation, as per the feedback loop in figure 5.3. As such, profiles scraped earlier contain less detail than data collected later. For example, names of retweeters was not a feature collected on earlier profiles. In total, 1767 separate profiles were scraped, totalling 1,745,161 tweets.

6.3 Retweet Analysis

I now present the results of retweet analysis on Twitter, and put forward example policies that may be constructed from such analysis. The first question we asked was in the use of retweet and favourite functionality on Twitter. When 'retweeting' content on Twitter, the post will appear on the users own wall, as a tweet. A 'favourite' instead is added to a user's list of favourite tweets, which may be viewed seperately by other members of Twitter. To demonstrate policy examples later, and to simplify data collection, we put forward that tweets similar in popularity will have equivalent amounts of retweets and favourites.

H1. As a measure of impact, the number of retweets and favourites for an item on Twitter are equivalent.

A simple approach to demonstrate that the use of retweets and favourites is equivalent in terms of impact or response is to check the correlation between these figures on a set of tweets.

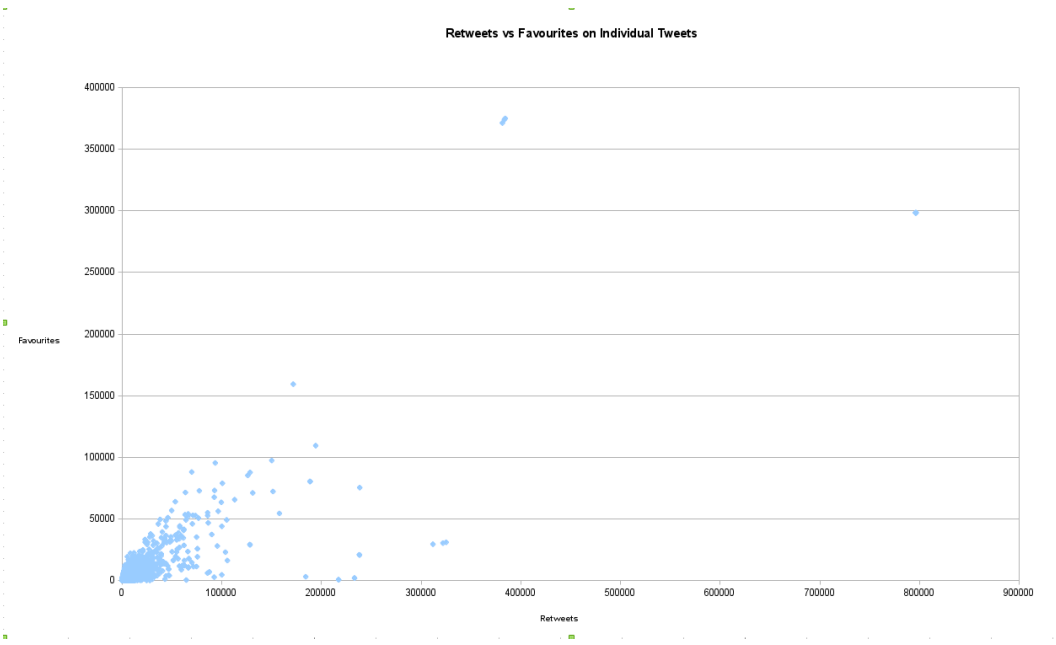


Figure 6.1: Correlation of Retweet Count to Favourite Count

6.3.1 An Impact Factor

We created an 'impact factor' measurement based upon a Twitter user's retweet count, and tweet activity.

First present the use of retweets vs favourites on Twitter, and show that they are similar

Next we look at impact factor of a person's retweets, pulled from the Hirsch-index formula.

Policies as an outcome of this, written using Ruler?

H1. Retweet and Favourite use is equivalent on Twitter

H2. Embedded social networks exist within Twitter, and can be detected. We predict that

Our definition of equivalent defines that

6.4 Temporal Clustering

6.5 Community Detection

6.5.1 MapEquation

This section discusses the design and implementation of my reputation policies associated with social media.

There are lots of different areas to discuss. For example, temporal analysis, sentiment analysis, retweet/favourite analysis, community detection discussion.

Constant emmitter - correlation between their impact factor and temporal bucketing will be higher than one-hit wonders. Detection for one-hit wonders.

6.6 Policy Construction

6.7 Temporal

6.7.1 MapEquation

Two phases to each facet of implementation; scraper, data understanding, and policy implementation and evaluation. Discuss each of these in seperate sections.

Define multiple hypotheses to add strength to the report

Brief description of LinkedIn Scraper. Ultimately decided that twitter had significant data interest to be the only focus of the project.

Discuss how the project was a study on what could be inferred from the data that we can access. Therefore some dead ends, etc. E.g. LinkedIn, Sentiment140 API for tweet analysis.

Temporal analysis

Evaluation tool development

Phases of implementation for each tool. Design, implement scraper, implement data analysis components, implement evaluation components and policies.

