JAMES MENEGHELLO

A SCALABLE FRAMEWORK FOR INTEGRATED SOCIAL DATA MINING

A SCALABLE FRAMEWORK FOR INTEGRATED SOCIAL DATA MINING

JAMES MENEGHELLO

Murdoch
UNIVERSITY

This thesis is presented for the degree of:

Doctor of Philosophy (Ph.D)

School of Engineering and Information Technology

Murdoch University

2016

DECLARATION

I declare that this thesis is my own account of my research and contains as its main content work which has not previously been submitted for a degree at any tertiary education institution.

Perth, Western Australia, 2016

James Meneghello

ABSTRACT

Social Networking Sites (SNS) are ubiquitous within modern society, forming communications networks that span across cultural and geographical boundaries. The information posted to these sites provide useful insights into individuals, but can also provide a wealth of information that can be used for further analysis into the surrounding environment. Three main challenges limit the use of this information in applications: the quantity of data is often unmanageable, there is a significant amount of data unavailable for use due to a lack of generic interfaces for access, and there is difficulty in integrating multiple disparate social data sources.

The overall aim of the research described in this thesis is to advance the field of data science and improve accessibility of social data in analytical applications, in both academic and commercial settings. This aim has been addressed with three primary contributions; new algorithms to efficiently locate and collect relevant social data, new methods of performing unsupervised data extraction from generic social sites, and the development and subsequent empirical evaluation of a framework to facilitate the collection, integration, storage and presentation of social data for use in applications.

The first contribution was the presentation of a search query optimisation algorithm designed to reduce the amount of noise resulting from social data collection by learning from collected content and iteratively building new query keyword sets. The algorithm was empirically evaluated and the results indicated that it

provides significantly more data than existing search tools while minimising signal-to-noise ratio.

The second contribution aimed to improve access to social data available on Web 2.0 sites but without any existing interface access to the data. The algorithm is designed to extract social data from sites without any *a priori* knowledge of design or page layout. Its efficacy was empirically evaluated against a testbed consisting of popular news and current affairs websites. Results indicated that the algorithm was very effective at unsupervised retrieval of social data.

The third major contribution presented a framework that integrated the previous two contributions into a framework designed to streamline use of social data in academic and commercial applications. The generic, component-based design was evaluated in real-world scenarios and determined to provide a full social collection and analytics workflow in an extensible and scalable manner.

This research has theoretical and practical implications for the use of social data in analytical research and commercial use. It extends the data extraction field to include user-generated content, while providing new avenues for performing semi-intelligent social data sourcing, and significantly improves the accessibility of social data.

PUBLICATIONS

Meneghello, J., Lee, K., & Thompson, N. (2014). Towards Social Media as a Data Source for Opportunistic Sensor Networking. Presented at the Australasian Data Mining Conference (AusDM2014), Brisbane, Australia: Australian Computer Society.

Under review:

Meneghello, J., Thompson, N., & Wong, K.W. (2016). Unlocking analytical value from social media and user generated content: Solutions for data extraction and feature detection of semi structured sources. Currently under review with Data & Knowledge Engineering (Elsevier).

Oh, hello there. I will stay behind, to gaze at the sun.
The sun is a wondrous body. Like a magnificent father!
If only I could be so grossly incandescent!

— Solaire of Astora, From Software's *Dark Souls*

ACKNOWLEDGMENTS

Thanks to Nik, Kevin and Kevin for the advice they gave and the amount of work they put in.

Mabel, for being my best friend and keeping me sane and smiling.

Hannah, for always ensuring I had someone to rant to when I needed it.

My parents, for putting up with me for so long.

All of my other friends - you guys are the best.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 OVERVIEW

Social media has begun to form an integral part of our society. Social networking sites (SNS) are the most common interface to social media, used by hundreds of millions of people per year - nearly 65% of US adults (Perrin, 2015). Users share aspects of their life, including pictures, video and general comments, to friends and the wider public. This provides a rich, available source of data from users across the world, often with embedded metadata such as location, age and gender. While this is traditionally heavily used for marketing, new social analytics fields are rapidly expanding the applications leveraging this data. The analysis of this rich source of data is commonly referred to as "social data mining" (Amento et al., 2003).

While much of this data is relevant only to those with a social connection to the author, publicly-posted data has previously been used to great effect. Social sensing uses social media posts from users to detect events - social, geopolitical or environmental. It has been used to organise civic protests, such as during the Arab Spring (Huang, 2011), to discover and isolate disease outbreaks (Parker et al., 2013; Kautz, 2013; Li and Cardie, 2013), to detect earthquakes (Robinson et al., 2013; Aki, 1995; Sakaki et al., 2010), and manage bushfire maps (Cameron et al., 2012). All of these applications use

public posts by users to detect events with much more widespread use, with a number of early applications specifically focusing on the ability to detect emergencies from social media posts, such as using Twitter to provide rapid response during the Boston Marathon Bombing (Cassa et al., 2013).

In addition to widespread event detection, applications have been developed that focus specifically on individual users. Previous work has provided early detection of depression (De Choudhury et al., 2013b) and post-partum depression (De Choudhury et al., 2013a) by examining users' Twitter feeds, demonstrating that the applicability of social data mining and analytics ranges from society-scale to individual use. Hence, there is significant focus in research on the benefits that can be provided by analysing social media.

While there has been a wide range of social analytics applications developed, the majority of studies are limited to a single data source (as discussed in Section 1.2). This is primarily due to the challenge of integrating multiple disparate and heterogeneous data sources, which is outside the scope of most research. It is highly likely that if data collection and integration between multiple social networks was simpler, authors would be more inclined to use data from a broader range of sources - an important consideration, given that different social networks cater to different demographics (Greenwood et al., 2016) and relying on a single source can skew results.

The aim of the research presented in this thesis is to advance the field of social data mining by facilitating the utilisation of heterogeneous SNS and social media as data sources for social analytics. This research has significant practical and research implications by dramatically expanding the set of easily-accessible

social data sources available for use in social analytics. It provides the ability to use multiple diverse data sources for research purposes, eliminating demographic bias between individual social networking sites. It also vastly expands the potential for using users as sensor nodes in wide-scale event detection. It does this by providing the design of a framework capable of collecting, integrating and processing social data from disparate sources into a format suitable for use in real-time analytics, developing techniques for exploiting previously-inaccessible social data sources and optimising the process of finding social data to enhance efficiency and reduce the need to handle irrelevant data. The new algorithms and techniques presented in this research are then evaluated against real-world data for efficacy and to examine their potential use in both research and commercial applications.

1.2 BACKGROUND

Interaction with social media has emerged as a regular activity for many Internet users (We Are Social Singapore, 2014). A vast and diverse range of information is shared between participants, relating to many different topics. Users share life events, rich media and thoughts over social media, as well as managing their schedules. The near-ubiquitous acceptance of social media within developed society has led to strong userbase growth across all platforms, with 56% of Americans across all age groups now using at least one social network and 96% of Americans aged between 18-35 (FormulaPR, 2011; Edison, 2012). Globally, 26% of the population use social media, including 57% of Australians and 46% of Chinese (We Are Social Singapore, 2014).

The data made available on SNS comes in many different shapes, sizes and formats. At their simplest, the networks are used as a communications platform in which short messages between users or within groups are passed around in plain-text. With the integration of media content delivery networks into social media, entirely new streams of data have become available, including video, photos, hyperlinks and games. These streams are data are often uploaded by users and passed around amongst networks through friend connections and public links, with some media going "viral" - in which a piece of content is continually passed across friend networks and reaches significant public exposure (Guerini et al., 2011).

Not all of this data is publicly available or easily accessible. Users often upload private photos and only share them to a subset of their friend connections, and thus are not available to users outside those circles. Even accounting for this portion of private data, social networks share an enormous amount of content and user information publicly (DOMO and Column Five Media, 2012) - data that can be leveraged for other purposes - such as integration in sensor networks. In addition, a significant amount of data created by users is published onto Web 2.0 sites that do not provide Application Programming Interfaces (APIs) [1] to access user-generated content, such as news websites.While data privacy is paramount, data that is not intentionally hidden can be made accessible to provide additional data sources for sensor networking.

In order to use social media data as a data source for sensor networking, data across the disparate range of social platforms must be integrated into a cohesive dataset. Not only does this reduce the complexity of having to manage structurally-diverse platforms as

1 APIs allow computer programs to directly interface and query data sources, providing a simple method of accessing information related to a site or platform.

separate entities, it prevents a number of shortfalls that occur within existing social media research - primarily, that most research only occurs over a single platform, severely limiting the demographic of participants. Using Google Scholar, the citations of the three most cited social event detection papers (Fung et al., 2005; Chen and Roy, 2009; Sayyadi et al., 2009) were analysed. Of the 140 papers related to social event detection, only 6 papers used more than a single data source during the experimental phase. By abstracting away the platform entirely, the demographic is expanded to include all users of social media. The integration process can also enable filters to be applied to data - these filters can be comprised of a number of algorithms designed to clean data produced by sensor networks (Jeffery et al., 2006; Deligiannakis et al., 2008), as well as providing functionality for data enrichment functions such as sentiment and gender analysis[2].

Existing data querying and analysis technology can be used to handle interaction with the integrated dataset, including database query engines (Arasu et al., 2006). These engines make it possible for users to execute queries to select, analyse, process and output desired data. This completely abstracts the data sources being queried, removing any need for the user to manually interact with the data or specifically tailor queries to particular data sources. Query languages executed by a database engine provide a generic interface, granting the ability to perform virtually any analysis of the data required.

Systems designed to collect, integrate and query social datasets already exist, but involve a number of trade-offs that can significantly affect their usage within applications (DataSift, 2010;

2 Sentiment analysis provides insight into how speakers feel towards the topic analysed by returning positive and negative sentiment scores. Gender analysis attempts to determine the gender of a speaker through name analysis and NLP.

Gnip, 2008). All of these systems function as Extract-Transform-Load (ETL) (Vassiliadis, 2009) processes, which can take significant amounts of time to make new data available for use. Social media data, which is nominally event-based and real-time, is effectively flattened into delayed data warehouses. This additional latency can compromise any potential use of real-time monitoring of integrated social sources, necessitating use of low-latency real-time streams straight from the sources themselves. Solving this problem will open up new applications in social sensing and data science.

1.3 AIMS

The research presented in this thesis is intended to advance the field of social data mining, by investigating and evaluating the use of social data as part of a sensor network. In order to leverage social data for this use, various data-cleansing and integration techniques can be adapted from traditional sensor networks, since they can be considered conceptually similar to social networks.

Previous research on social networks has involved very specific querying of single social networks, since there is a lack of interoperability between social media platforms. It would be desirable to integrate social networks and other social data sources into a unified data-set to ensure the widest possible coverage of users, devices and geographical area. To this effect, some method of agglomerating data from differing social data sources should be determined. In addition, new data sources should be located and integrated to further broaden the scope of social data sources available for use.

The aims of this thesis are as follows:

AIM 1: **Develop and evaluate methods of efficiently collecting and integrating generic social data for analysis.**

Data collection and analysis scripts used in current research have a number of issues. Due to integration difficulty, research using social analytics often only operate using a single SNS as a source (Robinson et al., 2013; Sakaki et al., 2010; De Choudhury et al., 2013b). This results in heavily skewed demographics and limited geographical and cultural variety. For global analytical applications, this is an important consideration. Hence, a method of collecting, integrating and presenting generic social data in an accessible manner is desired.

While some commercial applications (Gnip, 2008; DataSift, 2010) are capable of performing collection and integration of social data sources, they often have prohibitive or infeasible hardware requirements due to the sheer amount of social data available. Hence, it is also desirable that methods be developed to collect social data for sensor applications in an efficient and intelligent manner, only collecting data that is relevant.

AIM 2: **Develop new techniques to access alternative and untapped social data sources for use in social data mining.**

By further expanding the scope of social data sources able to be used in sensor networking, the range of potential applications is broadened. There are a significant number of data sources on the internet that allow the submission and publication of social data in the form of User-Generated Content (UGC) that do not provide data in an accessible manner, such as the comment sections of news

websites and blog comment systems. The data submitted to these sources is often timely and relevant, so collection would prove useful. While there are simple wrapper-based methods to extract data from individual sites (Ferrara et al., 2014), this incurs a significant development time to integrate each new source. Hence, it would be desirable to develop an automated method of extracting and presenting this social data in an accessible manner.

AIM 3: **Develop and evaluate a framework that integrates work from the previous two aims to provide an extensive and generic social data sourcing, collection and querying framework.**

After the methods for social data sourcing, collection, integration and querying have been evaluated, these techniques should be combined into a cohesive framework that supports the use of social data sources in sensor networking applications. This would then allow querying of social data for research purposes or for use in event-driven notification and alerting systems, such as those used in emergency response scenarios (Cameron et al., 2012).

1.4 JUSTIFICATION

There are benefits to supporting real-time querying and analysis of integrated social media datasets. News media was traditionally a slow communication - it took approximately a day for a journalist to investigate an event, and newspapers would publish it the following day (Meraz, 2009). Smartphones and Internet news have changed traditional media consumption habits due to much lower response times (Mayfield, 2008) to "societal events", such as emergencies,

disasters, disease outbreaks and other important events. By utilising smartphones and social media as a data source for detecting events, the reach of wireless sensor networking is extended to anywhere there are smartphones with a data connection. This reach is continually expanding as cheap consumer smartphones and data networks become more prevalent across the world (Voskresensky, 2013).

Developing new techniques for intelligently sourcing social data can dramatically reduce the amount of processing required to monitor social media. Early optimisation is paramount in data analytics, as it reduces the amount of data required to undergo integration, processing and serialisation. Hence, improving the relevancy of incoming data reduces the amount of extraneous data being processed, and can significantly reduce the level of hardware resources required for deployments, potentially opening up opportunities for new low-cost sensing applications.

Deploying bespoke collection networks to produce new data comes at a significant cost, as it can require new sensor hardware and processing resources. Leveraging existing data is preferred where possible, as collection cost is minimised. Social media can provide a plentiful source of data for sensor networking covering a broad geographic, demographic and cross-cultural area, but a significant amount of this data remains inaccessible due to a lack of site APIs for Web 2.0 sites. Developing new techniques to collect this user-generated content from generic sites can expand the scope of social data available to sensor networking to include user comments on news articles relating to current events, which can be particularly useful for event detection or disaster management.

Finally, developing a framework capable of sourcing, collecting, integrating, cleaning, storing and presenting generic social data can advance the field of data mining and social analytics by allowing studies to be easily run over multiple integrated networks. This reduces the potential for demographic bias specific to individual networks, providing a more holistic picture of social media users on the internet, as well as providing a framework for writing repeatable experiments and performing social analytics.

1.5 SCOPE AND ASSUMPTIONS

The scope of the research presented in this thesis is limited to developing new techniques for accessing existing (though inaccessible) data in an efficient manner, and using data integration techniques to present the collected data as a single unified schema. It uses software architecture techniques to provide an efficient, scalable and reliable solution to the challenges previously discussed.

This thesis presents new algorithms for data sourcing and extraction, but does not present new algorithms for specific event detection or data analytics tasks. Rather, it uses existing research to demonstrate the capability of the proposed framework to handle these tasks.

1.6 STRUCTURE AND ORGANISATION

This thesis is composed of seven chapters. To assist in navigation, Fig. 1 presents an architectural map of the proposed Social Media as a Sensor (SMAAS) framework, and indicates the parts of the framework that each Chapter represents.

Chapter 1 provides a brief introduction to the field of social data mining and the challenges faced in leveraging social data for sensor networking. The overall research aims and project scope are also detailed here. Chapter 2 examines the current state of fields relevant social sensor networking, including sensor networking, social media, social data mining, data sourcing, data extraction and event detection.

Chapter 3 discusses potential solutions to the broader aims described in Section 1.3 and defines a set of requirements for candidate solutions to meet. It presents the SMAAS framework, designed to intelligently source social data, perform unsupervised collection from generic sources, integrated data into a unified schema, allow for cleaning and post-processing and serialise the resulting social events - the combination of which satisfy all aims of the research.

Chapter 4 describes the intelligent sourcing algorithm used in the SMAAS framework to efficiently locate social data sources relevant to a predetermined topic. A new method of extracting the social data available from these data sources is then presented in Chapter 5.

The SMAAS framework, the intelligent sourcing and user-generated content extraction algorithms are then evaluated in Chapter 6. Two real-world scenarios are then designed to demonstrate the use of the SMAAS framework, including a real-time event detection algorithm and a delayed political sentiment analysis scenario.

Finally, Chapter 7 considers how the primary research aims have been met. The three contributions of the research to the field are discussed, including how the evaluation indicates that the framework advances the field of social data mining.

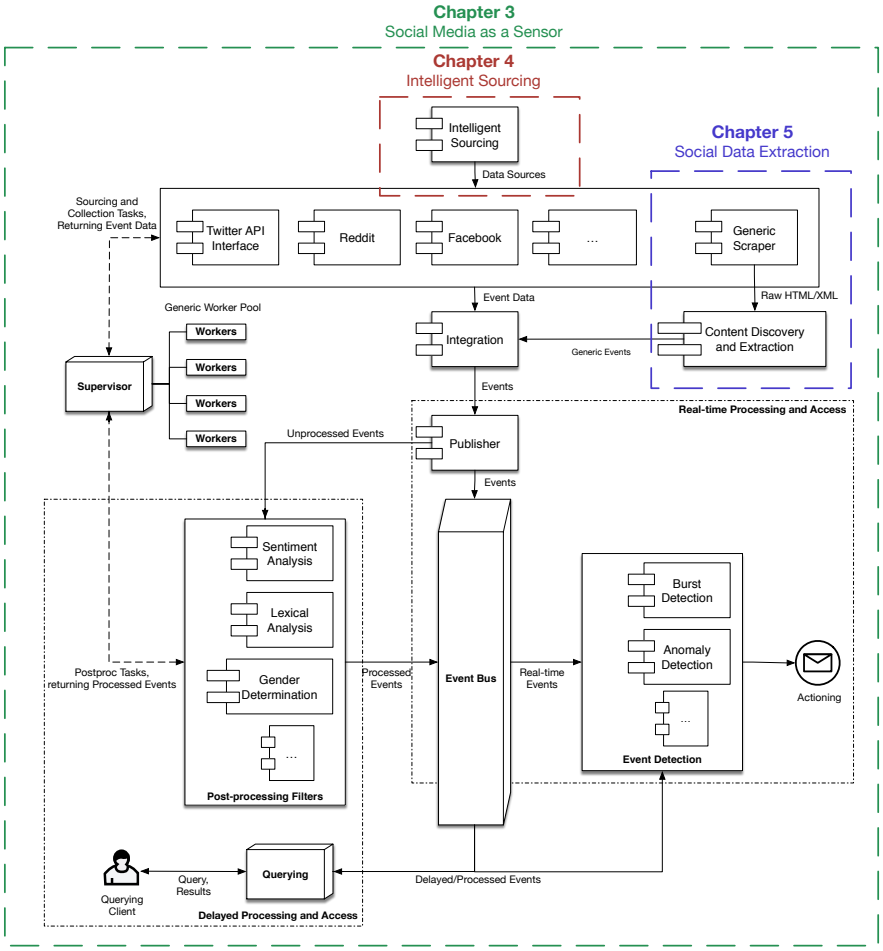


Figure 1: Architecture map to chapters in thesis.

This thesis makes a significant contribution to the field by making social data significantly more accessible for use in research and real-world applications, improving future social data mining research and opening up new opportunities for social sensor networking applications. This is proven to be viable through an extensive evaluation of the individual algorithms developed, as well as the use of the complete framework in real-world scenarios.