This section outlines my evaluation strategies, tests performed, and presents and discusses the results of evaluation conducted on the developed web scrapers. It then presents an evaluation of the policies and hypotheses formulated during analysis of data gathered.

6.8 Policy Evaluation

This section outlines my evaluation strategies, tests performed, and presents and discusses the results of evaluation conduceted on the developed system.

In order to evaluate my solution I need to refer back to the requirements gathered at the beginning of the project.

- These requirements will be moved to the Requirements Analysis section... is just useful to have them stored here for easy referrals.

- Develop a set of Policies that can be used to ...
- Develop a metric of reliability of information gathered, based on privacy settings.
- Potential to Develop an Interface with Filip Dimitrievski's project. Re-using my web-scraping libraries could be useful for his project. (Probably not going to happen)
- Aggregate data for storage into GRAft

Non-Functional Requirements:

- Privacy Protection
- Maintainability and Resistance to User-Interface Changes
- Performance of Scrapers
- Ability to Resist Blocking Detection, and Recover from Failure

6.9 Policies

6.9.1 Correlation of Impact Factor vs Bucketed Impacts

One strategy for inferring reputation information that I looked at was through temporal bucketing of impact factor, into days, months, and years. The figures below show the relationship between people's mean monthly calculated impact factor (i.e. by calculating a person's impact score for each month, and then averaging these values over the total number of months), and the individual's overall impact score. I infer that there is a weak relationship between average monthly impact and overall impact (Pearson's correlation coefficient of 0.273(3 s.f.)), and a stronger relationship between average yearly impact and overall impact (r=0.689(3 s.f.))

What I believe this data shows is that the impact factor is favourable to individuals who are consistent on Twitter over time.

6.9.2 Results of Monthly Bucketing

By applying the impact factor formula to individuals on a monthly basis, we are able to generate an impression of how regularly active a person is. The data also reveals how influential the person has been per month. This assists when comparing individuals who are consistently strongly influential (e.g. Barack Obama, companies such as instagram), with those who are popular for a limited period only. I looked at comparing different bucket sizes for this temporal aggregation. Days, months, and years were used as buckets, with monthly aggregation clearly showing the strongest and more interesting trends.

We can see that the daily and monthly impact trends follow roughly the same curve, which should be expected. The difference is due to the strictness of the impact factor formula in classifying someone as 'influential'. It is very difficult to get a value much above 1 on a daily basis, unless you are very frequently both making use of the social media, and having people respond to this activity.

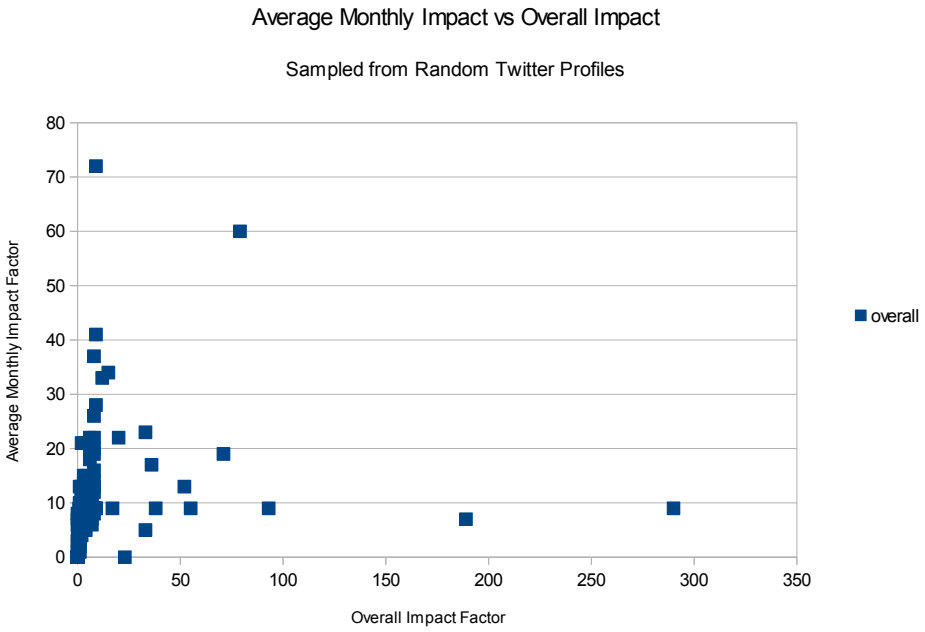


Figure 6.2: Monthly Impact against Overall Impact

6.10 Scraper

6.10.1 Functional Requirements

Aggregate data for storage into Graft - given that I am currently only storing data from Twitter this requirement has not yet been met.