2

LITERATURE REVIEW

2.1 OVERVIEW

This chapter reviews the core concepts crucial to social sensing, in which four main research areas are considered. Section 2.2 examines the role of social media in society and its use in research, including methods of collection and analysis, and the concepts of sensor and social networks are conflated. Section 2.3 briefly explores existing data sourcing methods, while Section 2.4 reviews and examines the multitude of state-of-the-art data integration techniques, particularly where applicable to social data. Section 2.5 analyses the current state-of-the-art in web data extraction algorithms for retrieving social data from generic pages. The concepts and techniques described in this chapter form the foundational groundwork of the systems featured later in the research.

2.2 SOCIAL MEDIA

2.2.1 *Overview*

Social networks were originally developed to facilitate global communications amongst users, providing a persistent platform for people to re-connect with old friends and share their life in detail

online. After the popularisation of Web 2.0[1] in 2005 and the global shift from static web content to participatory sharing and communications (o'Reilly, 2009), social networks became highly integrated with most large web systems for third-party authentication and other means. Combining elements of communications networks, content delivery and content creation, social networks co-opted the term "social media" to more fully describe their role in modern society.

With much of the content on the Internet now being created directly by users in the form of media or discussion, there is an increased focus on utilising social media as a data source for decision-making. While advertisers and marketing have always been an integral part of social media (Stelzner, 2011), government entities are also tapping social media to drive the creation of populist policy proposals and gauge public reaction to social events (Hall, 2014).

Effectively deriving useful data points from social media is a topic of much discussion, and presents a number of challenges (Maynard et al., 2012). Posts by users on social media are usually free-form - the data comes in no particular standard format, as they often represent part of a stream of consciousness from a user. Additionally, posts are not limited to text and users often share pictures, videos, diagrams or graphs which present unique challenges to data analysis.

The proliferation of social media has driven content and context creation, but has also made it more difficult to locate appropriate data sources (Wandhöfer et al., 2012). Where topics were once discussed in semi-centralised locations such as Usenet, the integration of commenting systems into many different news and blog sites has dispersed information to this point where it is

1 Web 2.0 is widely seen as the shift in online communications from site-generated content to user-generated content, such as comments on news websites.

unrealistic to attempt collection of all relevant discussion relating to a topic. Instead, discussion centers can be discovered by tracking the spread of topics across the Internet and monitoring the most active communities.

Finally, the majority of social networks approach data storage and content formatting in dissimilar ways and often have little interoperability. There is no API standard to allow programs to easily retrieve data from social networks, and some do not provide an API at all. Because of this, data integration between networks can be difficult and there is no easy way of retrieving and mapping data structures from generic social networks.

2.2.2 *Social Media Demographics*

Many existing studies on using social media for event detection tend to focus on a single social network (Li and Cardie, 2013; Cameron et al., 2012; Sakaki et al., 2010; Robinson et al., 2013; Xu et al., 2016; Rosser et al., 2017), limiting collection specifically to the demographics represented on the chosen platform. Cultural demographics can vary significantly per platform - Facebook holds 79% of internet users in America and Twitter holds 24% (Greenwood et al., 2016), while Cyworld is estimated to host 50% of the social networking profiles of South Korean users, and Mixi is more popular in Japan while 20% of Orkut's population is Indian (Vasalou et al., 2010). In order to develop a truly global and cross-cultural social media monitoring system, data collection mechanisms should be extensible to any form of online social media, rather than a specific few platforms.

In order to realise this concept of global generic social sensor networking, a diverse set of data collection and integration techniques need to be utilised. Different social networking platforms operate on different data structures, using different storage engines and producing entirely different output. Even considering this, social media data is conceptually similar across platforms, consisting of a number of common constructs (users, friends, connections, messages, events). Because of this conceptual similarity, it is possible to integrate this data into a single queryable dataset through the use of data sourcing, collection and integration techniques.

2.2.3 *Data Collection*

Social media has previously been used as a data source in a number of research applications. Data collection and analytical techniques vary significantly between studies, with some studies opting for real-time collection and analysis (Sakaki et al., 2010; Li and Cardie, 2013) and others opting for more manual off-line processes (Wandhöfer et al., 2012). The approach taken is usually dependent on the use of the data - while purely analytical projects can afford to be off-line, alerting systems are real-time so that they can provide real-time alerts. In both instances, data collection usually uses two techniques: retrieving data from Web APIs and web scraping.

2.2.3.1 *Web Application Programming Interfaces*

Web development has largely shifted away from monolithic server-side web systems (Fielding, 2000), with many new technologies emphasising clear separation of server-side logic, client-side logic and presentation logic (MacCaw, 2011). One of the

key technologies involved in this shift has been the adoption of AJAX (Asynchronous Javascript And XML) (Garrett, 2005) to dynamically update client-side presentation by asynchronously polling server-side resources for updates in real-time.

To improve scalability of server resources and take advantage of little-used client-side resources, the server often sends only the minimum required data, with presentation left to the client. To easily facilitate this, server-side web systems now operate more as web services rather than traditional web systems, with many opting to present server resources using REpresentational State Transfer (REST) (Fielding, 2000). RESTful servers then provide application programmers with a standardised format for requesting data from web services. The data returned is also standardised, with most new services opting to return data in JSON (JavaScript Object Notation (Crockford, 2006)) format rather than XML.

Data collection tends to rely on these network-provided APIs to streamline experimental setup and provide standardised output for the chosen social network, such as the output from the Facebook API (Facebook, 2007) seen in Fig. 2. While this is to be encouraged in most cases (as that is what the API is provided for), there are some drawbacks to relying solely on API-provided data. In some instances, the API provides less information than is publicly available through the graphical user interface to the social network eg. Facebook's API does not always provide user location even if accessible through the user's About page.

```
{
  "_request": "GET /100001806305913",
  "id": "100001806305913",
  "name": "James Meneghello",
  "first_name": "James",
  "last_name": "Meneghello",
  "link": "https://www.facebook.com/murodese",
  "gender": "male",
  "locale": "en_GB",
  "updated_time": "2013-09-24T03:37:58+0000",
  "username": "murodese"
}
```

Figure 2: Data available from the Facebook API

Most web APIs follow similar principles, with similar authentication methods (such as OAuth1) and similar data formatting for results. While the output format of Facebook's API (Fig. 2) and Twitter's API (Fig. 3) appear similar due to the use of JSON, the underlying data structure differs greatly and requires semantic re-mapping prior to combined use.

```
{
  "_request": "GET
  ↪ /users/show.json?screen_name=murodese",
  "id": 158337652,
  "id_str": "158337652",
  "name": "Murodese",
  "screen_name": "Murodese",
  "location": "Australia"
}
```

Figure 3: Data available from the Twitter API

2.2.3.2 *Web Scraping*

Prior to the development and widespread introduction of Web APIs, data had to be taken from the user-viewable web page and parsed using regular expressions for desired information. This technique, commonly known as web or screen scraping, still enjoys widespread

use for websites that do not provide an API or those whose API does not provide requisite data.

As an example, the Facebook API does not provide location information through the API even if that information is publicly available. Because this information is still available on a user's public profile page, web scrapers are able to collect employment and education data from this page instead of the API. Using both the network APIs and web scraping, a more complete profile of the user is able to be constructed, allowing for enhanced associations between users by matching data that would otherwise be unavailable (such as mutual workplaces).

Web scraping has its own set of drawbacks - there is significant resource wastage compared to API access, as the scraper has to process a large amount of content included to format the data in a manner appropriate for user viewing. This content needs to be rendered regardless of the fact that scrapers are headless and have no need to actually display content, so this only serves to slow the scraper down. Scraping usually also requires the developer to manually write regular expressions or parsing code that can retrieve the desired data and strip out unnecessary clutter, which can be a significant time investment. In the event that both techniques are available, API access is usually preferred.

2.2.3.3 *Collection and Integration*

In order to utilise data from different web services, integration between social media platforms is required. While programmatic access to web services has vastly improved since the introduction of Web 2.0 paradigms, interoperability and data integration between social networks remains to be an issue. There is limited

interoperability of services provided for the purposes of open authentication (Facebook, 2014), content sharing is usually limited. There have been some attempts to apply semantic web[2] principles to the problem, including Semantically-Interlinked Online Communities (SIOC) (Breslin et al., 2009a) which describes social networks using the Resource Description Framework (RDF) to improve interoperability. While RDF has yet to reach widespread adoption and is unavailable for use with many social media systems, the data structures and principles in use provide a good platform upon which to base further work.

There are also challenges surrounding the matching of social media profiles between networks. The FOAF (Friend-of-a-Friend) Project (Brickley and Miller, 2000) attempts to extend Semantic Web efforts to social media by providing a base for user profile matching across networks. It does this by combining names and user metadata (such as location, email addresses or education details) to provide a more substantial set of data to improve accuracy of matches. As with the SIOC project, few social networks actively support such efforts and most do not provide FOAF output for users.

The majority of studies conducted using data collected from social media follow a reasonably similar process: conduct (manual or automated) searches of a social network, save or export returned data in a simple format, and perform analysis (online or offline) to determine answers to a particular query. Query engines have been developed for a range of other purposes (Khoury et al., 2010; Madden, 2012) with the intention of providing a standard querying interface and abstraction layer on top of complex datasets. Because this process is reasonably generic, there is an opportunity to develop

2 The Semantic Web is an initiative to rebuild the web to use entirely structured data formats.

a query engine for integrated social data that could improve the quality and quantity of data available to researchers using social media data.

2.2.4 *Event Detection*

One of the primary advantages of social media is the ease and speed with which information can propagate from a source to a large part of the Internet. While content that "goes viral" is often just entertainment content, news and current events also feature heavily in discussions on social media. In some instances, the content spread is just notification of the event, but has also previously been used to organise protests (Huang, 2011; Grossman, 2009), aid in the planning and deployment of emergency infrastructure during disasters (Cameron et al., 2012) and help to track the spread of disease amongst the populace (Li and Cardie, 2013; Chew and Eysenbach, 2010).

Events that are tracked by social media can be broadly sorted into two categories: global events and user-localised events. Global events are those triggered by discussion of a topic amongst a large number of users, such as earthquakes (Robinson et al., 2013; Sakaki et al., 2010) and societal uprisings (Jurgenson, 2012; Grossman, 2009), and can be discovered through monitoring trending topics. User-localised events are discovered by tracking individual users and deeply analysing submitted content, performing textual analysis to determine user state, such as depressive episodes (De Choudhury et al., 2013b).

Many existing implementations of global event monitoring use burst detection for phrases on social media. Essentially, the system

tracks trending topics and establishes a historical baseline for used words. An event is detected when a positive distortion in that baseline occurs - when people start using certain words much more frequently than normal (Robinson et al., 2013). Phrases such as "earthquake" or "quake" or "shaking" start to appear more frequently in a localised geographical area, indicating the presence of a seismic disturbance. How this data is stored and treated depends on the use - some studies opt to compress the data into a form only useful for alerting (Weng and Lee, 2011) and opting not to store whole data for future use, while others do store data but operate over a more limited subset of social media (Cameron et al., 2012; Sakaki et al., 2010).

User-centric event monitoring requires in-depth collection and analysis of data specific to a single user over a longer period of time, depending on the application. In the instance of detecting depression in Twitter users, one year's worth of historical data prior to official diagnosis was analysed (De Choudhury et al., 2013b) - a mean of 4533 posts per user. When considering applying this level of analysis to a broader group of individuals, some appreciation of the processing time required to perform analysis is garnered.

Both types of analysis are useful. Global events tend to be monitored in order to perform damage mitigation or optimise responses to emergency situations, potentially saving lives or reducing burden on infrastructure. User-centric events can be considered in a similar manner - widespread analysis of users can provide important foresight into problem demographics, aiding in the early treatment of mental illness and combating criminal activity. Due to the depth of analysis required for user-focused event monitoring, implementation of widespread user analysis would be

extremely difficult and require significant processing resources - but could also derive positive results.

2.2.5 *Participatory and Opportunistic Sensing*

Participatory sensing uses users with mobile devices as environmental event sensors (Burke et al., 2006) by encouraging users to gather, analyse and share local knowledge in what is commonly referred to as "crowdsourcing". By treating users as sensor nodes, participatory sensing takes advantage of resources that already exist for a number of purposes, including urban planning and policy development. Smartphones also come with a number of sensors that can be made available for participatory sensor networks, including important contextual metadata sensors: location and time.

Participatory sensing requires direct and active participation of users, which comes with a number of problems. One of the key issues with using people as nodes in sensor networks is that they cannot be treated as reliable. Users are able to choose whether to provide data on a requested topic, and may choose not to do so. Participatory sensing is named as such for a reason - without active participation, the system is unable to produce useful outcomes.

Malicious users may also be a potential challenge in deployments using participatory sensing. In some instances the data being produced may be tampered with in order to align with the goals of a particular group (Ubermotive, 2012), particularly if data is being collected and analysed to form the basis of governmental policy. Finally, data provided by humans is rarely in a standard format and usually has to be processed heavily to derive a useful data point.

While these processes have been continually improved, they are not completely reliable (Harper et al., 2009).

2.2.6 *Analysis and Deriving Intent*

Collecting large datasets from social media is generally not useful without extensive analysis. The sheer scope and quantity of data completely overwhelms any attempts to perform manual analysis without significant filtering. In order to combat this problem, further research is being performed into analysing data streams such as those provided by social media - an area typically known as natural language processing (NLP).

natural language processing is an automated approach to analysing text by utilising human-like language processing and machine learning techniques (De Choudhury et al., 2013b). NLP aims to provide an artificial intelligence-based approach to analysing natural text, such as that posted on social media by users. Social media analysis commonly uses NLP techniques to categorise and analyse posts for intent and attitude, enabling analysis programs to determine what users are talking about and how they feel about those things.

Some studies have previously used NLP-like techniques to perform opinion analysis on social media posts about political articles (Wandhöfer et al., 2012), including affective computing techniques that determined the overall "mood" of posts. Other studies have used similar statistical techniques to monitor the language and frequency of posts (as well as user metadata) to attempt to predict depression in Twitter users (De Choudhury et al., 2013b). The same authors have used similar statistical and analytical

methods to detect significant post-partum changes in mood amongst new mothers based on social media posts (De Choudhury et al., 2013a). There is growing interest in using NLP techniques to analyse social media at a population-scale for the public good, with some systems even going so far as to predict the spread of influenza based on a user's physical proximity to social connections that have expressed signs of illness in social media posts (Kautz, 2013).

For the purposes of this research, existing NLP implementations can be utilised to determine message intent and context. natural language processing toolkits exist for a number of languages, including NTLK (Bird et al., 2008) for Python and GATE (Cunningham, 2002) for Java. There are also toolkits for lexical analysis that do not use NLP techniques, such as Wolfram Alpha, which generally has to deal with linguistic fragments rather than full messages.

2.2.7 *Ethics of Social Data Collection*

One of the challenges around collecting social data from social media is whether privacy is a major factor around information that is publicly available. While the majority of data collection is performed on entirely-public datasets, many people still expect their conversations between friends to remain relatively local.

Previously, some studies that have collected large amounts of social data have attempted to anonymise both the data source and any users within the set, to varying degrees of success (Zimmer, 2010). If this data is released to the wider public, de-anonymisation can occur quite quickly, depending on the data involved and the scale of collection. Ultimately, users must expect that anything posted to public media

can be (and are) subject to large-scale collection and storage, and the data can remain archived for long periods of time.

Governmental organisations have also been known to mine social media for biometric data for use in both facial and pattern-matching applications (Lauder, 2015). Many users are unaware that their data is being used for this purpose, and it potentially opens the door to misuse of data by governmental and criminal elements - not every server storing their personal data will be as secure as the social networks that originally stored it. The usage of data after it has been published into the future is currently a concern (as seen in a number of "revenge porn" cases (Citron and Franks, 2014)), and will continue to be a concern into the future.

2.3 DATA SOURCING

2.3.1 *Overview*

There is a significant amount of data published on the Internet, but isolating high-quality, relevant data sources remains a challenge. The diversity of topics and general lack of guiding metadata attached to websites limit the relevance of collected data fairly substantially, even with well-crafted search terms.

This section examines techniques used to locate data sources on social media and the Internet, including efforts to improve the relevance of returned results and extract high-quality information from raw data.

2.3.2 *Existing Methods*

Software exists that integrates multiple data streams including social media for queryable use in applications, such as DataSift (DataSift, 2010) and Gnip (Gnip, 2008). Both of these platforms use Firehose [3] data for integration and querying - essentially, they receive direct streams of all actions on partner social networks and store them for historical purposes. While this ensures complete access to all available social data, it also requires immense hardware infrastructure investment to process and store incoming data streams - Facebook alone is producing almost half a petabyte of data per day (Ching et al., 2012). Such an investment is generally infeasible for use in a bespoke, isolated sensor network deployment.

Social research experiments tend to take a different route to sourcing useful data by relying on existing search APIs. Wandhöfer et al. use the Google Custom Search Engine to find news articles related to government policy for aggregation and analysis of user comments (Wandhöfer et al., 2012). Sakaki et al. use the Twitter search engine to return results only containing keywords related to earthquakes, classifying the results in order to detect real-time earthquake events (Sakaki et al., 2010).

Other experiments take something of a hybrid approach, by collecting the entire real-time stream of incoming data from a social network but performing basic preliminary filtering to narrow the scope of returned results. Robinson et al. restrict the Twitter real-time stream by a set of geographical locations in order to detect burst events relating to earthquakes (Robinson et al., 2013).

3 A "Firehose" is a direct real-time stream of data being produced by a network, effectively allowing those accessing the firehoses to monitor all actions on the network and record all social data being produced.

2.3.3 *Challenges*

Existing integrated social data providers (DataSift, 2010; Gnip, 2008) operate in an ETL process, providing semi-real-time access to integrated social data. This presents a number of problems for certain applications, particularly those requiring very low-latency access to data. High frequency trading algorithms, for example, are unable to use integrated social data due to the lengthy post-processing steps that can delay the delivery of data by seconds.

In addition, there is a limited number of providers authorised to provide firehose data from social networks. Access to data through these providers can be quite expensive, particularly for academic applications, and there are forced limitations on the amount of data able to be drawn from these services. For applications that need to ingest a broad spectrum of social data for long periods, use of these providers is potentially infeasible.

There are also no generic extraction methods available from these services, limiting their collection of data to those networks for which adapters have been developed and maintained. While this represents a significant portion of social data available on the Internet, it generally does not provide access to site-specific data sources such as product reviews or bespoke commenting systems.

2.4 DATA INTEGRATION

2.4.1 *Overview*

Data integration is the process of taking diverse sets of data from multiple sources and providing a user with a unified view of the

resulting dataset (Lenzerini, 2002). This is relevant to the problem of querying generic social networks, as each network is designed with different specifications and data structures. In order to properly query social media for a broad range of purposes, the datasets taken from social media must be presented in a unified manner, and data integration supports this goal.

Integration of diverse datasets has long been a pervasive challenge faced by academia and industry alike (Lenzerini, 2002). Industrial applications have used data integration to great effect, with substantial systems integration being performed during mergers or acquisitions or consolidation of information technology infrastructure assets. Academics have often performed manual data integration to combine datasets resulting from experiments, in order to draw comparisons between related data. Manual data integration often involves direct manipulation of datasets, including approximation or interpolation of missing data, while automated integration can use a number of techniques tailored to the situation, such as semantic mapping or machine learning classification.

The remainder of this section examines some relevant data integration techniques and their application to social media data, as well as required prerequisite steps such as;

- preprocessing data to clean it and ensure high data quality,
- integrating diverse datasets from different source schemas, and
- presenting a unified interface for data querying.

2.4.2 *Data Cleaning and Preprocessing*

Data mining can be used to discover and extract knowledge from extremely diverse datasets, including archived data, but the process of doing so can be challenging. Integrating datasets from long-standing systems pose a particular problem, as they often experience significant schema changes over their history - a problem that has also become prevalent with new systems that undergo extremely rapid growth. Repeated changes in schema without proper data curation can result in significant amounts of dirty data being left in a dataset, such as old location types, non-daylight-savings compliant datetimes and even typographical errors (Rahm and Do, 2000).