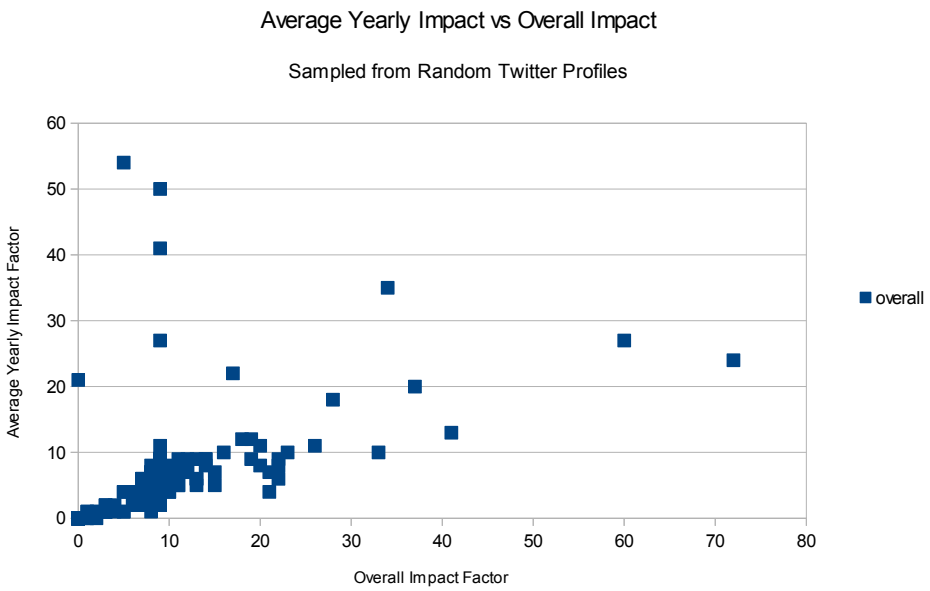


Figure 6.3: Yearly Impact against Overall Impact

Develop a metric of reliability of information gathered, based on privacy settings. Currently on Twitter there are two degrees of privacy - Open and Closed! Closed profiles means I cannot get any meaningful information, whereas on Open profiles I can get anything I want! Most profiles are open so this is not a concern. Profiles of celebrities etc are always set to open.