Schema changes can also result in a significant amount of missing data (Rahm and Do, 2000). Beginning the collection of metadata partway through a system's life can leave old results with blank fields, and no way of refilling those blank fields. Changes to field structure designed to improve relational design can also result in very confusing values being left in archived data. Data can also expire, such as user addresses or contact details.

All of these factors can dramatically affect the success of data mining and analytical operations. Some of these issues are more costly than others - using the results of analysis based on poor-quality data for decision-making can result in disaster, depending on the application. It is therefore imperative that data used in any operations is as high-quality as possible.

2.4.2.1 *Data Quality*

Determination of data quality is largely subjective, with data generally required to be fit-for-purpose. In some organisations, data quality is managed by data governance and document control teams whose sole purpose is to ensure that data remains of high-quality and able to support analytical needs. These teams work by developing a series of effective data quality metrics that allow them to evaluate existing data quality and provide a benchmark for improvement (Pipino et al., 2002). These metrics include:

VALIDITY How well the data conforms to defined rules and constraints. Many of these constraints are implemented as part of the database design process, such as unique constraints, foreign-key constraints or null constraints. This can also include whether the data adheres to validation constraints, e.g. "is the value between 1 and 10?"

ACCURACY If the data is an accurate representation of the truth. This often requires some kind of manual check: e.g. verifying the accuracy of email addresses requires users to validate them manually by sending an email to their address and requiring them to click a confirmation link.

COMPLETENESS Whether all relevant data is known. Using data cleaning to fill in missing data in this fashion is possible in some instances (e.g. using metadata from closely related records and making assumptions), but can produce misleading data when heavily extrapolating.

CONSISTENCY Whether the data is consistent across sources. When integrating multiple data sources and encountering conflicting information (e.g. one person with two different addresses), a

strategy should be developed to decide which source is more reliable - usually the newest.

UNIFORMITY How closely the data follows the same standard units of measurement. Values containing like-units (such as km/h and m/s) can be directly converted, but data is sometimes in completely different units hence is incomparable.

FRESHNESS How old is the data? Is it still relevant?

Data entry errors remain to be an issue even with trained and experienced operators. Social media poses a further problem, in which users are inexperienced and non-technical and so are even more susceptible to data quality concerns. It can also be difficult to determine whether social media content has been created by a user rather than a computer program for the purposes of advertising or spreading malware (Yardi et al., 2009). All of these issues conspire to lower the quality of data being produced by social media, demanding extensive preprocessing of social data before it is used in analytical applications.

2.4.2.2 *Cleaning Techniques*

The process of cleaning low-quality data can be broken into four general steps (Müller and Freytag, 2005):

AUDITING Discovery of data quality problems. This can be for individual rows (e.g. is this value acceptable, or does it require cleaning?) to more general queries, such as using statistical analysis to determine where data quality issues lie.

SPECIFYING Upon discovery of low-quality fields, specify the steps to improve each element to higher quality. This may require data conversion, value conversion or even lexical analysis to

determine the true, useful value of the field. This should describe the current states of the data and the desired final state, as well as the process to increase quality to the desired level.

EXECUTING Iterate over all low-quality data and execute the specification to convert it to the desired format.

CONTROLLING Migrate the data schema to our final result and enforce all appropriate constraints upon the data. In doing so, find any remaining data errors that exist within the dataset and amend the specification to eliminate these issues.

The techniques used to clean data prior to use in analysis are tailored specifically to the challenges presented by particular datasets. In some cases, this may require the development of conversion scripts for each affected field, while others may simply require a re-definition of the field type.

Fig. 4 presents an example process for cleansing an inconsistent enumerated field in a database table, broken into the four general steps mentioned previously. In this scenario, a field that was previously a plain-text free-form field has later been modified to only accept certain values (an example being a gender field, which may only accept MALE, FEMALE or UNSPECIFIED), but was never directly constrained to these values at a database-level.

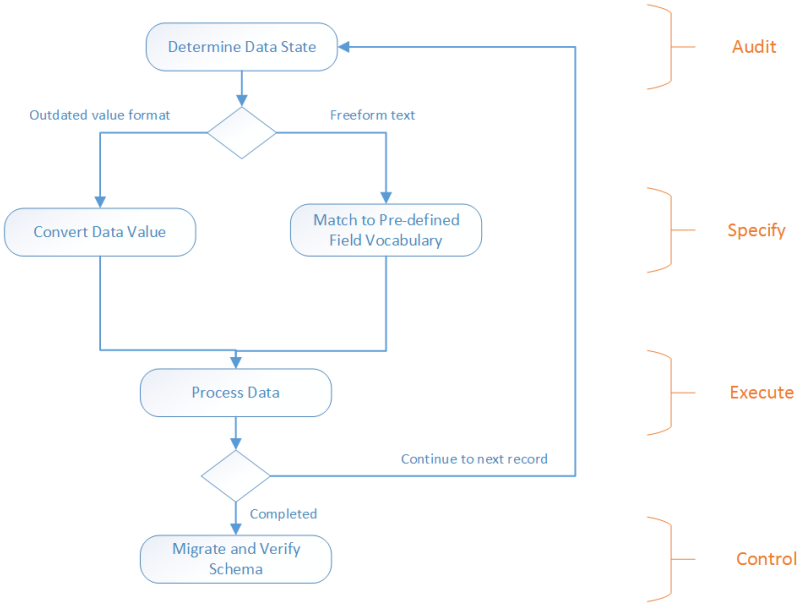


Figure 4: An example data cleaning process for one field

The following steps (represented in Fig. 4) can be used to convert this field with inconsistent values into a clean field:

1. Determine the current state of the data: is it in a form that we can already without conversion? Is the field's type correct? Does its value match to our predefined list of acceptable values? Can the data be used without modification?

2. Define a specification for the data, handling two possible cases:

   a) If the value is an old format, convert it to our new values. Some data precision may be lost, or conversion may be straight-forward (e.g. "other" becomes "UNSPECIFIED").

   b) If the text is an old format that was entirely free-form, attempt to convert it into an appropriate value (e.g. "m" or "male" or "man" or "Mr" becomes "MALE").

3. If value conversion was complete, continue to the next record. If all have been completed, migrate the field schema to enforce the

new standards. If the conversion failed, either delete the record entirely or set it to a default value (e.g. "UNSPECIFIED").

Each part of this process can be fluid and be comprised of many steps. Auditing can be a manual process in which a user personally checks for data quality issues in fields, or can be automated through the use of scripts that look for common problems. In most cases, examination of individual values to determine quality is automated, due to the volume of data being cleaned.

Specification development can be developed manually by writing mapping algorithms to convert datasets, directly handling the transition of data from its original state to the desired state. Mappings can also be automatically generated using mapping toolkits (Xu and Embley, 2004). Machine learning techniques can alternatively be used to classify values, allowing for more informed decision-making around conversion - though this usually requires a substantial training dataset (Doan et al., 2001).

The data cleaning process ensures that the multiple sources of data undergoing integration adhere strictly to a set of defined schemas, making integration a much less arduous and more reliable process. This also significantly reduces the complexity of integration systems, as it is far easier to integrate known data sources than inconsistent sets. After this preprocessing phase is complete, the cleaned data sets are integrated using several different techniques.

2.4.3 *Integration Techniques*

Data integration can be a complex process, depending on the complexity, relevancy and size of the converging datasets. Numerous techniques exist to handle the process of querying

integrated datasets, designed with different operating requirements. Some techniques require the offline transformation and storage of data for later querying, while others can handle-queries in real-time by inferring the goal of the query and transforming appropriate data as required. Some of these techniques are detailed below.

2.4.3.1 *Extract-Transform-Load and Data Warehousing*

Extract-Transform-Load (ETL) systems are commonly used in data warehousing, where data is initially cleaned and transformed for storage and later use (Vassiliadis, 2009). Due to the extensive processing that data undergoes in order to be in a clean state with unified schema, this process is generally not real-time, and can suffer from data freshness issues. This approach is therefore only useful for use in delayed queries, and its use with social media would exclude any real-time sensing systems.

The three primary steps of ETL are as follows:

EXTRACT Extract data from original sources and transport it for storage in data warehouses.

TRANSFORM Transform source data into the new schema, possibly including computation of new values, merging/splitting of fields and occasionally creation of new records based off existing data. This step also includes the isolation and cleaning of problematic tuples to ensure adherence to constraints, which is not completed as part of preprocessing in ETL.

LOAD Load the cleansed, transformed data into defined relations within the warehouse, including the construction of appropriate indexes and views.

These processes are manually designed and run at regular intervals to retain some measure of data freshness. Once stored, the data is presented to users as a unified dataset following a single schema that is able to be queried much like any other database, so the querying process is simple.

2.4.3.2 *Data Mapping*

Data mapping is a concept in which multiple different data sources are overlaid with a virtual schema that provides a mapping between the real data elements and the conceptual, queryable elements. There are two different approaches to this integration technique: Global-as-view (GAV) and Local-as-view (LAV). Both are based on the same formal integration system definition, in which: the system I is defined as a triple (G, S, M) where G is the global (virtual) schema, S is the heterogeneous set of source schemas, and M is the mapping that maps queries between S and G. Effectively, users create queries operating over G, and M handles the internal query rewriting to suit S (Lenzerini, 2002). The exact operation of M is defined by the technique used.

In all techniques based on this model, the concepts of mediators and wrappers are also introduced. Wrappers wrap external data sources in order to bring them to the format represented by S, while mediators handle the mappings controlled by M and define the global mediated schema, G.

Local-as-View: In LAV systems, M represents a mapping from entities in the original source schema S to entities in the virtual schema G. This approach scales well with the addition of new data sources, but introduces significant complexity into query processing and can result in significant performance losses during query

execution compared to GAV. It relies on the existence of a stable, well-formed global schema in order to support mappings. LAV prioritises scalability and the addition of sources over performance.

Global-as-View: In GAV systems, M represents a mapping from entities in the virtual schema G to entities in the original sources S. This has the benefit of allowing for much simpler and faster query processing, but requires updates to the global schema upon addition of new data sources. Scalability is a concern with GAV approaches, because the continuous addition of data sources requires constant updating of the global schema and leads to many of the data issues dealt with during integration preprocessing. GAV prioritises performance over scalability and the addition of sources.

Combinations: There are several data integration techniques that combine elements of these two concepts in an effort to reduce the penalties associated with them. BGLAV (Xu and Embley, 2004) is one such attempt that isolates the global schema by ensuring it is developed independently of any source schema S. In contrast, GAV approaches expand the global schema to ensure it includes all data elements from sources and LAV approaches minimise the global schema to only include common elements from sources. When new data sources are added, BGLAV uses automated schema matching algorithms to reduce the amount of wrapper development required, effectively automating (with oversight) the development of M. As there is a well-defined and unchanging global schema G (as in GAV), query processing is simplified. This approach is particularly useful for integrating social media, as independent data models have already been developed that can be adapted for use as the global mediated schema G, such as SIOC (Breslin et al., 2009a). Automated schema matchers can be applied to platform-provided Application

Programming Interfaces and the source schemas S to develop the wrappers supporting M.

2.4.3.3 *Ontology-based Mapping*

Ontology-based mappings work in much the same way as standard data mapping techniques (Noy, 2004), but are designed to integrate the flow of information across ontologies rather than data within integration systems. Ontology-based mapping expresses the map between the original ontology and the target ontology as either relational queries or a tuple map, allowing fairly direct translation from one ontology to another.

These mappings are of interest for use in social media integration because a number of social media networks have previously had their data expressed as part of ontologies, both specifically (in projects such as SIOC (Breslin et al., 2009a)) and in wider semantic web research (Mikroyannidis, 2007).

2.4.3.4 *Machine Learning Classification*

Machine learning algorithms have previous been used to find semantic mappings between different data schemas. Using these classification algorithms require a training dataset (an initial set of schemas with completed mappings) and can then attempt to perform auto-matching for future datasets (Doan et al., 2001). This is of particular use in social media integration, as the training set can consist of schematic mappings for initial network integrations (such as Facebook and Twitter), and the resulting algorithm can attempt to map the schemas for future networks.

Many of these algorithms can be implemented in conjunction with data mapping integration techniques to improve auto-mapping

abilities, reducing wrapper development time and improving the range of data sources able to be integrated into the main set.

Extracting data from web pages is a well-researched field that has been driven by the rapid expansion of online data presentation. There are two problem areas that have particular focus: automated interaction (commonly referred to as the "accessing the deep web"(Baumgartner and Ledermiiller, 2005)) and structure recognition/extraction, both of which are discussed at further length in Chapter 5.

2.5 DATA EXTRACTION

2.5.1 *Overview*

Data extraction is the process of finding and retrieving relevant data from structured and unstructured sources. While requesting data from a web API is simple and provides well-formatted structured data, collecting data from web pages and unstructured sources requires significant effort. This section evaluates the state-in-the-art of data extraction techniques designed to ease this process, and provide the user with well-structured data from unstructured sources.

2.5.2 *Mining Dynamic Sources*

To extract data from pages using data extraction algorithms, raw data must first be collected from the desired source. Early efforts toward this task consisted primarily of "web crawling", in which pages were collected using simple HTTP requests and parsed to

identify hyperlinks in an iterative process (Heydon and Najork, 1999). While the principles behind crawling are still in use, the rapid expansion of AJAX requests to fill page content (Ihm and Pai, 2011) has ensured that standard HTTP requests are no longer able to retrieve all relevant data from sites. In many cases, this approach only retrieves basic page structure and design that is used to render AJAX results into a complete page.

There have been a number of proposed solutions to this problem. Xia (Xia, 2009) proposes using an embedded headed browser (such as Firefox or Chrome) to render the page off-screen, including all javascript content. Increased usage of Test-Driven Development with heavily client-interactive web sites has also driven the development of automated testing tools that are able to utilise browsers to render dynamic content, such as Selenium (Holmes and Kellogg, 2006). Headless browser agents such as PhantomJS have also been developed to reduce memory and software requirements, of particular use when executing distributed tests from server clusters (Hidayat, 2013). Each of these methods allow automated scripts to collect post-rendered content from web sites.

Increased use of AJAX to render page content has introduced another concept: pagination. Where content was previously retrieved during the initial web request and immediately rendered, browsers now have to make additional requests to fill content panes. As additional data can only be requested after all associated Javascript has been retrieved and executed, this can lead to a significant lag time between first page request and content visibility. To reduce this lag time, dynamic data is requested in smaller chunks - rather than load 150 comments in a single request, the page instead requests the first 20 comments and then presents the user with a

series of numbered index buttons to request the rest of the comment stream. Hence, to retrieve web data from a page, scraping systems need to be able to navigate these pagination fields and directly interact with pages in a complex way.

2.5.3 *Mining Interactive Sources*

Complex interaction has been investigated in previous studies. A series of articles by Baumgartner et. al. (Baumgartner and Ledermiiller, 2005; Baumgartner et al., 2009) introduce Lixto, a software package that assists in the creation of data extraction wrappers including user interaction. This can be used to record user interaction and repeat it against other pages in order to collect data from pages, but the interaction must be manually trained. Wrapper languages have also been developed that support user interaction, such as OXPath (Furche et al., 2011), but also require manual rule creation. Projects such as Crawljax (Mesbah et al., 2012) enable web crawling through AJAX pages by evaluating possible user interactions, but do not directly support intelligent interaction - because the aim is to collect every page of a page, it tries to interact with *every* element on a page.

There has been no previous research on intelligent automated interactivity with pages for the purpose of social data extraction.

2.5.4 *Structure Recognition*

The majority of algorithms developed for Web Data Extraction focus on structure recognition, in which an attempt is made to group data regions on a page according to similarity. Both manual and

automated solutions to this challenge have been developed, each with benefits and drawbacks. Both types of solution tend to operate by extracting information from the page's Document Object Model (DOM) tree, which is a tree-based representation of page design and content.

2.5.4.1 *Manual Methods*

Manual wrapper generation involves the development of an interface wrapper for each individual page structure e.g. one wrapper per website. They often involve the use of Regular Expressions or XPath to extract data from pages, but entire languages and language frameworks have also been developed in early research towards WDE. One such framework is EDITOR (Atzeni and Mecca, 1997), which allowed users to write wrappers in a script-like language to extract data from HTML tags. Much of the functionality from frameworks like this was later built into DOM-parsing libraries. EDITOR was also used in further research to write extracted data directly into relation data tables (Crescenzi and Mecca, 1998). These solutions, as with others that required manual wrapper development, require wrapper creation for every individual data source as well as regular maintenance on the wrappers themselves. Any change in page design or structure would usually render the wrapper useless until updated.

To streamline this process of wrapper development and maintenance, several partially-automated and assistive solutions were developed. STALKER (Muslea et al., 1999) used supervised training sets to identify certain extraction rules that could be applied to pages, limiting user input to developing the training sets. Assisted wrapper development software like OLERA (Chang et al., 2004)

presented a visual interface to wrapper generation and tag alignment, allowing less technical users to perform data extraction. These wrappers still had to be maintained and individually developed, though the development process took less time than purely manual solutions.

More recently, manual methods have focused on providing end-to-end extraction, while making the level of user interaction required as small as possible. Such applications allow users to provide simple configuration files to extract data from sites such as web forums for pharmacological research (Audeh et al., 2017), though user intervention and configuration maintenance are still required.

2.5.4.2 *Automated Methods*

Further research focused on unsupervised wrapper generation and data extraction, as this would completely remove the need for developers to create and maintain wrappers. Research on these solutions focused on two primary methods: DOM tree comparison algorithms and visual identification of data regions.

Similar data regions on a page can be identified by comparing the similarity of different parts of the DOM tree. By breaking the tree up into branches and performing tree-similarity comparisons between branches, repeating structures on the page can be identified. These structures often contain template-generated data, which often represent data regions. In identifying these regions, there were two approaches: single-page algorithms and multi-page algorithms.

2.5.4.3 *Single-Page Wrapper Induction*

Single-page data region identification algorithms used a single DOM tree to derive a wrapper and extract data, meaning that the

algorithm was required to focus on repeating structures within the same page. Pages containing a large number of data rows are useful for this approach, as the same type of visual container is used for each data row - resulting in a large number of similarly-structured DOM branches. Mining Data Records (MDR) (Liu et al., 2003) was an early attempt of this approach that used a tree edit-distance comparison algorithm to identify the distance between branches. Branches that had minimal distance between them were considered similar structures, and were grouped for data extraction.

Data Extraction through Partial Tree Alignment (DEPTA) (Zhai and Liu, 2005) builds on MDR by using partial tree alignment to identify optional and mandatory data fields within structures, improving the reliability of extraction. This assists in the identification of fields such as special product sales prices, which show up optionally, versus the standard price which should be present on every item on an e-commerce site.

More recent techniques have also been developed that build in text-mining or subject-identification algorithms to improve accuracy of wrapper generation, such as Bottom-Up Wrapper (BUW) (Thamviset and Wongthanavasu, 2014a). Because these algorithms work on a single page, some difficulty can be encountered in instances where a page contains a small number of data records, because they are often unable to sufficiently differentiate themselves from other elements of page design.

2.5.4.4 *Set-based Wrapper Induction*

Multi-page data region identification algorithms used a set of similar pages to generate extraction wrappers. Given a number of pages built using the same page template, some of these algorithms

examine the differences between pages to assist in aligning data records (Crescenzi et al., 2001; Arasu and Garcia-Molina, 2003), while others use multiple pages as a training set to develop wrappers (Ma et al., 2003; Sleiman and Corchuelo, 2014). Because these pages require multiple pages from the same site to design a wrapper, they are often unsuitable for fire-and-forget extraction duties, and are more suited to use in web crawlers that collect data from entire websites.

Visual extraction algorithms attempt to identify data regions by identifying visually-similar regions. VIDE (Liu et al., 2010) uses this approach to identify data regions and then performs data alignment between similar regions to identify optional fields.

2.5.4.5 *Hybrid Approaches*

There are also algorithms that combine elements of the three previous approaches. Generalised Simple Tree Matching (G-STM) (Jindal and Liu, 2010) can automatically develop wrappers from both single pages and multiple-page sets. Tag Path Clustering breaks down the DOM tree into a signal representing tree depth and uses signal analysis to locate repeating patterns, which indicate repeating data regions (Miao et al., 2009).

2.5.5 *Machine-learning Extraction*

In addition to these traditional approaches, the rise of machine-learning-based (ML) algorithms has led to their use in many different fields, including data extraction. The majority of these algorithms are supervised learning algorithms, which require a set of tagged training data to teach the model before use. Some of

these perform this task by asking the user to provide a basic set of sample input, which it uses to infer the correct extraction regions (Raza and Gulwani, 2017). Others use deep learning to train a generalised extraction model (Gogar et al., 2016), though the ability of learning-based models to extract data accurately is highly dependent on the quality of training data supplied to it. Others focus more specifically on text-mining, retrieving data from unstructured text (Poria et al., 2016) - but without the ability to extract object-oriented data.

2.5.6 *Social Extraction*

There is little research that specifically focuses on the extraction of social data from pages. One study collected social data from weblogs for analytical purposes, but used manually-developed wrappers to collect the data from a small number of data sources (Kao and Chen, 2010). Likewise, another study examined the identification of high-quality social data, but used a manual wrapper for Yahoo! Answers, a single data source (Agichtein et al., 2008).

Most of the web data extraction techniques previously mentioned are specifically designed to locate tabular data or data records, which do not precisely fit social data. While these techniques may be used to identify the blocks that store social data, isolating it from irrelevant results requires further processing.

Social data (such as user comments) often appear differently on the page to the data targeted by these collection algorithms, which are primarily designed to extract tabular information from pages. User comments are often hierarchical and are made up of far more

complex fields than tables, requiring a more in-depth collection and filtering algorithm.

2.6 AREAS OF IMPROVEMENT

As highlighted throughout this chapter, there are a number of key gaps in the currently available research:

1. The majority of studies using social data focus on the use of a single social data source, due to difficulty of collection and integration (see Section 1.2). Even using APIs for collection is rare, due to diverse data schemas between platforms. As a result, population demographics are limited to the platform used.

2. There is a significant amount of social data currently unused due to difficulty of access, such as social comments on news or eCommerce websites, because there is no API to access them and collection would require the writing of a manual scraping wrapper per-site (see Section 2.5.6).

3. Collection of social data can be challenging due to the sheer scale of data available, putting any large-scale social data collection efforts out of reach of most researchers (see Section 2.3.3).

4. Existing methods of integrating multiple social data sources are high-latency, providing integrated data too slowly to be used in some applications, such as high frequency trading algorithms (see Section 2.3.3).

This thesis aims to develop solutions for these challenges. Chapter 3 details a set of requirements for a framework aimed at solving

these problems, and provide additional benefits for both academic and commercial social data mining applications.

2.7 CONCLUSION

This chapter has presented a review of the state-of-the-art research in key areas of interest relevant to the work described in this thesis. The chapter considers the ability for a social network to function in the same role as a sensor network, using users under passive observation as sensors for a range of applications. Techniques for finding, collecting, integrating and cleaning the resulting social data was explored for use in the following chapter, which defines a set of primary requirements needed to facilitate the primary aims of this thesis, and presents a framework to enable the use of social data in sensor networking.

In the process of completing this review, social media was identified as a potential source of data for global opportunistic sensor networking, but key challenges to sourcing, collecting and integrating social data were identified. The initial method of data sourcing used dictates the amount of processing resources required to integrate, clean and post-process the data collected, as early filtering can dramatically reduce the set of data sources to be collected from. Collecting data from identified sources also poses a challenge, as most existing solutions use manually-written bespoke wrappers and only collect from a set of predefined sources. The next chapter describes a framework to source, collect, integrate, clean and query social data that attempts to alleviate these considerations.

3

SOCIAL MEDIA AS A SENSOR

3.1 OVERVIEW

The use of Social Networking Sites (SNS) is ubiquitous within modern society, providing communications that span across cultural and geographical boundaries. Users post a large quantity of information online about their lives, environment and thoughts. This information provides useful insights into individuals, but can also provide a wealth of information that can be used for further analysis of the surrounding environment (Breslin et al., 2009b).

Sensor networking applications collect useful data from the environment using sensors, requiring new deployments of sensor nodes for new applications (Akyildiz et al., 2002). These deployments can be costly, and the deployed networks rarely support generic applications. To avoid the cost of new deployments and support new applications without the provisioning of new hardware, existing datasets can be used. By using the data produced by SNS users in sensor networking, applications can reach anywhere that social media is used (Burke et al., 2006).

This chapter presents a justification of the use of social media as a data source for sensor networking. Functional and quality requirements to support this goal are defined. A framework designed to source, collect, integrate and query social data in adherence to these requirements is proposed, intended to integrate

with sensor networks and perform both historical and real-time querying and event-detection.

3.2 BACKGROUND

The scope of sensor networking applications is constrained by the availability of data sources. Supporting new applications often requires new deployments of sensor hardware that are able to collect new types of data. This data can then be used in analytical applications or used to extend and enhance existing applications. Using existing data sources to enhance sensor networking is potentially beneficial, as this does not require the expense associated with developing, implementing and maintaining a new sensor deployment.

SNS are candidates for use as data sources in sensor networking. They are comprised of a number of isolated networks covering much of the global population, providing better coverage within diverse communities. They produce very large amounts of variable-quality data with accompanying metadata. When properly filtered and cleaned, they can be a source of good-quality, relevant data for use in sensor networking applications. SNS also follow similar design patterns to sensor networking, being primarily event-based, and can conceptually be treated much the same way as a traditional sensor network (Sakaki et al., 2010).

A major obstacle to the widespread use of social data in sensor networking is the need for the integration of disparate SNS. There have been attempts to ease the migration of data between SNS, as identified in Section 2.4, but these have generally had little industry support. Most SNS have an interest in "locking-in" customers

(Zauberman, 2003), to dissuade them from migrating between networks, taking with them associated advertising revenue. Improving data integration processes would ease migration between competing networks, so efforts to standardise export formats are unlikely to ever achieve industry co-operation. Therefore, new techniques supporting inter-network social data integration need to be developed in order to facilitate the integration of social data into sensor networking.

3.3 BESPOKE VS REPURPOSED SOURCES

Data collection for sensor networking falls into two categories; i) bespoke deployments, and ii) re-use of existing data sources. Bespoke hardware deployments require the creation and installation of sensor hardware designed to collect data specific to the application, and can incur significant hardware and deployments costs. Repurposing existing data sources alleviates those costs, but locating relevant and accessible sources can be difficult.

Bespoke deployments are appropriate when there are no existing methods of collecting data required for research, and collection requires abnormal processes (such as sensors able to withstand immense heat). For example, deployments such as these have previously been used to monitor volcanoes (Werner-Allen et al., 2005). Wireless sensor motes were constructed specifically for the application and deployed over an active volcano. Sensors collected vibrations using specialised infrasonic sensors and transmitted the results wirelessly back to a monitoring station. Results were then used to map volcanic eruptions, providing early warning to surrounding population areas. In this case, the sensor hardware was

developed for a specific application and only monitored a very specific set of environmental data, and so the network is unlikely to be used for other applications.

When appropriate data sources are available, existing devices and networks can be re-purposed to good effect. This technique is used by Google Maps, which repurposes existing devices such as GPS locators in smartphones (Google, 2009). Users that are running Google Maps automatically send data (including location, direction and speed) back to the Maps servers, allowing optimal routing calculations based on current traffic conditions. The broad scope of environmental data collected by smartphones allows for their re-use in similar applications, such as road surface monitoring (Strazdins et al., 2011) and improving the precision of weather monitoring (Demirbas et al., 2010).

Some applications will always require new bespoke deployments, as existing data sources do not always provide appropriate data. However, the associated costs of new hardware deployments can be minimised by replacing or augmenting parts of the sensor network with existing data sources. Using social data can open up opportunities for new applications, as SNS cross cultures, providing a diverse set of potential data points about the users, their environments and experiences.

3.4 APPLICATIONS

Social data is being used to drive many new sensing applications. The use of social data in these applications can be categorised into three broad actions; i) widespread event detection, ii) individual event detection, and iii) contextual augmentation.

3.4.1 *Widespread Analytics*

Widespread event detection mass-monitors the social feeds of users for bursts in activity around certain topics, effectively treating users as nodes within a sensor network. Widespread analytics can be very useful for discovering mass-interest events, but also requires significant resource investment.

This process has been used to detect earthquakes and provide early warnings by monitoring Twitter users in Japan and detecting activity bursts that mention earthquakes (Sakaki et al., 2010). Algorithms used with sensor networks to approximate event location were also used to detect the earthquake's epicenter. This form of event detection is used to detect events with widespread effect by watching for a burst of relatively localised activity around a single topic.

3.4.2 *Individual Analytics*

Individual event detection is used to analyse a specific subject, usually a set of social accounts being monitored. Because the collection of data is far more focused and specific, far fewer resource are required to perform useful analysis. This eliminates any kind of wide-spread event detection, but can be useful for deeper analysis.

This technique has previously been used to detect the onset of depression in Twitter users by analysing their post content, usage patterns and relationships with other social media users (De Choudhury et al., 2013b). This type of application examines the node and its communications with other nodes to detect significant events related to a specific subject, rather than an event affecting multiple nodes.

3.4.3 *Sensor Augmentation*

Contextual augmentation uses social data to augment existing sensor networks by providing additional contextual information or assisting in missing-data estimation. The users effectively fill the gaps in sensor networks, requiring a smaller outlay of sensors to be deployed in order to create a functional application.

One application using this technique is WeatherSignal (OpenSignal, 2013), which uses a collaborative network of smartphone users to perform far more granular weather monitoring than traditional meteorological stations, which tend to be sparsely distributed across wide geographic areas. The accuracy and specialised results of the meteorological stations can be combined with the collaboratively-collected data to form a more holistic picture of weather patterns over the area, providing contextual augmentation of sensor data.

Most social analytics applications tend to be consistent but isolated in their function - meaning that any new application has to go through the same process of sourcing data, collecting it, integrating it (if required), processing it, filtering it and presenting it for use. While this processing flow remains the same for the majority of applications, there is no generic framework to support these applications. The development of a generic framework for integrating social data into sensor networking applications could greatly expand the interest in social sensing and drive the creation of new applications that are useful to society.

3.5 REQUIREMENTS

To use social data as a data source for sensor networking, the challenges previously stated need to be solved. To provide solutions to these challenges, a framework capable of finding and collecting from heterogeneous data sources, integrating the data and presenting appropriate output for use must be designed.

The solution to each of these challenges represents a process in the framework. A core requirement to solve each challenge is defined below. Any candidate solution must meet the following requirements:

1. As discussed in Section 2.6, collecting data from all available sources for use in sensor networks can be infeasible due to the sheer amount of data being produced (Ching et al., 2012). To query social media data in an efficient manner while retaining the ability to query as much relevant data as possible, some way of reducing the incoming flow of data to exclude irrelevant sources is needed.

   R1. Capability to locate application-relevant social data sources in an efficient and extensible manner.

2. To use social media for analytical applications, data needs to be retrieved from identified relevant sources. Retrieving social media data can be straightforward if an API is provided, or require web scraping if information desired by the intended application is not provided by the API (Russell, 2013). User-generated content should be collected from web pages automatically, without requiring manual wrapper development.

R2. Capability to automatically collect data from identified sources (both SNS and other media) in a generic and extensible manner, regardless of platform.

3. Social media data is noisy because it is almost entirely user-generated and does not adhere to a standard structure or format (see Section 2.4.2.1). This data requires extensive cleaning to remove spam and bring low-quality content up to a usable standard (Agichtein et al., 2008). Additionally, some data sources require follow-up processing to fill missing data. Without performing this cleaning, using the data in automated applications becomes significantly more difficult.

   R3. Support for extensible data cleaning and post-processing methods to ensure integrity of collected data.

4. To present the collected data in a format that can be integrated with sensor networking applications, there must be a way of comparing diverse datasets (Breslin et al., 2009a). A method is needed to mediate schematic differences between data sources, so that individual applications do not require manual mapping of data structures (see Section 2.4).

   R4. Support for the integration of heterogeneous data sources, such that applications are not required to manually handle data from different sources.

5. Processed data should be available both in real-time and historically. This supports the use of real-time event detection algorithms in addition to longitudinal analytical applications (see Section 3.6.6). It is expected that data available in real-time

may be missing attributes and won't have undergone post-processing, and that any filtering of the real-time datastream should be completed by the requesting application. Historical data should accessible through a query interface able to provide more complex search and aggregation functionality.

R5. Provision of unprocessed data in real-time and post-processed data historically, with advanced querying available to historical data.

6. Using social data in a range of different applications can require different methods of data presentation to the requesting application. Some sensor networking applications can integrate data in simple formats such as JSON, but others require more complex presentation techniques to use the data effectively.

   R6. Support for presentation of data in extensible formats for compatibility, including standard formats (XML/JSON) and more complex methods (AMQP, event-passing).

7. Enabling social data mining in an accessible manner for a broad range of applications requires a framework that can be run on a very diverse set of hardware platforms and configurations. This can range from datacenter-level platforms to ad-hoc networks of commodity-level mobile devices and home workstations.

   R7. Support for scaling to a wide range of hardware, from high-resource clusters to commodity and mobile hardware.

Each of these requirements represents a key area of functionality, respectively: Sourcing, Collection, Cleaning, Integration, Querying and Presentation. A framework designed to support the integration

of social data and sensor networking should fulfill all these requirements.

3.6 SMAAS FRAMEWORK

The following section proposes a framework designed to support the integration of social data and sensor networking, and aims to fulfill the requirements defined in Section 3.5, and meet the research aims described in Section 1.3. The architecture of the framework is described, with each major component and all connecting interfaces defined.

3.6.1 *Architecture*

The overall architecture of the proposed SMAAS framework is presented in Fig. 5. Of particular note is the divide between real-time and delayed processing portions, identifying the components which are expected to take additional time to execute.

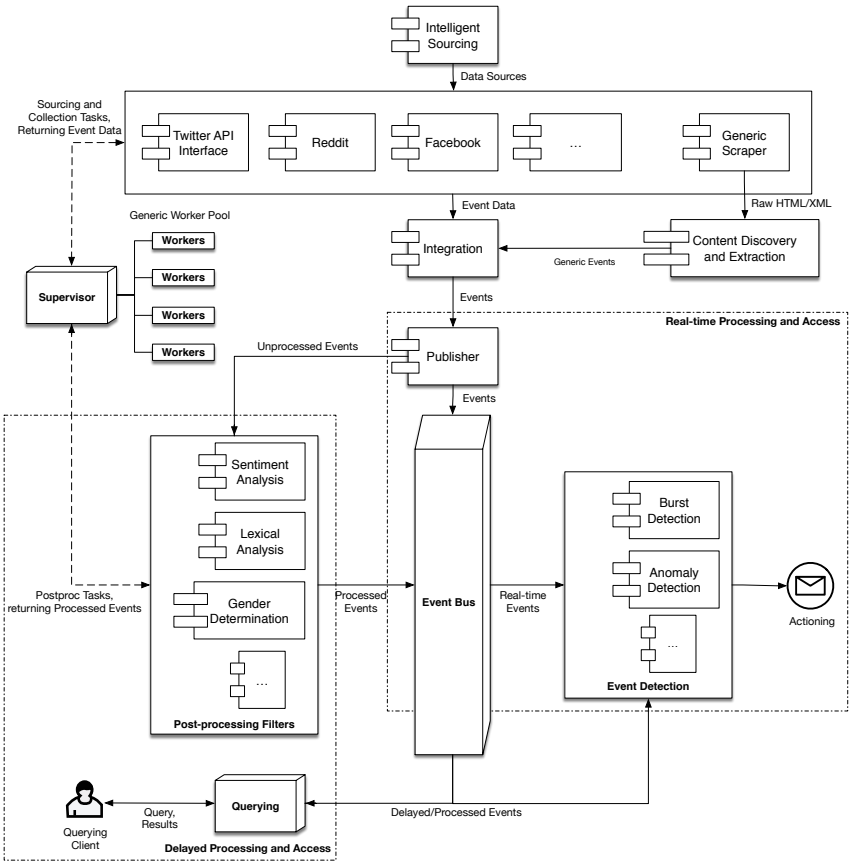


Figure 5: The proposed SMAAS framework architecture

The sourcing component is visible on the left of Fig. 5, and provides data sources in a generic format to the collection APIs. A number of platform-specific APIs are provided that form a proof of concept for a variety of different authentication and platform access mechanisms, in addition to the generic scraper capable of extracting social data from generic sources (described in Chapter 5). These APIs provide functionality used by the sourcing component (e.g. search functions), as well as data extraction functionality used by the collection and post-processing components.

Once data has been collected using the platform or generic APIs, it passes through integration. While identified as a separate logical component, the integration for each platform is included within the

collection API for organisational reasons. All data that exits integration conforms to a standardised schema.

Integrated data is immediately copied and published to two endpoints: the event bus, which is a message queue that holds data to be passed between framework components, and into a buffer for post-processing. Real-time event detection monitors the event bus for new events, and hence requires data to be made available as soon as possible. Multiple event detection filters can operate over the event bus and take immediate action if required.

Historical querying is not real-time, and can afford to take extra time to be run through post-processing for data enrichment before being stored for querying. Post-processing filters are also extensible, and often consist of sentiment analysis filters, gender determination and lexical analysis. Post-processing can also involve more complex operations, such as additional data collection to fill missing data. Data is only provided to a persistent storage and querying engine once all applicable post-processing has been completed.

Resource-intensive processes can take advantage of a generic worker pool capable of executing sourcing, collection and post-processing. Tasks can be prioritised to focus on collection and integration over post-processing as required.

The framework aims to support the development of sensing applications using social data, not to automate the entire process. For this reason, every component is entirely replaceable and extensible - from the sourcing algorithms to the collection/integration APIs, to the event bus and post-processing filters. Each component is effectively implementation-agnostic; there are preferred technology choices for each component, but can be swapped out to cater for existing solutions without issue, requiring

only the creation of an adapter between the component and the underlying event bus architecture. Each of these components is described more fully in the sections below.

3.6.2 *Sourcing*

One of the most effective optimisations in large-scale data processing applications is to filter data early, resulting in reduced processing for each successive step in the analytical process (White, 2009). As social data has an extremely broad scope, relevant data sources must be sourced to efficiently collect social data for use in sensor networking applications.

There are several approaches to this challenge. The simplest solution most often used in research is to perform simple keyword searches of SNS, dramatically reducing the scope of incoming social data (Sakaki et al., 2010; Robinson et al., 2013; De Choudhury et al., 2013b). This technique suffers from several drawbacks - while providing a set of the most relevant data, it is also prone to over-restriction and will automatically exclude data that is related but does not precisely contain given search terms. It is not able to adapt to shifting topics in real-time, and hence can miss important developments in on-going situations.

A new approach to this challenge, and the one implemented in this research, is intelligent sourcing. This technique treats SNS users as sensor networking nodes, allowing for a much higher-level approach to optimisation by searching for sources of data, rather than data itself. Intelligent sourcing finds relevant data sources by refining search queries using natural language processing and

machine learning techniques. This process is described more fully in Chapter 4.

Sourcing is an optional component that can be used to define a specific set of data sources to be used by the framework. These can be individual accounts on an SNS, or a set of pages across multiple websites, or any combination of data sources. It utilises the APIs described in the collection component to access search and collection functions in each supported source platform. Because the component only relies on the standardised platform APIs, any sourcing algorithm can be swapped in if different data collection methods are required. For example, a sourcing algorithm could be developed to only find and collect from real-time SNS firehose streams, rather than finding generic sources of real-time and historical data across the Internet.