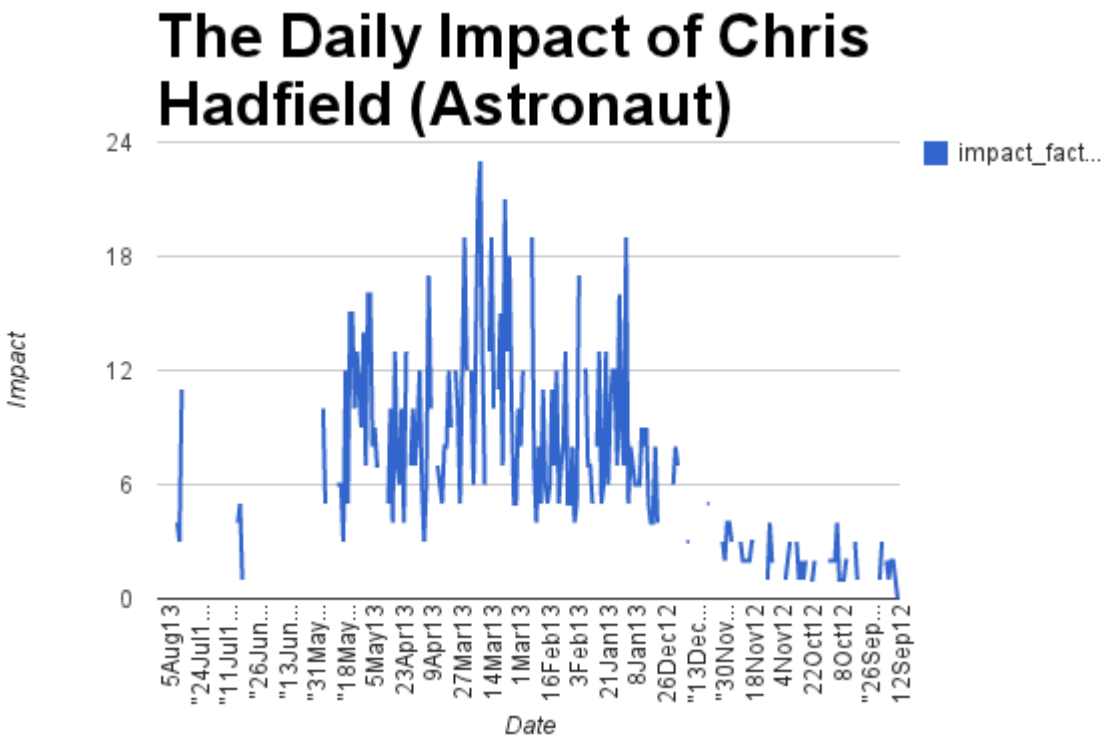


Figure 6.4: The Daily Impact of Chris Hadfield - Famous Astronaut

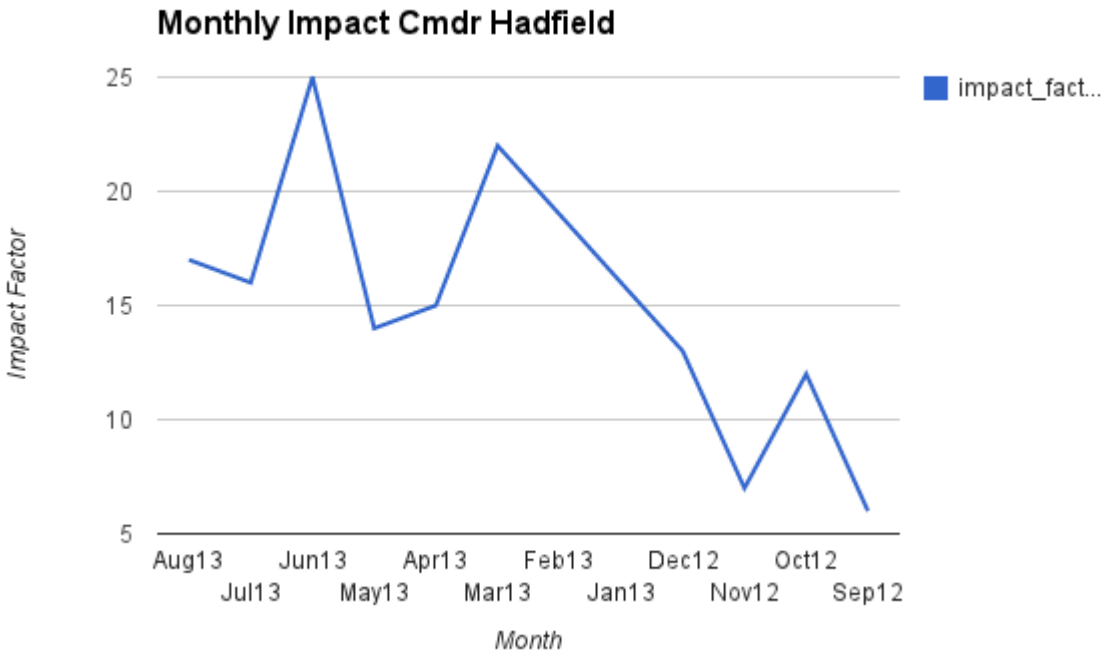


Figure 6.5: The Monthly Impact of Chris Hadfield - Famous Astronaut

Developing a set of Policies - still in the production line, still experimenting with different ways of using the data.

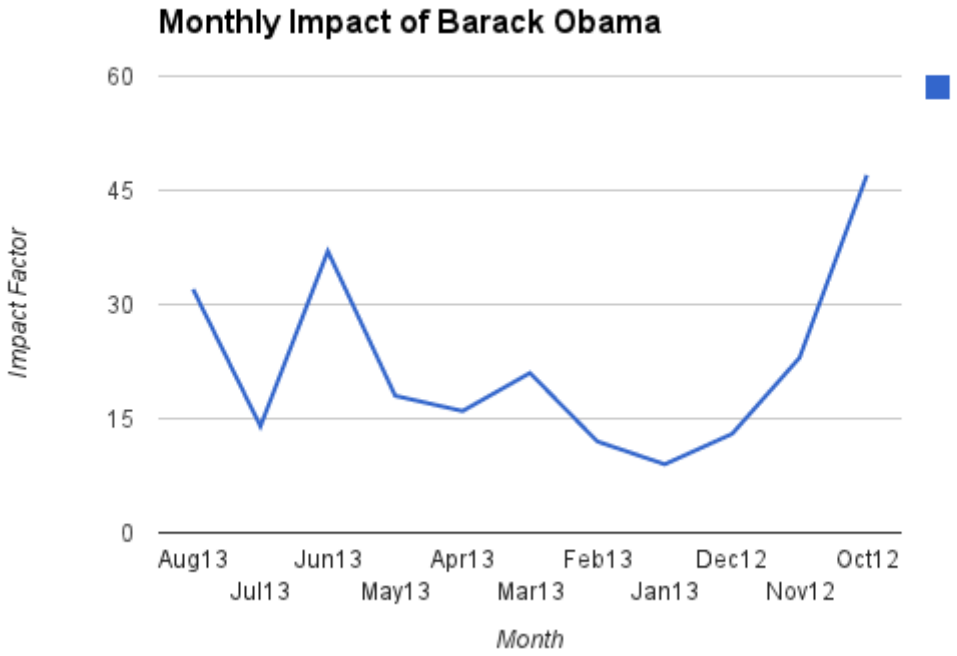


Figure 6.6: The Monthly Impact of Barack Obama - President of USA (As if you didn't know that)

Chapter 7

Evaluation

This section outlines my evaluation strategies, tests performed, and presents and discusses the results of evaluation conducted on the developed web scrapers. It then presents an evaluation of the policies and hypotheses formulated during analysis of data gathered.

7.1 Scraper Accuracy

http://www.beevolve.com/twitter-statistics/

7.2 Performance Evaluation

Given the iterative approach taken to develop the scrapers, I evaluate their performance with respect to incremental builds, as well as final performance. As the definition of a 'profile' differs between sites, different performance metrics are used for each site.

Performance testing of scrapers was carried out throughout the project, through performance logging mechanisms designed to measure the time taken to fetch an entire profile. To reduce bias caused by varying network speeds at University, the software was run over extended periods of time.

7.2.1 Twitter Performance Evaluation

Twitter profiles consist of differing numbers of tweets, retweets and so on. Thus I evaluate the performance of my Twitter scraper with respect to average time taken to fetch and parse a tweet. The major limiting factor for scraping useful information from Twitter was that each tweet had to be fetched with a seperate HTTP request.

Should mention that some profiles viewed (not necessarily scraped) can contain hundreds of thousands of tweets.

7.3 Web Scraper Evaluation

Performance - what performance goals did I have. How did they improve over time. Justify use of the incremental prototype-based strategy, through performance gains throughout the project.

Resistance to Detection - what performance goals did I have.

Problems

Accuracy - issues and why. Aims that I had - don't think I achieved them. In general entire profiles were not scraped.

7.4 Scraper

7.4.1 Functional Requirements

Aggregate data for storage into Graft - given that I am currently only storing data from Twitter this requirement has not yet been met.

Develop a metric of reliability of information gathered, based on privacy settings. Currently on Twitter there are two degrees of privacy - Open and Closed! Closed profiles means I cannot get any meaningful information, whereas on Open profiles I can get anything I want! Most profiles are open so this is not a concern. Profiles of celebrities etc are always set to open.

Developing a set of Policies - still in the production line, still experimenting with different ways of using the data.

7.4.2 Non-Functional Requirements

Maintainability and Resistance of Scrapers to User Interface Change

This is difficult to evaluate. Experiment: Change the layout of twitter pages, see how the scrapers react to these changes. How many changes do I need to make? There will undoubtedlu be single points of faiure, which I should mention. Important point: used as little xpath expressions as possible, as these inherently result in less flexbile scrapers. Any structural changes, or even a tag changing result in an xpath expression having to change. Still resistant to stylesheet changes, though. Single points of failure.

I identify areas which will result in the scraper breaking.

Twitter Scraper Performance

I evaluate the performance of my twitter scraper with respect to average time taken to fetch and parse a tweet. The major limiting factor for scraping twitter was that each tweet had to be fetched with a seperate http request. Figure ¡x¿ shows the average time taken to fetch and parse each tweet, with respect to my incremental build stages. The success of continuous and incremental improvement in performance helps justify my decision to use an incremental approach. Tweets were gathered over several days, and continuously throughout different times of the day on the Victoria network to ensure that a representative range of times were gathered for each stage.

I evaluate the performance of my twitter scraper with respect to average time taken to fetch and parse a tweet. The major limiting factor for scraping twitter was that each tweet had to be scraped with a seperate http request. Figure ¡x¿ shows the average time taken to fetch and parse each tweet, with respect to incremental builds. This also helps justify my incremental development strategy - we can see that succesive builds gradually increase performance. Tweets were gathered over several days, and at different times of the day on the Victoria network to ensure that a representative range of network performance speeds were used to gather data.

The significant feature of improved performance was through parellization of tweet-fetches.

Absolute limitations - Every tweet has to be fetched in an individual http request. This produces upper limits as to how fast the scraper can go, and means that the majority of performance speed is reliant on the speed of the network. (data - show how on some days tweets are fetched slightly faster than on other days. Times of day.)

Ability to Resist Blocking Detection

The primary measure of my scraper libraries avoiding blocking detection is with regards to their failure or incomplete-scrape rates. Although in early, less stable builds I had parsing errors, my later work only failed when detected by twitter and blocked. As such detection rates in later builds can reliably be calculated by analysing failing or incomplete twitter profiles.