3.6.3 *Data Collection*

There are three main approaches to data collection; i) the use of APIs, ii) access to formatted real-time streams, and iii) ad-hoc data collection.

Using APIs to collect data involves sending a specifically-constructed request to a provided server. This request contains a number of parameters used to focus the scope and range of data returned by the API. The requested data is then returned to the requester in a well-defined format. Some APIs provide a limited view of data available to the source, requiring the requester to follow-up with an ad-hoc request for more data. These initial API requests often lead to further queries for related data (such as

collecting user profile data for users that have posted in a comment thread).

Data can be collected from real-time social data streams by requesting stream access from a social network server. The source server then pushes a constant stream of real-time data towards the requester, in what is often called a "firehose" stream. This data is presented in a well-defined format that can be mapped to an appropriate data schema. Many firehose streams consist of all available real-time data being produced by the social network. The amount of data provided by this real-time stream can be problematic for systems operating with restricted bandwidth, processing or storage resources, and can easily overwhelm low-resource systems.

Ad-hoc scraping of data can be used for data sources that have no defined format and do not provide an API or formatted data stream (Pol et al., 2008). This usually requires the manual development of parsing algorithms tailored to specific sources that are able to discern useful information from unstructured content. Automated parsing algorithms can also be used. Ad-hoc scraping can also be used to enhance data provided by APIs, in instances where there is data missing or deliberately restricted from API access.

The SMAAS framework is designed to extensibly support all three methods of collection by defining platform APIs that present a unified interface for accessing SNS and other platforms, as presented in Section 3.6.4. Automated collection of social data from generic web pages is used to greatly broaden the scope of social data available for use, beyond what is currently available through the use of SNS APIs. This is described in greater detail in Chapter 5.

3.6.4 *Platform Interfaces*

Fig. 6 presents the interface classes forming the platform APIs. These provide the standard interface through which to access source platforms in the SMAAS framework, while still retaining flexibility for each platform implementation.

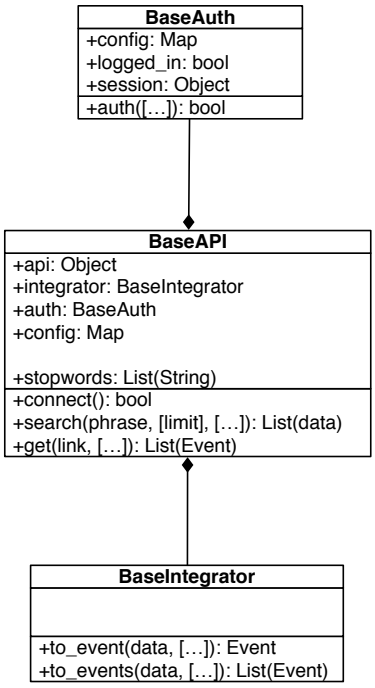


Figure 6: UML class diagram of SMAAS base platform API interfaces

The framework exposes the BaseAPI interface with requisite functionality to complete any sourcing, collection or platform-specific analytical or cleaning tasks. Additional functionality supported by all platforms can be added to the BaseAPI interface. Platform-specific implementations of the BaseAPI class can then be individually developed.

Source platforms can utilise many different types of authentication to restrict access - from simple HTTPAuth (Franks et al., 1999) to open standards such as OAuth1/2 (Hardt, 2012) or

even custom session handling. The authentication method for each platform can be developed by subclassing the BaseAuth class, with specifics handled entirely within the implementation. Common authentication schemes (such as OAuth) can inherit instead from the BaseOAuth1/2 classes. The authentication functionality in the BaseAuth class is flexible, taking any permutation of authentication parameters, and should return a custom API session that is then referenced by the implementation's BaseAPI specialisation.

The framework provides a set of standard authentication components used by many APIs, but custom components can be provided if required. Fig. 7 illustrates the simplicity of developing an auth module for a new interface, e.g. Twitter.

```
1  class Auth(BaseAuth):
2      def __init__(self, config=None):
3          super().__init__(config)
4
5      def auth(self, **kwargs):
6          session =
           ↪  twitter.Twitter(auth=twitter.OAuth(**self.config))
7          if not session:
8              log.error('twitter: login failed')
9              return None
10         self.session = session
11         return self.session
```

Figure 7: The Twitter interface Auth module in the SMAAS framework.

Each platform has very different access requirements - some may only accept API requests, others may not provide an API. Hence, the BaseAPI class is flexible - it presents a very simple interface to the framework, preferring to rely on platform convention over configuration. BaseAPI functions support flexible parameters, opting to pass through any parameters given and provide warnings for those unsupported by the platform that processes the call.

Some platforms may only support a subset of the base interface. One such example is the Google Custom Search Engine platform API, which provides full search functionality but passes off all collection responsibility to the generic HTML platform API. Similarly, the HTML platform API supports generic collection and integration, but does not support searching. Any requests for functionality that is unsupported is designed to warn but not fail, as it is expected that the majority of usage will involve passing a single call to a set of platform APIs, rather than individual platform API calls.

Most components of the SMAAS framework interact with the platform interfaces in order to source, collect and integrate social data.

3.6.5 *Integration and Data Design*

There are several ways to integrate the diverse range of schemas represented in the collected data, which are discussed in Chapter 2. For the SMAAS framework, integration is performed by using data mapping techniques to reconcile all data into a single, unified schema that has been designed in isolation of the available sources. As the framework only deals with social data, this task is simplified - all social data can fit into a certain design frame, with only occasional extraneous platform-specific elements excluded during integration.

The schemas defined in this section are only base schemas - they represent the minimum common set of data that each platform API must provide. This allows analytical applications to operate with the base schema and fully support every platform defined within

SMAAS, while also allowing further analysis per-platform to occur for specialist usage.

The SMAAS framework defines a global social Event schema as a JSON (Crockford, 2006) document, which allows it to be easily passed through the event bus and into the document storage and querying component. This also makes the design very easy to work with and extend - without a strictly enforced schema, enrichment and post-processing filters can append additional structures, ensuring extensibility. Ultimately, the task of ensuring schema validation falls to the interface developer in ensuring that the platform interfaces map data appropriately.

```
{"Headers": {
      "locale": (String: IETF Language Tag),
      "id": (String: Platform-assigned ID),
      "page": {
          "id": (String: Platform-assigned ID),
          "uri": (String: URI of page),
          "site": {
              "id": (String: Platform-assigned ID),
              "base_url": (String: Base URL of site)
          },
      },
      "interface": (String: Name of platform interface
       ↪ used)
}}
```

Figure 8: JSON Schema for Headers structure

Fig. 8 presents the JSON schema designed to represent the headers for a social event. As with many attributes, if the communication does not suit the structure, it can be left null. The headers are primarily of assistance to post-processing, which may require further direct access to the event. As an example, the headers may include the Twitter ID of a Tweet, allowing for further collection and analysis to take place on the raw data.

```
{"Node": {
    "id": (String: Platform-assigned ID),
    "groups": [ (List of Strings: Associated group names)
    ↪ ],
    "person": {
        "age": (Int or Intspan: Age or range),
        "gender": (String: Male, Female, Other),
        "family_name": (String: ),
        "given_name": (String: Any given names)
    },
    "identifier": (String: Account identifier / name /
    ↪ nickname),
    "interface": (String: Name of platform interface
    ↪ used)
}}
```

Figure 9: JSON Schema for Node structure

Fig. 9 presents the JSON schema designed to represent node details. This structure gets re-used to represent both the origin and destination of a communication (where applicable), and handles information about both the social media account used to send/receive events, but also the individual behind said accounts. Much of the data within the Node structure is designed to be filled during post-processing, for example using gender and age analytics.

```
{"Time":{
    "updated": (DateTime: Date and Time event was updated),
    "detected": (DateTime: Date and Time event was first seen),
    "created": (DateTime: Date and Time event was published)
}}

{"Location": {
    "place": {
        "type": (String: Descriptor of type, e.g. city, poi, country, landmark),
        "name": (String: Name of place),
        "country": (String: Containing country of place)
    },
    "geo": {
        "longitude": (Float: ),
        "latitude": (Float: )
    }
}}
```

Figure 10: JSON Schema for Time and Location structures

Fig. 10 presents the JSON schema designed to represent time and geolocation details. Time is a very common datapoint for virtually all social events, while geolocation tends to be somewhat more rare. Both are pivotal for many kinds of social analytics that rely on time or distance to correlate events. Both can be filled during initial collection, but geolocation can also be added during post-processing for those events that did not provide it with the initial communication.

```
{"Payload": {
    "description": (String: Descriptor of content,
     ↪ subject, heading),
    "extra": (Data: Additional attached data e.g. images,
     ↪ extra text, tags),
    "content": (Text: Social event content)
}}

{"Keywords": [ (List of Strings: Common keywords in event)
 ↪ ]}
{"Tags": [ (List of Strings: Platform-specific tags
 ↪ associated, e.g. hashtags, categories) ]}

(Private:)
{"Refs": [ (List of Strings: Platform-specific, used for
 ↪ cleaning ) ]}
```

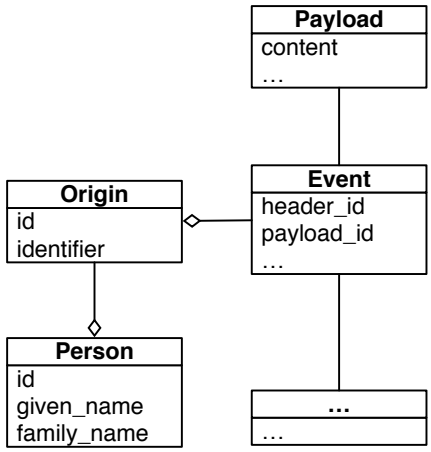
Figure 11: JSON Schema for Payload, Keywords, Tags and Refs structures

Fig. 11 presents the JSON schema designed to represent both the payload of the social event, plus additional metadata. The payload itself can store text or data, depending on the type of media accompanying the event. Keywords can be detected and populated during post-processing or initial collection, depending on performance requirements. Tags are often provided directly by the platform, but can also be generated during post-processing. Refs is a private, platform-specific datapoint that can be used during the cleaning phases, if required.

As JSON is being used to define a loose schema, additional data structures can be added by developers using the framework to allow for result storage of additional enrichment filters and post-processing functions.

```
{
    "id": (Unique ID associated to this event by
     ↪  SMAAS),

    "headers": (Headers),

    "origin": (Node),
    "destination": (Node),

    "time": (Time),
    "location": (Location),

    "payload": (Payload),

    "keywords": (Keywords),
    "tags": (Tags),

    "refs": (Refs)
  }
```

(a) JSON Schema for Event structure, composed of previously defined structures



(b) RDBMS Representation of an Event

Figure 12: Schema for presenting SMAAS Event data, JSON and RDBMS.

The JSON schema for the main Event object is presented in Fig. 12. This combines the previously defined structures into a single structure that can be passed through the event bus and used by event detection, or through post-processing and queried through the querying component. The loose schema used allows for customisation while providing solid cross-platform integration of social data, even in instances where there is no defined data schema (such as when collecting directly from web pages). Where required, additional structures can be added for events undergoing further post-processing and analytics, such as the sentiment analysis structure presented in Fig. 13.

```
{"Sentiment": {
  "polarity": (Float: polarity score),
  "objectivity": (Float: objectivity score)
}}
```

Figure 13: JSON Schema for additional sentiment analysis post-processing enrichment filter.

3.6.5.1 *Data Cleaning*

Data collected from social media and other Internet sources is noisy (Kietzmann et al., 2011), and can require extensive cleaning and processing. Most social data is user generated and adheres to no particular structure, primarily being unstructured conversational language. Differences in data formats between sources can also be problematic, such as the use of different date standards between cultures and platforms. Conceptual differences between individual data elements across different networks can also require manipulation of data into a unified standard.

There can also be structural problems with some social sources depending on the age and nature of data. Legacy data has often

undergone repeated format and schema shifts, so data collected from these sources can require extensive cleaning. Modern data sources adhere to stricter data standards and usually require less cleaning.

The method of collection can determine the extent of data cleaning necessary. API-provided results are usually retrieved from the database and output without presentation, and therefore adhere to internal database quality standards. Data returned from ad-hoc scrapers can require extensive cleaning and parsing. This can involve cleaning of content for undesirable elements, such as HTML tags and extraneous Unicode characters.

Data can also be cleaned for privacy reasons, for example if individual users are to be de-identified. Anonymisation of users can be performed during the cleaning process to ensure user privacy for sensor networking applications that do not require identifying information to be stored or processed.

The SMAAS framework requires data to be cleaned during the integration process, with platform interfaces defining the functions required to translate and convert data as necessary to move from the format provided to the schemas defined in Section 3.6.5.

Textual content is retained in a raw format, to ensure that any analytical applications get access to the original dataset. As such, text does not go through any text processing (such as stemming or lemmatising) during integration - though this is often added during post-processing to perform more complex text analysis.

3.6.6 *Event Handling: Real-time vs Delayed*

Once social events undergo integration and cleaning, they exist as a set of data in memory conforming to the schema defined in Section 3.6.5. Once in memory, there are two possible uses for this data; i) immediate use in event-detection filters, and ii) to be later stored and queried after undergoing post-processing. Events can be duplicated and sent through both processes, as illustrated in Fig. 14.

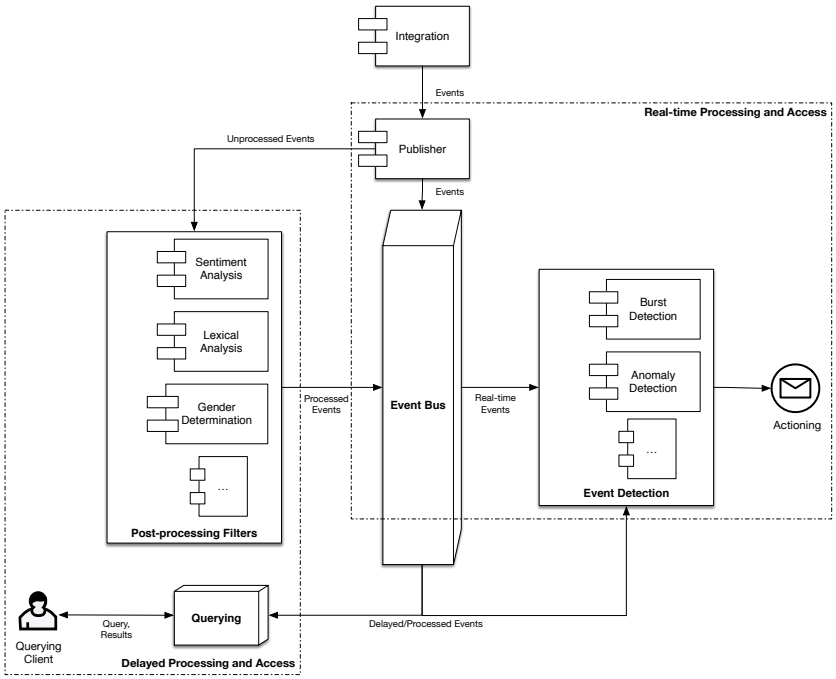


Figure 14: Event handling for real-time and delayed processing. Post-processing filters are designed to operate on the delayed queue in the event bus as they take additional time, allowing applications that require low-latency access to data to instead use the real-time queue. The real-time queue then passes events through as soon as they are integrated, requiring far less time to process than the fully post-processed events.

Event detection applications are often extremely sensitive to time latency, as they are designed to detect rapidly-occurring events (McCreadie et al., 2013). In order to reduce latency to these applications, data intended for real-time use can be pushed straight

into the event bus in its base format. Event detection filters can then take them from the bus, analyse available content and metadata and perform any required actioning as soon as possible, without having to wait for events to run through post-processing. This provides social events to real-time filters as soon as they are collected and integrated, but the events may be missing additional data. Advanced querying is also unavailable for real-time events, and may be implemented by the filters if required.

Delayed applications include analytical applications that do not require real-time actioning but do require additional data enrichment and advanced querying. Data can be collected and integrated as normal, but is then immediately pushed into a post-processing queue rather than the primary event bus. Data enrichment and filling filters can then modify the events to add additional data, as specified by the platform interface that performed the collection. After events undergo post-processing, they can be added to the event bus.

All events that pass through the event bus and the appropriate filters should then be serialised by the storage component. This then allows the querying component to accept advanced querying requests from clients over the full set of completed data. All events, whether real-time or post-processed, should then be sent for storage - a fully post-processed event should replace a minimal event in persistent storage, once available.

Post-processing operations can be developed as a post-processing filter within the SMAAS framework, which allows for work to be completed in a delayed fashion after collection and integration have been completed. The filters themselves can be simple modules that operate on worker machines. Fig. 15 illustrates the intended flow of

event data along the post-processing pipeline, and demonstrates a simple sentiment analysis enrichment filter used in the SMAAS framework that uses the vaderSentiment (Gilbert, 2014) library to perform entity-based sentiment analysis in social content. The module should take an event that has yet to undergo sentiment enrichment as input, perform the appropriate enrichment, and return the enriched event.

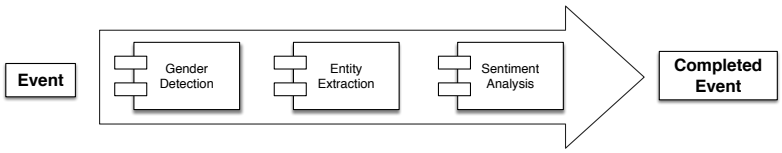


Figure 15: The post-processing flow, with an example sentiment analysis filter shown.

3.6.7 *Querying*

Querying is required for analytical applications to request filtered or aggregated views of the social data collected. Data querying is also a well-researched and supported field (Florescu et al., 1998; Hartig and Pérez, 2016; Bartman et al., 2017), with databases and querying forming an integral part of virtually every information system in use.

Traditional Relational Database Management Systems (RDBMS) (such as PostgreSQL, MySQL and Oracle) are designed to store relational data, or data of different schemas that have well-formed and defined relationships. While social data is able to fit this mould, the combined real-time/delayed nature of the SMAAS framework makes the use of relational models more difficult, as continuous read/write access of the database would be required to maintain referential integrity.

As the social data in the SMAAS framework is stored adhering to JSON schemas, document storage engines (Han et al., 2011) can be used instead of RDBMS. These storage and querying engines are designed to store data in more free-form standards, such as JSON/BSON or key-value documents. Any of these engines could be used as the querying component in the SMAAS framework, but ElasticSearch (Kuc and Rogozinski, 2013) was selected due to its highly scalable nature and advanced querying/aggregation support.

3.6.8 *Scalable Processing*

In order to scale to application requirements and incoming data streams, the collection, integration and post-processing components must be distributable. While post-processing is a delayed component, collection and integration must be extremely responsive to incoming data.

The SMAAS framework manages this process by using a distributed worker pool. A supervisor manages task allocation by priority, automatically prioritising real-time tasks over delayed tasks. In the event that workers are unable to keep up with incoming tasks, additional workers can be seamlessly and automatically spawned to handle the additional workload and despawned during periods of inactivity.

Workers are categorised by their resource strengths. Those with higher bandwidth and reduced memory and processing resources are used for collection tasks. Workers with greater available processing and memory resources are used for generic collection and integration, a processing-heavy task. Well-rounded workers are used for post-processing tasks, which tend to require a mix of

network access and processing time. Tasks sent by each different component (collection, integration and post-processing) are then forwarded to the appropriate workers and results can be published into the event bus or on to post-processing, as suitable.

3.6.9 *Command Line Interface*

```
SMAAS Collectortron 2000.

Usage:
    collector.py get [--stream] <links>
    collector.py search [--stream] <phrase>
    collector.py stream [--interface=<name>]
    collector.py intelligent <phrase>

Arguments:
    <links>             Comma-separated list of links to
     ↪  collect data from.
    <phrase>            Search query to use.

Options:
    -h --help           Show this screen.
    --stream            Continue to stream results from
     ↪  source.
    --interface=<name>  Interface to stream from.
```

Figure 16: The Command Line Interface to the SMAAS collection module.

The collection functionality of the framework should be accessible in two ways; through use of the framework API, and a Command Line Interface (CLI). A proposed CLI is presented in Fig. 16, with all collection functionality available.

3.7 CONCLUSION

This chapter defined a set of requirements for efficiently sourcing, generically collecting, integrating, cleaning and serialising social data from disparate social data sources. It then proposed the SMAAS framework, an architecture designed to fulfill these requirements by providing extensible solutions to each challenge encountered.

Efficient sourcing of social data is one of the most important optimisations that can occur, as it reduces the requisite work in each successive phase of the processing pipeline. The next chapter details an algorithm designed to intelligently source social data by iteratively expanding the scope of search queries using machine learning and natural language processing techniques.

4

INTELLIGENT SOCIAL DATA SOURCING

4.1 OVERVIEW

One of the major challenges with collecting social data for analysis is the sheer volume of user-generated content being created. Facebook alone collects and warehouses almost half a petabyte of data per day (Ching et al., 2012), and there is substantial social data being created outside social networks. The total throughput of social data is overwhelming, with very few services able to supply integrated social data (Gnip, 2008; DataSift, 2010), and only by using very substantial processing hardware and a lengthy ETL process with which to process and store the data. These services take the "collect-all" approach to social data, wherein as much data as possible is collected and stored for later use by subscribers, who can pose searches and queries against the dataset. Even for systems of this scale, complete collection is infeasible - DataSift archives 2TB of data per day across all sources (DataSift, 2010), approximately 0.4% of Facebook's daily total or 6.5x Twitter's daily total[1].

The alternative collection method is referred to as the "collect-little" approach, in which as little data is collected as possible to draw a conclusion. In existing applications, this involves a simple keyword search that finds a very narrow scope of relevant data, often missing relevant data not containing the appropriate keywords. Studies that

1 Assuming 140 UTF-8 chars per tweet (560B) plus half again in metadata (840B total) at 277,000 tweets per minute (DOMO and Column Five Media, 2012) totalling 312GB per day.

use this approach often have a very narrow application scope, and are used with topics that have unambiguous wording (Sakaki et al., 2010; Robinson et al., 2013).

Developing a system able to collect and query all UGC across the Internet in real-time is likely infeasible, as it would require a system capable of scaling to the size of all sources of social data combined. Hence, a more intelligent and efficient method of collecting data is required. An intelligent method of sourcing social data would involve efficiently locating social data semi-relevant to the specified query, while providing significantly lower noise than collecting all available data.

Most analytical applications are developed for a specific purpose, such as determining user opinions related to a product, or detecting earthquakes. Each of these purposes is distinct and does not require the collection of all UGC to make a determination. Hence, data returned from the collect-all approach would require significant filtering to retrieve data relevant to the topic. Rather than filtering after collection (and requiring all UGC to be integrated into the system), filtering early can be an effective method of only retrieving the data that has a reasonable chance of being used.

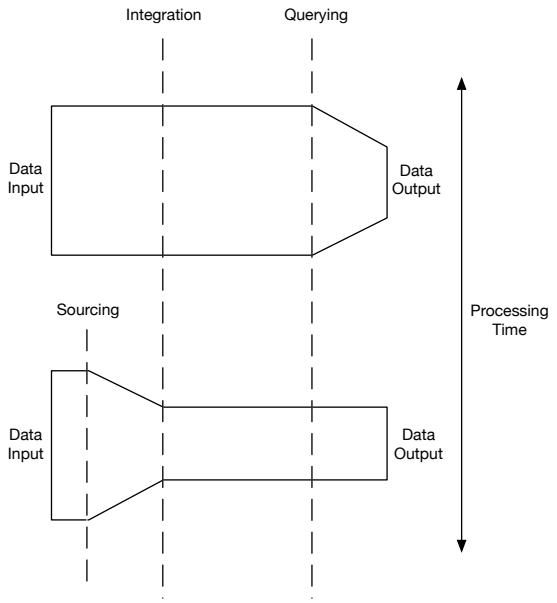


Figure 17: Data processing reduction using early filtering. Resource consumption is reduced in each successive stage after filtering, so filtering early reduces the overall resource load required.

Social data collection can be optimised by going back to core principles and conflating the concepts of SNS and sensor networks. Users tend to have persistent interests, and discussing a topic on social media suggests that they will discuss it again in the future. Hence, the relevance of a source can be identified rather than individual events, saving considerable processing time. This source list can then be directly collected from in addition to expanded keyword searches, providing more relevant data. The collect-all and intelligent collection processes are illustrated in Fig. 17, which highlights the reduction in requisite processing time when early-filtering is used. As shown, resource consumption is reduced in each successive stage after filtering, so filtering early reduces the overall resource load required.

There are some disadvantages to this approach over the collect-all approach. There is a chance that an amount of relevant data will be missed due to misclassified sources. The collected data is only

relevant to the query, and subsequent unrelated applications would have to perform their own data collection rather than repurposing the collected set. If using queries that collected a broader set of data, the collected set can be used for multiple applications.

4.2 BACKGROUND

There are three methods currently used to find social data from SNS for analytical applications. The first is simple keyword searching, provided by most SNS APIs (Guy et al., 2013). The second is parsing firehose streams (Zhao, 2013), which provide a fast outlet of all produced content in real-time. The third is to collect data from user profiles and lists, which provides all messages written by users (De Choudhury et al., 2013b). Each of these three methods produces varying amounts of data, with firehose streams typically providing the most data (but least relevant), and keyword searches providing the least data (but mostly relevant). As analytical applications tend to focus on a very specific topic scope and desire relevant input, keyword searches are primarily used (Sakaki et al., 2010; Robinson et al., 2013; Wandhöfer et al., 2012).

Most applications perform a simple search against the SNS API, using a list of pre-specified keywords (Sakaki et al., 2010). To detect posts relevant to earthquakes, the words "quake", "shake", "earthquake" etc. may be used. These searches are quite specific, and the collected data is unlikely to be of use to many other applications. Nodes can be extracted from these lists to identify relevant data sources.

Other applications designed for more broad event detection use firehose streams as an input for burst analysis algorithms (Cameron

et al., 2012; Thapen et al., 2015). These algorithms look for rapidly-growing events around a similar set of topics that may indicate an important event. Because there is no specific topic to focus on, filtering cannot occur and the algorithms have to directly analyse all incoming data. By performing simple checking of relevancy on incoming messages, this data source can be used to identify nodes posting relevant data (which can then be further examined per-user).

Direct access of user streams can be used for analytical applications designed to perform per-user analysis (De Choudhury et al., 2013b). These algorithms retrieve a subsection of the user's messages and classify them based on factors such as time or sentiment. While this only retrieves a single user's messages, it can be used on a broader scale to collect data from multiple users - if the users are pre-identified as providing data relevant to the desired topic.

The majority of social analytical studies tend to source data from a single SNS (Breslin et al., 2009b), as cross-integration of networks can be a significant obstacle (Driscoll and Thorson, 2015). By only sourcing from a single network, the applications used to collect and analyse data are greatly simplified, at the expense of limiting data scope and demographics. The research presented in this thesis aims to propose a generic framework for generic social data analysis, so generic data sources and cross-platform integration must be considered. To this end, any techniques developed must be platform-agnostic.

4.3 OPTIMISATION OF SOCIAL DATA MINING

One of the simplest optimisation steps in large-scale data processing applications is to filter incoming data, resulting in reduced processing for each successive step in the integration process. As social media has an extremely broad scope, only relevant data sources need to be located in order to efficiently leverage social data in sensor networking applications. Much of the social data processed may be irrelevant to the application. As this data must undergo cleaning, integration, storage and querying, early filtering can result in significant resource savings. Hence, the process of finding quality data sources is very important.

Most applications that perform early filtering of social data do so by using keyword searches. The SNS API is given a short list of keywords that are used to locate relevant material, excluding material not containing the keywords. These applications (Sakaki et al., 2010; Robinson et al., 2013) use a set of predetermined words related to the topic that are used to search for data. While this approach usually returns appropriate results, conversations about ongoing events often evolve new language or metatags, and any further information using these terms would not be collected. Hence, a better method of adapting queries to evolving situations is required in order to retrieve as much relevant data as possible. Applying machine learning techniques to retrieved content would allow for iterative adaptation of search queries, ensuring that the application is kept abreast of new developments in ongoing situations.

Collecting and classifying UGC and SNS content is resource-intensive work primarily due to the sheer amount of data

involved (DataSift, 2011). To further improve the social data collection process, higher-level collection optimisation can be performed by isolating and evaluating data sources, rather than individual pieces of content. By conflating the concepts of Sensor Networking and Social Networking, SNS users are treated as as sensor nodes that emit events (messages, statuses) and have connections (friends, followers).

Most existing systems determine the relevancy of one element: events, or social messages. By instead considering the relevancy of nodes, the amount of work required to identify relevant data is dramatically reduced. Nodes in a sensor network are capable of collecting and broadcasting many types of data, much like users on a social network will often talk about similar topics repeatedly. Only a fraction of the events emitted by a node need to be analysed to determine relevancy, rather than determining relevancy for every single event emitted by users. While this allows some irrelevant data to pass through the filter, it narrows the scope of collection considerably while also broadening collection beyond a simple keyword search. This approach provides a welcome middle-ground to the collect-all and collect-little techniques.

Relevancy ranking for nodes is based on the presence of keywords within content emitted by the nodes. This ties in with the iterative querying approach: as the algorithm searches with an expanded keyword set, additional nodes are identified based on their use of adapted key terms that have been discovered. Nodes can then be filtered more strictly as required by ranking the appearance of keywords within content, such as through the use of a TF-IDF[2] algorithm (Ramos, 2003).

2 Term-Frequency - Inverse Document Frequency is a measure of how important a keyword is to a document by examining how many times it appears in a document compared to its average appearance throughout all documents.

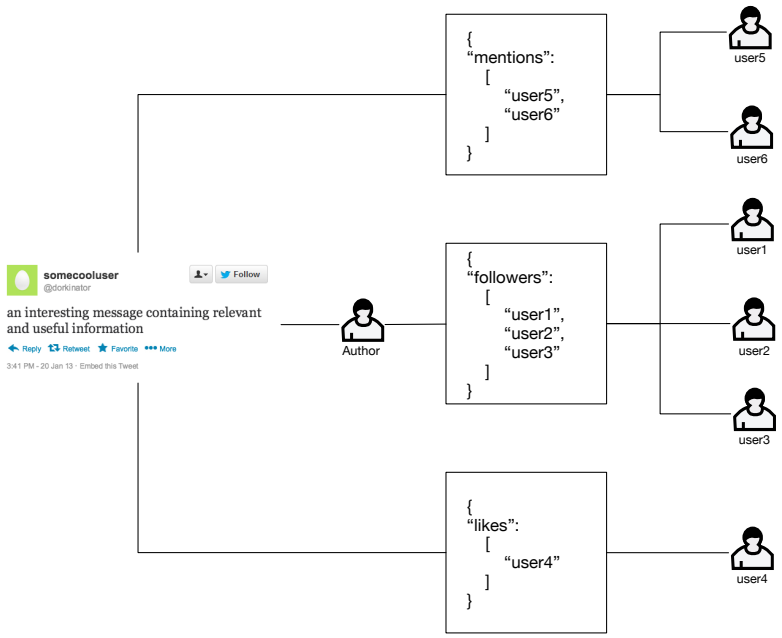


Figure 18: Applying WSN principles to SNS to find new nodes over connections. The boxes represent an example JSON structure exposed by an SNS API, and highlight the number of user connections that can be derived from a single message.

Once relevant nodes have been located (and hence, relevant events), the scope of collection can be broadened if required. Fig. 18 illustrates three different ways that additional nodes can be discovered from existing classified nodes. Connections (followers) of existing nodes can be examined for relevancy and added to the source list, if required. Nodes that have expressed an interest (by "liking" events) are likely to share similar interests to the relevant node, and can be examined. Nodes that have directly replied to relevant events are also candidates, and are likely to produce relevant information themselves. These nodes can all be evaluated for relevancy and added to the source list.

4.3.1 *Platform Agnosticism*

As one of the driving motivations behind this project is to simplify cross-platform social data mining, the intelligent sourcing process must be platform agnostic. This allows data to be collected from a much broader demographic than a single SNS, including websites and other site not typically used for social analytics.

To simplify cross-platform sourcing, the SMAAS abstraction framework detailed in Chapter 3 can be used. The framework provides a set of standard API endpoints that can be applied across all platforms, providing generic access to search, collection and metadata functions. By using these standard elements (rather than specific APIs), any future platforms can be integrated into the sourcing process simply by building a new interface in the overall SMAAS framework. Listing 19 presents a simple search in the SMAAS framework operating over multiple platforms.

```
1    results = []
2    for api in available_apis:
3      results += api.search('python', limit=50)
```

Figure 19: SMAAS code to execute a simple keyword search across available APIs

In addition to platform agnosticism, sourcing itself can be cross-platform. Social data on one platform often links to a second platform, such as a Tweet linking to a website or a Facebook message linking to a Tweet. SMAAS provides the ability to traverse between platforms at-will, providing much richer and more plentiful data.

Users often have multiple accounts on different platforms. These accounts can be linked together, forming a holistic node from which to source data. This can further expand the links between nodes

across platforms, improving metadata surrounding nodes. For example, a Twitter user may not provide a location in their profile, but their Facebook account might contain this information. Linking the accounts and filling the missing metadata can assist greatly with further querying.

4.4 INTELLIGENT SOURCING

Intelligently and efficiently sourcing social data is a highly iterative process. All collected content is used to further optimise the search queries used to locate data sources, and the sources themselves are used to locate additional relevant sources. This process is broken into distinct steps:

1. Initial Search - use a set of seed keywords to initiate searching across any available SNS or web Search APIs.
2. Extract Metatags - extract any platform-specific tags within the message for further use, e.g. Twitter hashtags, email addresses, HTTP URLs.
3. Analyse and Process - use natural language processing to categorise words and extract nouns for use in query optimisation.
4. Build Node List - add discovered source nodes to our list.
5. Follow Connections - follow connections out from nodes on our list to discover other relevant nodes.
6. Iterate to Initial Search - using the collected metatags, the new common keywords and our set of nodes, improve the search query and re-iterate the process.

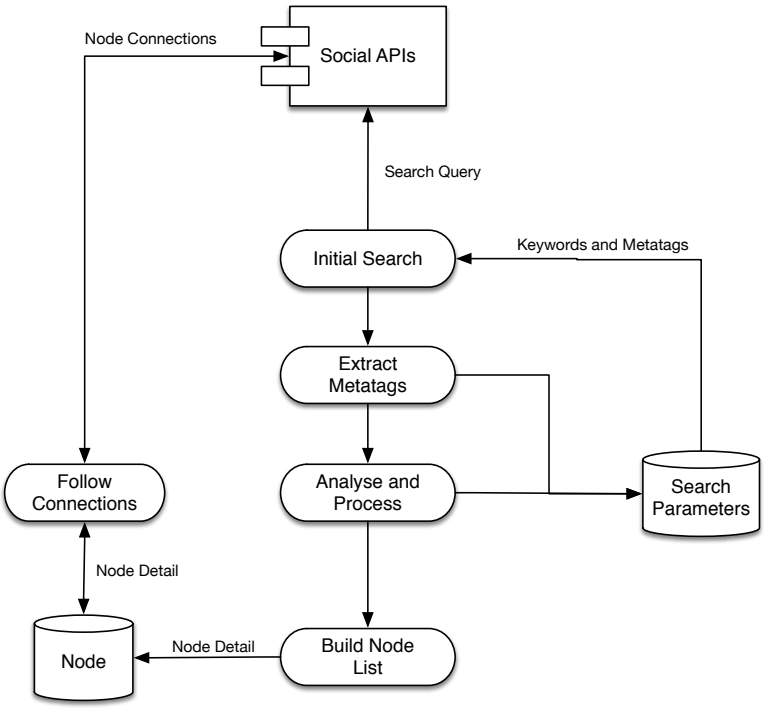


Figure 20: Finding relevant sources using an iterative learning search process

Fig. 20 illustrates this sourcing process. The query specifies relevant keywords, which can be expanded upon by use of predefined databases and appended to by examining oft-used keywords on strongly-relevant search results. These keywords are used to query known search APIs, returning a list of locations that potentially contain results. The exact method used for data access and searching can vary for each system, as each platform provides differing methods of accessing and filtering data. Some examples of different methods are:

FACEBOOK Using the API, search for public posts with related search terms, popular news feeds and other items of interest. Additionally collect comment authors.

TWITTER Using the API, search for public tweets containing related keywords. Additionally collect information about replies to

tweets, and also examine hashtags commonly appearing within the initial set.

BLOGS Using Google's Search API, search for public blog posts containing relevant keywords, including author information and comments. Additionally collect relevant results from commenters' own blogs.

FORUMS Using manual page scraping and authentication for forum software such as VBulletin and phpBB, search for forums containing threads relevant to our query.

This initial phase provides a list of relevant places to look for information relating to a single query. Not all data found is guaranteed to be useful, and some may be completely irrelevant. By selecting a fairly wide sub-set of available data, initial collection is limited to a feasible scope, allowing for more accurate filtering later. This optimisation can potentially exclude obscure results, but this is a necessary trade-off. The only data required to start this process is a set of APIs to be used and an initial seed keyword, and the result of this sourcing process is a list of relevant data-producing nodes. The process can be repeated infinitely to locate additional source nodes, until either the data stream is tapped out or the minimum acceptable signal-to-noise ratio (SNR) is reached.

Each step in this process is described more fully in the sections below.

4.4.1 *Initial Searching*

The initial searching process is virtually identical to existing social data collection methods used in previous research (Cameron et al.,

2012; De Choudhury et al., 2013b; Robinson et al., 2013). An SNS API is given a basic keyword or keyword set, and returns a set of results. In most research using social data, this is the limit of collection done - the results retrieved make up the complete dataset used. Most analytical applications only take from a single API, due to integration difficulty (Driscoll and Thorson, 2015). Listing 19 shows a simple keyword search using SMAAS' abstraction framework. This retrieves 50 results relevant to the given keyword from each API (e.g. Twitter, Facebook, Reddit, Google Search) in a standard format.

Using simple keyword searches are fast and easy, but suffer from problems. Poor choice of keyword can exclude a significant amount of useful data. The nature of conversation on the Internet often results in dramatic shifts in language within evolving real-time situations, meaning that an effective keyword now may not remain so as an event unfolds. Users often define metatags to denote messages relevant to current or ongoing events, such as hashtags on Twitter, which would require a new search to collect. Additionally, new topics can come up that are relevant to our interest, but do not contain the keyword specified. Without manual perusal of conversations (a nigh-insurmountable task given the amount of content), most of these topics would be missed.

To improve on this, it makes sense to have a search algorithm capable of evolving to stay current with situations and adapt to suit user language. Rather than operate on a single keyword or set, our method should begin with this to get a "feel" for the area, but learn from any content discovered in order to cast a wide but relevant net. Common keywords discovered within collected content can then be integrated into the keyword set to expand the search, in an iterative

learning process. Before this analysis can take place, special tags embedded within content need to be extracted so as not to interfere with the analysis - a process described in the next section.

4.4.2 *Metatag Extraction*

Metatags often appear within social data to provide overall context for the message. SNS such as Twitter use metatags to denote who the message is to and which topics are under discussion. Web URLs also appear often in social content, as users link to other content. This content can be very useful, as it provides additional links to new nodes, keywords and cross-platform content.