Privacy Protection

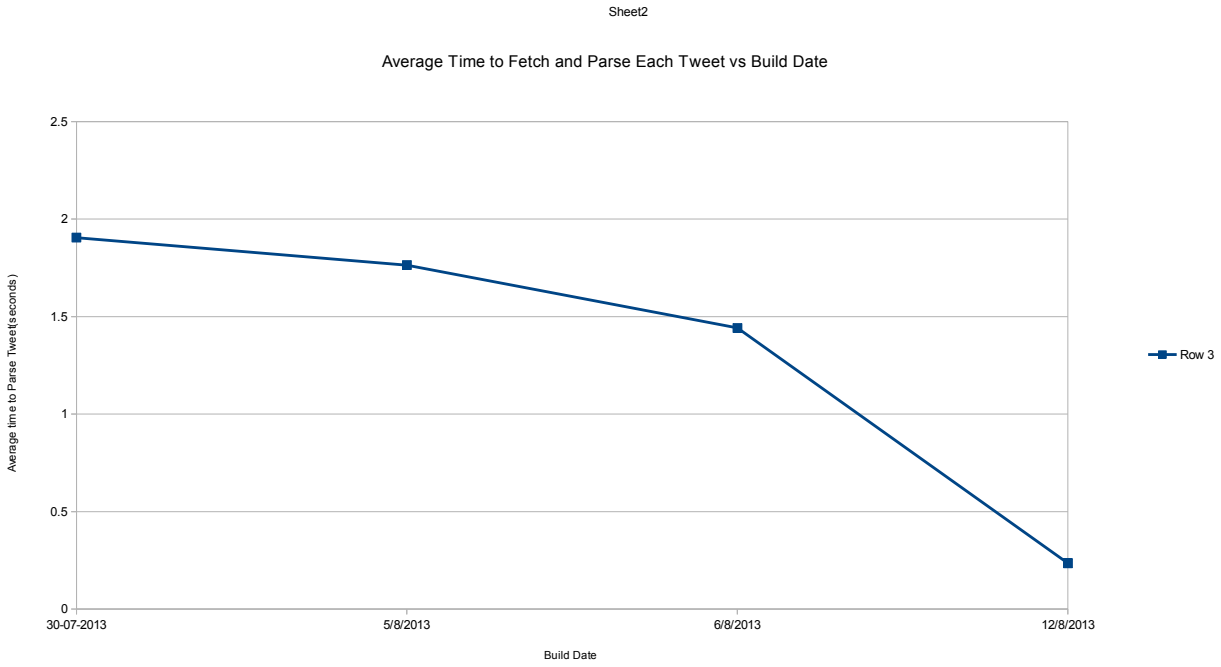
Privacy protection - this is still pretty poor, saved as structured xml, data not anonymised (makes my life easier at present)... Are there things I could be doing here potentially to increase privacy for individuals that I am scraping? Document locking, or store as RDB? Since I plan on storing this info in Graft, this could be anonymised at this point. Individual profiles and what-not.

7.5 Possible Improvements

Greater parallelizaion - over the ECS Grid for example, rather than froma single machine. Parallelize components other than tweet fething - entire profile fetching could be split up. Over engineered?

7.6 Scraping Conclusions

Conclusions about what pages are suitable to scrape


Sheet2
Average Time to Fetch and Parse Each Tweet vs Build Date
2.5
2
1.5
1
0.5
0
Average time to Parse Tweet(seconds)
30-07-2013
5/8/2013
6/8/2013
12/8/2013
Build Date
Row 3

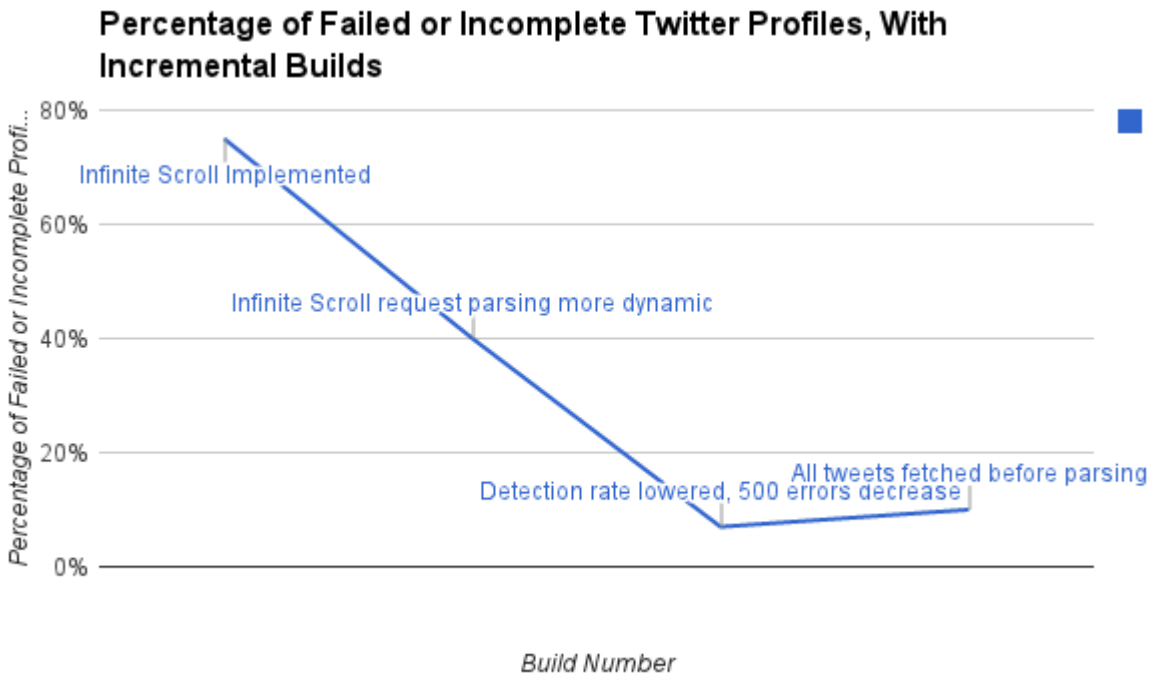


Figure 7.2: The Percentage of Incomplete or Failed Profile Scrapes, Ordered by Builds

Chapter 8

Conclusion and Future Work

8.1 Future Work

Compare detected embedded social networks against small-world networks and check for correlation.

Bibliography

[1] ADALI, S., ESCRIVA, R., GOLDBERG, M. K., HAYVANOVYCH, M., MAGDON-ISMAIL, M., SZYMANSKI, B. K., WALLACE, W. A., AND WILLIAMS, G. Measuring behavioral trust in social networks. In *Intelligence and Security Informatics (ISI), 2010 IEEE International Conference on* (2010), IEEE, pp. 150–152.

[2] ADALI, S., AND GOLBECK, J. Predicting personality with social behavior. In *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on* (2012), IEEE, pp. 302–309.

[3] ADALI, S., SISENDA, F., AND MAGDON-ISMAIL, M. Actions speak as loud as words: Predicting relationships from social behavior data. In *Proceedings of the 21st international conference on World Wide Web* (2012), ACM, pp. 689–698.

[4] ARRINGTON, M. Facebook users revolt, facebook replies. `http://techcrunch.com/2006/09/06/facebook-users-revolt-facebook-replies/`, 2006.

[5] BACHRACH, Y., KOSINSKI, M., GRAEPEL, T., KOHLI, P., AND STILLWELL, D. Personality and patterns of facebook usage. In *Proceedings of the 3rd Annual ACM Web Science Conference* (2012), ACM, pp. 24–32.

[6] BACK, M. D., STOPFER, J. M., VAZIRE, S., GADDIS, S., SCHMUKLE, S. C., EGLOFF, B., AND GOSLING, S. D. Facebook profiles reflect actual personality, not self-idealization. *Psychological Science 21*, 3 (2010), 372–374.

[7] BRABHAM, D. C. Crowdsourcing as a model for problem solving an introduction and cases. *Convergence: the international journal of research into new media technologies 14*, 1 (2008), 75–90.

[8] BRODY, H. I don't need no stinking api: Web-scraping for fun and profit. `http://blog.hartleybrody.com/web-scraping/`, 2013.

[9] BRZOZOWSKI, M. J., HOGG, T., AND SZABO, G. Friends and foes: ideological social networking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2008), ACM, pp. 817–820.

[10] DE RAAD, B. *The Big Five Personality Factors: The psycholexical approach to personality.* Hogrefe & Huber Publishers, 2000.

[11] DEBATIN, B., LOVEJOY, J. P., HORN, A.-K., AND HUGHES, B. N. Facebook and online privacy: Attitudes, behaviors, and unintended consequences. *Journal of Computer-Mediated Communication 15*, 1 (2009), 83–108.

[12] DOAN, A., RAMAKRISHNAN, R., AND HALEVY, A. Y. Crowdsourcing systems on the world-wide web. *Communications of the ACM 54*, 4 (2011), 86–96.

[13] DUBOIS, T., GOLBECK, J., AND SRINIVASAN, A. Predicting trust and distrust in social networks. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)* (2011), IEEE, pp. 418–424.

[14] EDLER, D., AND ROSVALL, M. The mapequation software package. available online at `http://www.mapequation.org`, 2013.

[15] FACEBOOK. Facebook quaterly earnings slides, q4 2012. `http://www.scribd.com/doc/123034877/Facebook-Q4-2012-Investor-Slide-Deck`, 2012.

[16] GAL-OZ, N., GRINSHPOUN, T., GUDES, E., AND MEISELS, A. Cross-community reputation: Policies and alternatives. In *Proceedings of the IADIS International Conference on Web Based Communities, Amsterdam, The Netherlands* (2008), pp. 197–201.

[17] GOLBECK, J., AND HANSEN, D. Computing political preference among twitter followers. In *Proceedings of the 2011 annual conference on Human factors in computing systems* (2011), ACM, pp. 1105–1108.

[18] GOLBECK, J., KOEPFLER, J., AND EMMERLING, B. An experimental study of social tagging behavior and image content. *Journal of the American Society for Information Science and Technology 62*, 9 (2011), 1750–1760.

[19] GOLBECK, J., ROBLES, C., EDMONDSON, M., AND TURNER, K. Predicting personality from twitter. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)* (2011), IEEE, pp. 149–156.

[20] GOLBECK, J., ROBLES, C., AND TURNER, K. Predicting personality with social media. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems* (2011), ACM, pp. 253–262.