These tags can be extracted from content easily, as illustrated in Fig. 21. Regular expressions can be used to extract metatags on a per-platform basis, with each platform interface designed to extract appropriate metatags from its content. The SMAAS interface for that platform can then be used to follow-up on each metatag, sourcing further content.

```
>>> content = '''
... .@TonyAbbottMHR branded 'Captain Chaos' over his support for a plan to set up another Australian medical school. http://ow.ly/N2BRg
...
  '''

>>> mentions = regex.findall('(@\w+)', content)
>>> mentions
['@TonyAbbottMHR']

>>> hashtags = regex.findall('(#\w+)', content)
>>> hashtags
['#RohingyaCrisis', '#refugees']

>>> urls = url_regex.findall(content)
>>> urls
['http://ow.ly/N2BRg', 'http://bit.ly/1JS0Zru', 'http://ow.ly/N1rC6']
```

Figure 21: Python code to extract metatags from Twitter content

Some APIs used to access social data already provide the metatags as an extracted field, generally simplifying the process. In some instances (e.g. rich HTML content) metatag extraction can take a substantial amount of time.

These tags generally have to be extracted from content prior to content analysis, else they can confuse any NLP-based analysis. Once they have been extracted, saved and removed from the original content, the content can undergo NLP.

4.4.3 *Analysing Content*

Analysing collected content can lead us to additional relevant nodes. These nodes are often mentioned directly in the messages collected, but natural language processing (NLP) can also be used to extract keywords from content to improve searches.

```
>>> content = '''
... .@TonyAbbottMHR branded 'Captain Chaos' over his support for a plan to set up another Australian medical school. http://ow.ly/N2BRg
...
'''

>>> words = nltk.pos_tag(nltk.word_tokenize(content))
>>> words
[('.', '.'), ('@', '``'), ('TonyAbbottMHR', 'NNP'), ('branded', 'VBD'), ("'Captain", 'JJ'), ('Chaos', 'NNP'), ("'", "''"), ('over', 'IN'), ('his', 'PRP$'),
  ...
]
...
>>> words = [(word, tag) for word, tag in words if 'NN' in tag]
>>> words
[('TonyAbbottMHR', 'NNP'), ('Chaos', 'NNP'), ('support', 'NN'), ('plan', 'NN'), ('school', 'NN'), ('East', 'NNP'), ('West', 'NNP'), ('Link', 'NNP'),
  ...
]
...
>>> cn = collections.Counter([word for word, tag in words])
>>> cn
Counter({'politics': 3, 'thought': 1, 'support': 1, 'school': 1, 'Link': 1, 'RohingyaCrisis': 1, 'drug': 1, 'refugees': 1, 'sea': 1, 'accompanies': 1, 'lives': 1, 'cares': 1, 'Crime': 1, 'Chaos': 1, 'seizures': 1, 'plan': 1,
  ...
})
...
```

Figure 22: Python code to part-of-speech tag and count frequency of nouns

Fig. 22 demonstrates the process behind identifying keywords. Collected content can be split by token to isolate words and then put through a part-of-speech tagger, which can determine the grammatical type of each word. Words that are tagged as nouns (or pronouns, adnouns, etc) are kept. A counter can be used to identify high-frequency words for use in the search algorithm. This word list

is continually updated as new content is discovered, so that the most-relevant keywords are used.

The process of extracting nouns from the text allows for expansion and contraction of the search space as the topic changes in real-time. People refer to topics by new words, and new information becomes relevant over time. By using standard search methods that don't learn from results, the search queries often become outdated and can't adapt to shifting conditions. But by adapting to suit the language in use by authors writing about events, the search query constantly evolves to suit the topic. However, this can sometimes lead to unexpected consequences - if users are talking about two similar topics, the search query can quickly diverge from its original focus.

This technique is more effective on longer content such as web pages, but can be equally effective on shorter content when multiple pieces of content are combined. Common keywords can be discovered by evaluating the combined set. Individual content can then be classified for relevancy using the combined keyword set.

4.4.4 *Sourcing through Connections*

Social media can be used for event detection (Sakaki et al., 2010), which is particularly useful for sourcing data. We consider each user to be a node that emits events and has connections. Hence, related nodes can be discovered by following each node's connections. Fig. 23 presents part of a Tweet discovered through a simple search, showing the text content of the message and the user (node) details.

```
/* /statuses/show */
{
  "created_at": "Mon May 18 03:13:13 +0000 2015",
  "id": 600137035781836800,
  "text": "Journalists hit back: Can sexism or bias
  ↪ explain outcry over @leighsales 'standard
  ↪ political interviews'? http://t.co/0H4R8k2W0L
  ↪ #auspol",
  "user":  {
    "id": 70617159,
    "screen_name": "PoliticsFairfax"
  }
}
```

Figure 23: A tweet found through the Search API, returned in JSON

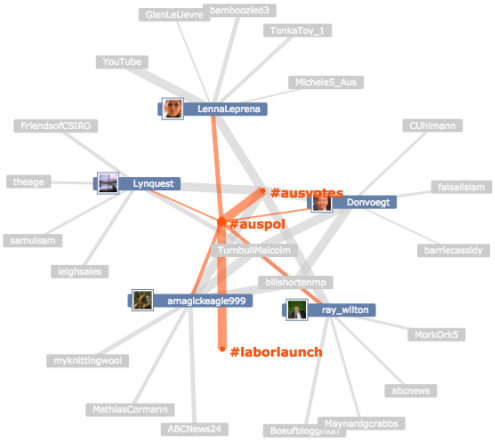


Figure 24: A social graph depicting a popular Twitter tag for Australian politics.

Messages can be "liked" or referenced by other users on most SNS, as shown in Fig. 24, which usually indicates their support of the message or a reply on the same topic. These connections can be followed to each event by examining users that have interacted with the message. Fig. 25 shows an example from a list of interactions with a message. Users that have interacted with the message are probably relevant, and their details can be stored for later evaluation of node relevancy.

```
/* /statuses/retweets */
[{
    "created_at": "Mon May 18 05:12:44 +0000 2015",
    "id": 600167111462260700,
    "text": "RT @PoliticsFairfax: Journalists hit back:
    ↪ Can sexism or bias explain outcry over
    ↪ @leighsales 'standard political interviews'?
    ↪ http://t.co...",
    "user":  {
      "id": 26469521,
      "id_str": "26469521",
      "name": "Rob Giblin",
      "screen_name": "Tassiebeef",
    }
  },
  ...
]
```

Figure 25: A list of users that have forwarded the tweet in Fig. 23.

Connections can be followed directly from nodes, finding other relevant users. Events only flow across SNS connections if the users make a conscious decision to agree to see the other user's messages. As such, a node connected to a relevant node is also possibly relevant. We can get a list of all nodes following another node, as seen in Listing 26. This list can then be worked through to find relevant nodes.

```
/* /followers */
{
  "ids":  [
    411898978,
    2704323866,
    293852007,
    618822342,
    3004531998,
    219272010,
    ...
  ]
}
```

Figure 26: A list of users following the original author of the tweet.

This connection-following process can spider out from relevant sources to any number of levels, increasing the number of data sources and broadening the source list of social data. This process can take a significant amount of time, due to the sheer amount of sources to evaluate and propagate from, but only a single run is required. After sourcing has been completed, data can be collected from identified sources and new sources can be incrementally identified from newly collected content.

4.4.5 *Cross-platform Connections*

One of the major advantages of using the intelligent sourcing algorithm through the SMAAS framework is the additional flexibility gained in cross-platform searches. Not only are the searches themselves able to be performed against multiple platform targets, but any cross-platform links (e.g. tweets containing Facebook post links) can be automatically evaluated and followed. This provides further enrichment to the search, allowing for a much broader search to occur - particularly when combined with the CFH-NS algorithm detailed in Section 5.4.2.

The end result of this sourcing process is a large social graph with many inter-platform links that can continuously provide useful and valuable data when continuously monitored, can include any new sources discovered over time, and can automatically cull those sources that become irrelevant. When combined with the SMAAS framework, this can result in a significant amount of relevant social data with relatively few processing resources required.

4.5 CONCLUSION

This chapter has discussed the development of an intelligent sourcing algorithm, designed to improve the efficiency of social data collection while ensuring that relevant data is collected. The algorithm uses the platform APIs described in the previous chapter to execute search queries. These search queries are iteratively built using machine learning (ML) and natural language processing (NLP) techniques to identify important entities within the discovered data, broadening the scope of each search while retaining relevancy.

The next step in the research is to develop a method of collecting the data from these discovered sources, without having prior knowledge of the format or design of the pages containing data. Chapter 5 describes an algorithm capable of doing this.

5

USER-GENERATED CONTENT EXTRACTION

5.1 OVERVIEW

While the majority of research studies using social data collect data from an API, there remains a significant amount of social data inaccessible to most research. This data, referred to as User-generated Content, consists of user comments, testimonials and content that lay in the comments sections of websites, reviews on eCommerce platforms and posts on web forums. This thesis aims to address this problem by increasing the availability of UGC for academic and commercial applications.

UGC is a promising data source for analytical applications, providing up-to-date information about a wide range of topics. There is increasing focus on the use of UGC on Social Networking Sites (SNS) to drive applications for emergency management (Sakaki et al., 2010; Robinson et al., 2013; Buscaldi and Hernández-Farias, 2015) and opinion mining (Maynard et al., 2012), amongst other uses. While previous research has extensively analysed data from micro-blogging SNS such as Twitter, only a few studies have used traditional SNS (Wandhöfer et al., 2012) or embedded UGC such as user comments and reviews (Cao et al., 2008). Since the shift to Web 2.0 paradigms, static content on websites has increasingly been replaced by UGC (O'reilly, 2007) that is not easily or generically

accessible, resulting in a large amount of UGC being unavailable for mining.

While this data is timely and relevant, its relative inaccessibility is a major obstruction to its use in analytics. There are few methods of retrieving UGC embedded in pages, and there are no unifying interfaces to perform data collection from such sites. This contrasts to UGC on SNS, which is usually accessible via APIs - interfaces specifically made for automated applications to query stored data. Developing a method of accessing UGC in a standardised and reliable way would allow for the use of this data in social analytics applications, and provide the same accessibility as API-provided data from SNS.

UGC presents an excellent opportunity for providing data about a wide range of topics, both historical and real-time. Users often post timely, relevant information on news articles about current events that can enhance or enable real-time event detection. Similarly, there is great value to be found in product-related UGC attached to product pages or reviews for marketing purposes (Tang et al., 2014; Tuten and Solomon, 2014). With no formal interface to collect this data, generic collection of UGC is difficult and a significant proportion of data remains inaccessible, particularly if the data is on sites not accessible to web crawlers (Liakos et al., 2015).

Web Data Extraction (WDE) algorithms have had success in extracting template-based data from web pages (Xia, 2009; Ferrara et al., 2014; Liu et al., 2003; Arasu and Garcia-Molina, 2003; Zhai and Liu, 2005). Similar techniques could be applied to extract UGC from pages, but there are several complicating factors, primarily related to advancing design and rendering techniques. Logical page design and rendering on the Internet has changed significantly since many

WDE algorithms were developed, requiring newer structure detection techniques (Blanvillain et al., 2014). Modern pages are often dynamically rendered using AJAX, which many web scraping tools and WDE algorithms do not support. Complex user interactivity has also become the norm for modern sites, and needs to be accounted for.

UGC differs substantially to the data typically collected by WDE algorithms. Most of these algorithms are designed to extract discrete repeating structures - search record results (Liu et al., 2003), e-commerce data tables (Zhai and Liu, 2005) or the data from product detail pages (Crescenzi et al., 2001). These discrete structures may contain nested data, but rarely contain nested structures. UGC differs in this way - comments often have replies that may or may not follow a similar tag structure, and can be nested to many depths. Thus, there may be nested data, but not always predictable nested structures. Any algorithm designed to extract UGC needs to consider and mitigate these problems.

In order to unlock the value of UGC stored within web pages, a complete data extraction framework is required. Designing a WDE algorithm specifically for UGC is not enough - a framework able to automatically navigate pagination structures, expansion links and render dynamic DOM trees is required in order maximise the amount of useful data that may be accessed. UGC structures can then be isolated and filtered to provide a standardised interface for reliably accessing UGC on web pages.

5.2 BACKGROUND

WDE algorithms designed to extract semi-structured data from web pages have been in development for the last 15 years (Atzeni and Mecca, 1997; Crescenzi et al., 2001; Arasu and Garcia-Molina, 2003; Liu et al., 2003; Zhai and Liu, 2005; Cao et al., 2008; Miao et al., 2009). WDE algorithms have had great success in scraping search result pages in order to facilitate web crawling (Liu et al., 2003), as well as collecting e-commerce data (such as competitor prices) (Zhai and Liu, 2005), database detail pages (Thamviset and Wongthanavasu, 2014a) and news articles (Gupta et al., 2015), amongst others.

WDE algorithms use wrappers to extract data from pages, but the method of creating wrappers has evolved over time. Initially, wrapper development was manual and required a developer to write appropriate code to extract the desired fields from a single page (Atzeni and Mecca, 1997; Crescenzi and Mecca, 1998). Most wrappers are not generic, and each new data source required the development of a new wrapper. As this represented a significant development and maintenance cost, automated techniques were highly sought after.

Automated WDE algorithms can be classified based on their method of function and the manner of data input. Functionally, these algorithms work either by comparing the page DOM trees (Liu et al., 2003; Zhai and Liu, 2005; Jindal and Liu, 2010; Sleiman and Corchuelo, 2014; Ferrara and Baumgartner, 2011), by examining and mining the text directly (Thamviset and Wongthanavasu, 2014b,a), or by recognising visually-similar page regions (Liu et al., 2005, 2010). Some algorithms also combine several of these techniques to improve accuracy, such as using text mining to locate data and tree

comparisons to identify the enclosing environment (Thamviset and Wongthanavasu, 2014a). Similar techniques can be used to scrape UGC from pages, hence UGC extraction can therefore be considered a subfield of Web Data Extraction - henceforth, User Generated Content Extraction (UGCE).

Prior research in which UGC has been extracted from from web pages have used manually-developed wrappers to parse page contents Mishne and Glance (2006); Cao et al. (2008). Due to the development time involved in this approach, most studies that used UGC have been limited to platforms that provide an API for data access. For example, Twitter feeds have been used to detect earthquakes Sakaki et al. (2010) and predict depression in social media users De Choudhury et al. (2013b). The data input for these projects is of course constrained by the presence and functionality of a vendor supplied API, in this instance, Twitter. Some studies have accessed web pages as part of social data analytics, but had no automated method of extracting the data from pages (Meneghello et al., 2014).

UGC is plentiful on the Internet, but only API-provided content is easily accessible, and such content only makes up a fraction of UGC created. By developing an automated method to extract UGC in a standardised format, web pages such as news articles, blogs and other widely used media can be unlocked and made accessible as viable data source. This would greatly broaden the scope of data available for use in social analytics applications and decision making.

5.3 CHALLENGES

The ability to extract user-generated content can vary greatly depending on the type of data to be extracted, the intended source, and the intended usage. While a significant amount of UGC is now accessible by API, there is a significant amount that exists merely as rich-text on websites. Data extraction algorithms exist to handle these cases, but only in a number of specific situations, which are described below.

- If the content is provided in a structured format (such as XML, JSON or CSV) then extraction is simple and can be converted into structured data. This type of data is typically provided by APIs.
- If the content is originally in a semi-structured format (such as an HTML table), it can generally be extracted into a structured format - this is a common use case amongst data extraction algorithms. This data is typically provided by "listing" sites, such as auction sites, online stores, or various others. Often, the extraction process is a manually-written wrapper.
- If the content is originally in an unstructured format (such as HTML paragraphs or DIVs), the accuracy and effectiveness of extraction is quite poor. Few options exist to generically extract this data - usually it requires the development and maintenance of a manually-written wrapper.

As much of the UGC across the internet is tied up in unstructured format across millions of websites without APIs to provide the data, it is desirable to design a method of extracting this data in an unsupervised fashion. The goal of this research is to satisfy this aim:

to allow the extraction of unstructured UGC from websites in an automated and unsupervised fashion, returning structured data.

5.4 USER-GENERATED CONTENT EXTRACTION

UGC can represent current events, peoples' views and thoughts on issues or their current state (Subrahmanyam et al., 2008). These comments, statuses, messages and tweets, although given many different names, often possess a common set of attributes. While different sources may provide more or less metadata for a social event, the minimal set usually includes an origin, a timestamp and some content. An origin can be a person, an account or commonly a pseudonym, while a timestamp can be a date, a time, a relative time or any combination of these. Content usually contains the detail about the event occurring - a message between friends, a comment on a topic under discussion or an update on the state of an origin. Fig. 27 shows an example of UGC, with each user comment containing each of these attributes - name, date and content.

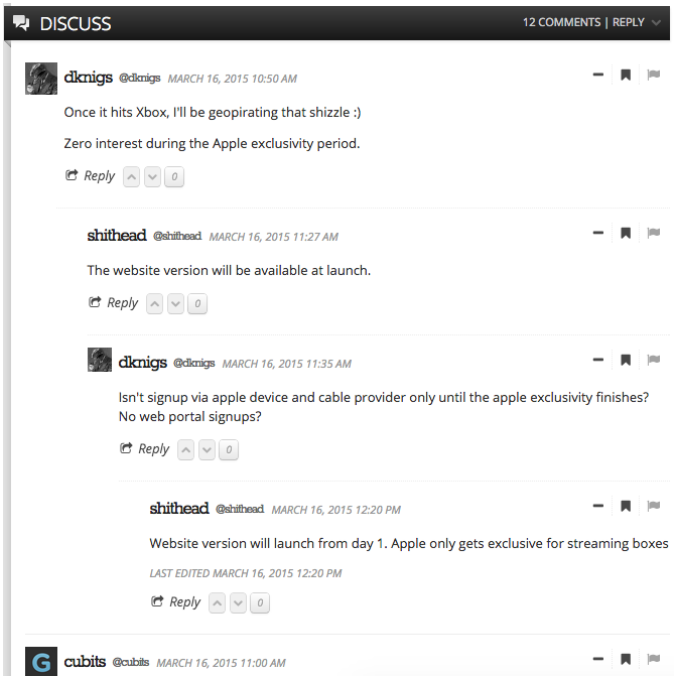


Figure 27: Embedded UGC from a popular news website, ABC Australia.

The UGC present on such pages is similar to that on SNS, except it tends to be anchored around a particular subject or content. This makes page-based UGC extremely useful for user based analytics such as intent or sentiment analysis. UGC is already major consideration for users during eCommerce research (Cheong and Morrison, 2008), often having significant effect during product selection. While data from SNS is frequently used for this, inaccessibility of page-based UGC is an obstacle to providing more effective social analytics. By developing a reliable, accessible method of automatically extracting UGC from pages, this research aims to address this deficiency.

There are three primary steps that must be considered when developing an automated UGC extraction (UGCE) process:

collection: A method of automatically interacting with and collecting data from dynamic sources without any *a priori* knowledge of the source structure;

data extraction: An algorithm to detect commonly-occurring data structures and extract location and data types of fields contained within, while avoiding nested data complications, and;

ugc filtering: A rule-based filter that ensures only UGC is extracted from the page, which enhances precision.

This process does not describe later steps, such as data cleansing, which occur after the data has been extracted from the page and are considered out of scope for this research. If the UGC is present and disassembled into type groups, it can be accessed and manipulated further.

Each of these steps is described further in the sections below.

5.4.1 *Collection*

Prior to running any data extraction algorithms on a data source, significant preparation is required. Because AJAX and dynamic DOM manipulation are in heavy usage on modern sites, it is no longer possible to simply issue an HTTP request and save any raw page data returned. The advent of the "'single page website"' requires that any method of extracting UGC needs to fully emulate a user by emulating the browser rendering, Javascript engine and any required user interaction in order to expose linked UGC hidden behind pagination fields.