[21] GUHA, R., KUMAR, R., RAGHAVAN, P., AND TOMKINS, A. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web* (2004), ACM, pp. 403–412.

[22] HENDRIX, F., AND BURBENDORFER, K. Malleable access rights to establish and enable scientific collaboration. Unpublished paper, submitted to eScience 2013 Conference.

[23] KEWALRAMANI, M. Sentiment analysis on twitter.

[24] KLOUT. Klout.com. `http://klout.com`.

[25] KOSINSKI, M., KOHLI, P., STILLWELL, D., BACHRACH, Y., AND GRAEPEL, T. Personality and website choice. In *ACM Web Science Conference* (2012), pp. 251–254.

[26] KOSINSKI, M., STILLWELL, D., AND GRAEPEL, T. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences* (2013).

[27] KUNEGIS, J., LOMMATZSCH, A., AND BAUCKHAGE, C. The slashdot zoo: mining a social network with negative edges. In *Proceedings of the 18th international conference on World wide web* (2009), ACM, pp. 741–750.

[28] LESKOVEC, J., HUTTENLOCHER, D., AND KLEINBERG, J. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web* (2010), ACM, pp. 641–650.

[29] LURIE, I. How to:scrape search engines without pissing them off. `http://searchnewscentral.com/20110928186/General-SEO/how-to-scrape-search-engines-without-pissing-them-off.html`, 2011.

[30] MOVSHOVITZ-ATTIAS, D., MOVSHOVITZ-ATTIAS, Y., STEENKISTE, P., AND FALOUTSOS, C. Analysis of the reputation system and user contributions on a question answering website: Stackoverflow.

[31] NAJAFI, M., SARTIPI, K., AND ARCHER, N. Web service competition: a new approach to service selection. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research* (2012), IBM Corp., pp. 161–175.

[32] OPENID. `http://openid.net/`, 2013.

[33] QUERCIA, D., KOSINSKI, M., STILLWELL, D., AND CROWCROFT, J. Our twitter profiles, our selves: Predicting personality with twitter. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)* (2011), IEEE, pp. 180–185.

[34] QUERCIA, D., LAMBIOTTE, R., STILLWELL, D., KOSINSKI, M., AND CROWCROFT, J. The personality of popular facebook users. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (2012), ACM, pp. 955–964.

[35] RESNICK, P., KUWABARA, K., ZECKHAUSER, R., AND FRIEDMAN, E. Reputation systems. *Communications of the ACM 43*, 12 (2000), 45–48.

[36] RESNICK, P., AND ZECKHAUSER, R. Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. *Advances in applied microeconomics 11* (2002), 127–157.

[37] SABATER, J., AND SIERRA, C. Review on computational trust and reputation models. *Artificial Intelligence Review 24*, 1 (2005), 33–60.

[38] STELZNER, M. 2013 social media marketing industry report. `http://www.socialmediaexaminer.com/social-media-marketing-industry-report-2013/`, 2013.

[39] TSAI, W., CHEN, Y., BITTER, G., AND MIRON, D. Introduction to service-oriented computing. *Arizona State University* (2006).

[40] WADWHA, T. Lessons from crowdsourcing the boston bombing investigation. `http://www.forbes.com/sites/tarunwadhwa/2013/04/22/lessons-from-crowdsourcing-the-boston-marathon-bombings-investigation/`, 2013.

[41] WARDEN, P. How i got sued by facebook. `http://petewarden.typepad.com/searchbrowser/2010/04/how-i-got-sued-by-facebook.html`, 2010.

[42] XU, Y., CAO, X., SELLEN, A., HERBRICH, R., AND GRAEPEL, T. Sociable killers: understanding social relationships in an online first-person shooter game. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work* (2011), ACM, pp. 197–206.

[43] ZIEGLER, C.-N., AND LAUSEN, G. Propagation models for trust and distrust in social networks. *Information Systems Frontiers 7*, 4-5 (2005), 337–358.

Appendix A

Appendices

A.1 Twitter Data Schema

```
 <?xml version="1.0" encoding="ISO-8859-1" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="profile">
<xsd:complexType>
<xsd:sequence>

<xsd:element name="key_values">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="number_followers" type="xsd:string"/>
<xsd:element name="number_tweets" type="xsd:string"/>
<xsd:element name="number_following" type="xsd:strng"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="tweets">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="tweet" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="tweet_content" type="xsd:string"/>
<xsd:element name="retweet_count" type="xsd:positiveInteger"/>
<xsd:element name="favourite_count" type="xsd:positiveInteger"/>
<xsd:element name="retweet_names">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="name" minOccurs="0" maxOccurs="unbounded" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:attribute name="tweet_id" type="xsd:string" use="mandatory"/>
```

```
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:sequence>
</xsd:complexType>
</xsd:element>


</xsd:schema>
```