5.4.1.1 *Accessing Deep Data*

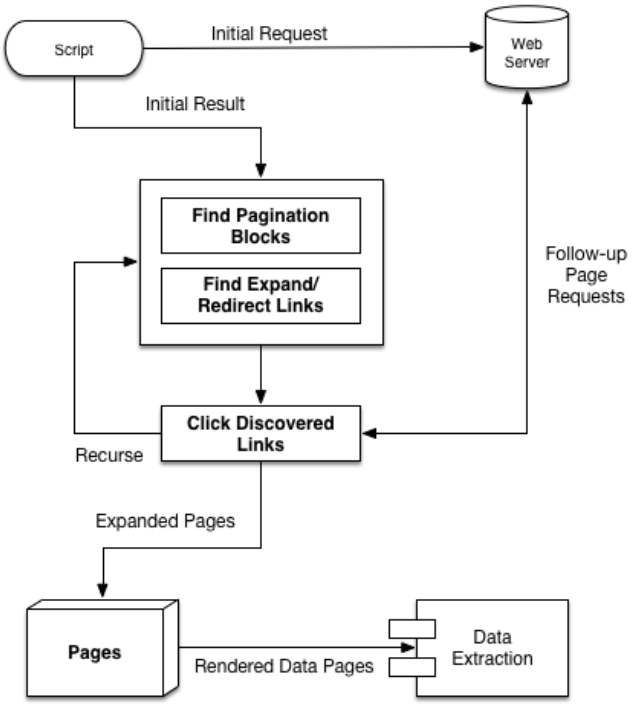


Figure 28: Proposed architecture for pagination handling

In order to interact with these elements, they first need to be reliably identified. Fig. 28 depicts the preparation process. Once the page and all AJAX requests have been completed and rendered, expansion and pagination elements can be identified. This process must be performed recursively - new content injected dynamically into the DOM from an AJAX request may itself contain further expansion or pagination elements, which must also be processed. New page states are stored. Once this process has been completed, page states may be passed to any extraction algorithm - even older algorithms not designed to operate on dynamic DOM trees.

Pagination serves to limit the size of requests required to dynamically render pages, but also to restrict the amount of content available to the user. While a user may not want to view 600

comments at once, applications analysing social comments desire access to every available piece of data, and therefore requires a method of identifying and triggering pagination and expansion elements so that comprehensive data can be collected.

Fortunately, the frequent use of Cascading Style Sheets (CSS) and Javascript frameworks has led to similarities between pagination structures and expansion links. By designing broad rules, most of these structures can be identified regardless of page design. This research identifies three primary structures that require handling:

pagination: Pagination elements generally consist of a numeric sequence containing links to data pages, sometimes within a select box, as seen in Fig. 29.

expansion link: An expansion link is often interleaved throughout nested comment trees, and can be clicked by a user to expand the replies within that thread.

redirection link: A redirect link is used on some pages when comments are completely isolated from the page, and provides a link to another page consisting primarily of social content.

(a) (b)

Figure 29: Examples of pagination elements

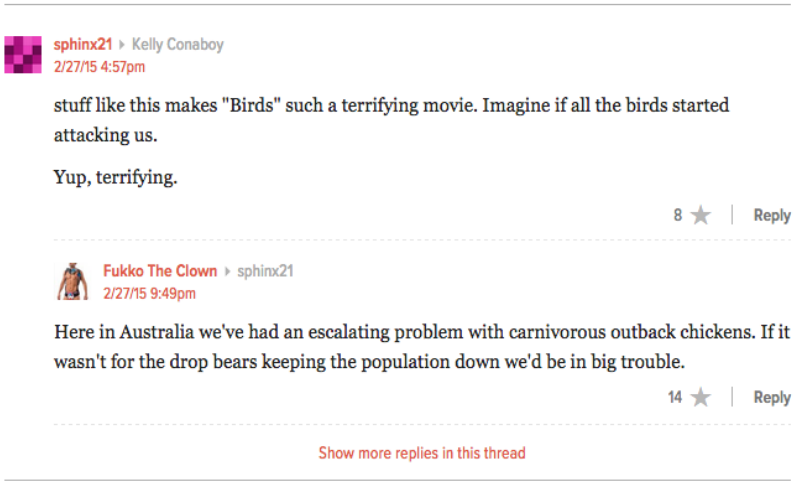


Figure 30: An example of an expansion link within a social data structure

To effectively collect the raw data required to extract UGC, the collection process must be robust and able to properly handle the above structures. Each part of this process is described in the following sections.

5.4.1.2 *Scraping Dynamic DOMs*

To retrieve the fully-rendered page, a method of emulating user interaction and standard browser processes (such as Javascript and page rendering) is required. Solutions have previously been suggested that use an embedded browser to render page content before further processing takes place (Xia, 2009; Mesbah et al., 2012). Automated approaches have no need for actual presentation of the page, so a headless solution is preferable in order to reduce external dependencies and resource usage.

To satisfy these requirements, PhantomJS is used. (Hidayat, 2013). PhantomJS is a headless browser emulator capable of dynamically rendering pages, including handling any requisite Javascript requests.

5.4.1.3 *Locating and Interacting with Data Links*

In order to extract all possible data from a page, user interaction with pagination and expansion elements needs to be simulated. The interaction should cause the browser emulator to dynamically modify the DOM tree, which can then be stored for later parsing by extraction algorithms.

To facilitate this interaction emulation process, Selenium (Holmes and Kellogg, 2006) is utilised, a real-time interactive web testing framework. Selenium is able to emulate user interaction while rendering pages. Combined, PhantomJS and Selenium provide a virtual browser and testing agent that is able to load and render content and interact with it in a realistic manner, while still providing common DOM manipulation and parsing tools such as XPath and CSS selectors.

DISCOVERING DATA LINKS Content expansion elements can be identified by searching for elements containing certain text patterns, which allows a reasonable estimation of redirection and expansion links to be made. Links that lead to additional data are often identified by a qualifier followed by a certain noun, such as "more comments". While there are several English words synonymous to "more" and "comments", a regular expression is able to match combinations of these words in order. Similar principles can also be applied to languages other than English.

Regular expressions can be generated that can be applied to a page to locate data link candidates. The candidates can then be filtered further to exclude any words that can lead us to

unwanted content. Of the discovered elements, those that are able to respond to user interaction are stored.

Once appropriate data link elements have been discovered, Selenium is used to simulate a user "click" on these elements. The resulting data from this interaction - whether a new page or dynamically-loaded content - is stored in an set of page content for later use.

NAVIGATING THROUGH PAGINATION Identifying and interacting with pagination elements on a page is more difficult than the data links in the previous section. While some pagination blocks contain names or classes that can be matched to regular expressions, many structures do not contain relevant identifiers. To detect these blocks, features that define a pagination structure should be searched for instead.

Listing 31 presents a simple algorithm designed to locate these structures. Pagination blocks typically consist of a list of elements containing consecutive ordered integers representing data pages. Because values are often skipped for presentation, the entire list is unlikely to be consecutive - formations like this can be included by instead checking for spans, e.g. 1,2,3,7,15 becomes 1-3,7,15. These numbers can either be the values of individual clickable elements or option fields within a select box, which are both common designs for pagination blocks. Structures that contain value lists following these rules are searched for, which identify pagination blocks. These blocks can also contain "previous", "next" and "all" buttons, which can be used. Repeatedly interacting with the "next" button cycles through every available page, while the "all" button often redirects to a view showing all available data.

```
1: function FindPaginationNumbers(domtree)
2:     for each tag in domtree do
3:         values ← tag.children.strings
4:         digits ← list(value for value in values if value is an
   integer)
5:         spans ← intspan(digits)      ▷ converts [1,2,3,5] into [1-3,5]
6:         if length(digits) > length(spans) then          ▷ found
   consecutive ints
7:             if sorted(digits) == digits then     ▷ ints appeared in
   order
8:                 links ← buttons or links in tag.children
9:                 if length(links) ⩾ length(digits) then
10:                    pagination ← tag
11:                    return pagination
12:                end if
13:            end if
14:        end if
15:    end for
16: end function
```

Figure 31: A simplified example of finding a numbered pagination block

Elements in the DOM tree are matched against these candidate structures. To optimise this process, a regular expression search can be used to quickly locate any possible candidate elements, followed by a slower (but more thorough) iterative search. An example search algorithm is presented in Listing 31, which searches for consecutive integer spans representing clickable elements, such as the example in Fig. 29a. Listing 32 presents a similar algorithm designed to locate pagination select boxes, like the example in Fig. 29b.

```
1: function FindPaginationSelect(domtree)
2:     for each tag in domtree do
3:         select ← tag.find("select")  ▷ find a single child tag of type
   select
4:         if select then
5:             options ← FindAll(select, "option") ▷ find all children
   of tag select that are type option
6:             values ← list(tag.value for tag in options if tag.value
   is an integer)
7:             if length(values) = length(options) and
   length(options) > 1 then
8:                 pagination ← tag
9:                 return pagination
10:            end if
11:        end if
12:    end for
13: end function
```

Figure 32: A simplified example of finding a select-box pagination block

The pagination block can be expressed as an identifying XPath (Meneghello, 2015) which allows the block to be located again easily on new pages. Links such as "next" or "all" are expressed as direct relative XPath (Meneghello, 2015), while numbered elements are expressed as grouped relative XPath. These XPath are then passed to Selenium, which handles the interaction.

Using this process, interaction with sites for the purpose of gathering UGC can be automated. Redirection links are identified and new page states collected. Expansion links are recursively interacted with to expand the DOM tree with additional UGC. Pagination structures are interacted with to include additional UGC pages. Upon completion of this process, a full set of page states are available that can undergo the Data Extraction process presented in the next section.

5.4.2 *Canopied Feature Hashing with Nested Structure Detection*

A structure detection algorithm such as those described in Section 5.2 can be applied to find UGC structures within collected HTML blocks. CFH-NS is a new UGC extraction algorithm developed in isolation from previous research, specifically targeting user-generated content. To implement this algorithm, the Python (Van Rossum and Drake, 2003) programming language was used, along with a heavily modified BeautifulSoup (Richardson, 2013), a DOM tree parsing library, to provide additional functionality.

```
<li>
    <a id="m_ucMessageDisplay1548290_m_anchMessageAnchor"
     ↪  name="m1548290"></a>
    <h3 class="">Patrick:</h3>
    <p class="date">29 Jan 2015 3:15:55pm</p>
    <p>Abbott displays all the hallmarks of a highly
     ↪  delusional right-man. He appears egotistical in the
     ↪  extreme and it should now be obvious to all that he
     ↪  is an extremely dangerous individual and one who
     ↪  should never be in a position of power, let alone
     ↪  being leader of a nation</p>
    <p>
        <span>
            <a class="popup"
             ↪  href="NewMessage.aspx?b=69&amp;t=12532">
                Reply
            </a>
        </span>
        <span>
            <a class="popup"
             ↪  href="AlertModerator.aspx?b=69&amp;m=1548290">
                Alert moderator
            </a>
        </span>
    </p>
    <ul></ul>
</li>
```

Figure 33: An example DOM tree for a single comment

Structure matching operates by breaking a DOM tree into individual branches, anchoring at different nodes within the tree. An example DOM tree for a single social comment is presented in Fig. 33, in which the desired anchor would be the top-level LI tag. To locate similar structures, an attempt is made to match the structure derived from that branch with other branches in the tree. A quick and effective means of comparing these branches is through the generation of a canopied feature hash, henceforth known as a TagHash.

5.4.2.1 *Tag Hashes*

TagHashes express the structure of a tree and its associated tags in a simplified manner, stripping out unique features. Rather than attempting to match element attribute values directly (which may include unique attributes, such as an ID) the *existence* of attributes, specifically ID and name, are matched. This provides a level of differentiation not present if only matching tag types, and assists in tree comparison.

TagHashes are expressed as a Javascript Object Notation (JSON (Crockford, 2006)) string. Figs. 34 presents two example TagHashes of the same structure, Fig. 33, to different depths. Each TagHash is a simplified representation of the DOM tree for that structure, genericised to allow for partial-tree matching whilst retaining some differentiating features. Each TagHash is effectively a signature for the DOM block.

```
{'li.00': [
  {'a.11': []},
  {'h3.00': []},
  {'p.00': []},
  {'p.00': []},
  {'p.00': []},
  {'ul.00': []},
]}
```

(a) Depth 1

```
{'li.00': [
  {'a.11': []},
  {'h3.00': []},
  {'p.00': []},
  {'p.00': []},
  {'p.00': [
    {'span.00': []},
    {'span.00': []},
  ]},
  {'ul.00': []},
]}
```

(b) Depth 2

Figure 34: TagHashes of the structure in Fig. 33

5.4.2.2 *Comparing Tag Hashes*

To match similar structures together and identify as many UGC structures as possible, a method of comparing DOM structures must be developed. Several techniques have been used in previous research (Jindal and Liu, 2010; Bille, 2005; Zhai and Liu, 2005). Initially, a simple tree-comparison algorithm was used that checked for an identical match between structures. Due to the structurally-variable nature of UGC and page design (such as the use of BR or P tags to structure paragraphs), this was later expanded to allow for partial matches.

The TagHashes are compared using text diffing tools. Using a text representation of each TagHash and computing the difference between them using standard tools provides surprisingly good results, including the ability to ignore unaligned branches. The string representations of TagHashes ensure that the type of difference (whether an attribute change, a full branch change or a node change) is easily distinguishable. This even allows for variance in scoring penalties for different tree modifications, similar to how tree edit-distance comparisons (Bille, 2005) function.

```
{'li.00': [
  {'div.01': []},
  {'div.00': []},
  {'p.00': []},
  {'a.00': []},
  {'div.00': []},
  {'a.00': []}
]}
```

(a) TagHash 1

```
{'li.00': [
  {'div.00': []},
  {'p.00': []},
  {'dl.00': []},
  {'p.00': []}
]}
```

(b) TagHash 2

```
--- a
+++ b
{
  'li.00': [
  @@ -0,3 +0,5 @@
  +{'div.01': []},
   {'div.00': []},
   {'p.00': []},
    {
   +'a.00': [],
   -'dl.00': [],
   },
    {
   +'div.00': [],
   -'p.00': [],
   },
  +{'a.00': []}
  ],
}
```

(c) Diff

Figure 35: Example diff: two TagHashes and their resulting diff, showing node and branch modifications

Fig. 35 shows the result of a text diff between two TagHashes. As illustrated, a node addition or subtraction is represented by a line starting with “+" or “-" respectively, with accompanying braces. A tag modification is represented by two lines starting with “+" and “-" and no accompanying braces. These are seen in Fig. 36. The addition or subtraction of a branch is represented similarly to a node addition or subtraction, but contains multiple tags on the diff line. Hence, by checking the number of tags on a diff line the number of tags made up the branch can be determined and scored appropriately - which can be accomplished with a simple regular expression. The diff format provides a simple way of performing scored comparisons by assigning each action (tag modification, tag addition/subtraction, branch addition/subtraction) with a penalty score. By comparing this penalty with the number of tags involved in the comparison, a

similarity score between the trees can be determined - if over a specified threshold, the two trees are deemed similar.

```
+{'div.01': []},
-{'div.00': []},
```

(a) A branch addition/modification

```
+'div.00': [],
+'li.00': [],
```

(b) A node addition/modification

Figure 36: Examples of diff syntax for branch modifications

5.4.2.3 *Finding Repeating Structures*

To detect repeating UGC structures on a page, a list of TagHashes is first built - a TagHashList. This can be built using every possible tag on a page, or restricted to a subset of tag types likely to hold UGC items. The TagHashList class contains additional functionality that automatically groups together structures into buckets based on their TagHash, and can then output common structural XPaths (Meneghello, 2015). The TagHashList also provides the ability to apply custom filters over the data to ensure that the identified structures contain certain items - this is particularly useful when dealing with UGC.

With a populated TagHashList, structural XPath can be generated to match discovered structures. These XPath can then be constructed into wrappers and used to locate structures to extract data from additional pages.

5.4.3 *UGC Filtering*

Upon conclusion of the structure-matching process, a set of similar data structures remains, but little information as to what they contain. As social data is the primary focus of this process, it is desirable to be able to filter the detected data records to only those that are likely

to contain social data. These same principles can be applied to other data types as well - filters can be developed to limit the data records to those containing e-commerce data, for example.

As discussed previously in the section, social data is defined to be a structure that contains at least three primary elements: an origin, a timestamp and a message. In terms of datatypes, this is a string (origin), a date/time/datetime (timestamp) and text (message). To expand the applicability of filters, fields within the discovered structures can be typed and filters designed to match these types.

5.4.3.1 *Type Discovery*

Determining the datatype of a single field is difficult - a set of three words could be a short string, or a block of text. For UGC, the difference is important - a string can represent a user identifier or location, while text usually indicates content. To ensure that fields are not mistyped, individual fields cannot be examined in isolation, but rather grouped with the same fields from other structures that have a matching signature.

With multiple values available per-field, the *probability* that a field is of a certain datatype can be determined and compared to a predetermined threshold. Using this method of probabilistic type-checking, the type of the field can be inferred.

To perform this type determination, the value of each descendant tag is first extracted from the parent structure. By iterating through the tree below the parent, all descendant tags are extracted. This presents a further challenge: if a UGC structure is nested within a parent structure, such as replies to a comment, fields from within the nested structure can be accessed. To ensure that this does not occur, the TagHash of possible nested children are compared against

their ancestors - if a match exists, a nested structure has been discovered. This approach does not work for nested children that have a different structure to parents, however.

Some types require more work to validate than others. Text can be discovered by checking for the presence of newline characters or BR tags and having high average word count, while strings are short and generally do not contain punctuation. Dates can be particularly problematic, as they can be represented in standard ways (e.g. ISO8601 (Klyne and Newman, 2002)) or just as times (e.g. 5:12am) or even in relative terms (e.g. 16 minutes ago). Very lenient date parsing must be used, and the probability-based checking can be used to exclude some fields that contain dates embedded within text content - while some content fields may contain a time or date, it's not likely that every instance will.

The type determinations for each instance of the field are represented as boolean values in a list, and a simple calculation is used to determine the percentage of instances that identify as each type. If any of these types exceeds the predetermined threshold, the field is considered to be of that type. Once an appropriate type has been decided upon, the relative field location and the position of the data (e.g. within a certain attribute, or the text value of the tag) is noted.

During this process, additional data alignment and cleaning is performed. If the field is only aligned with a few structures, it gets tagged as an optional field. Fields that have constant templated values are discarded. Any tag that contains a series of text-like tags is noted as a text parent, and children are discarded from the list - this prevents individual paragraphs being included as separate fields, preferencing the enclosing tag instead.

5.4.3.2 *Field Cleansing*

```
{
  'datetime': ['p[1]/inner_text'],
  'url': [
    'p[3]/span[2]/a[1]/@href',
    'p[3]/span[1]/a[1]/@href'
  ],
  'text': ['p[2]/text'],
  'string': ['h3[1]/string']
}
```

Figure 37: Field types discovered from a set of UGC structures similar to Fig. 33

Fig. 37 presents a completed set of type-checked fields. With each valid field sorted into type-buckets, a filter checking for the existence of certain types can be applied. For UGC, a string field, a datetime field and a text field are required, all of which are present in the figure. Hence, the structure being tested satisfies the requirements of a UGC field and is appended to a list of discovered fields.

```
{
  '// li [ count(a)=1 and count(h3)=1 and count(p)=3 ]': {
    'other': [],
    'content': ['p[2]/text'],
    'datetime': ['p[1]/inner_text'],
    'name': ['h3[1]/string']
  }
}
```

Figure 38: A wrapper constructed from an expanded version of Fig. 33, including fields

Using the discovered set of UGC structures and their fields, data is extracted and aligned. In this phase of filtering, the most appropriate instance of each type is automatically selected. Once completed, the resulting structure and list of fields is generated into a wrapper, seen in Fig. 38.

5.4.3.3 *Filtered Extraction*

```
[
  {
  'other': [],
  'content': ['Abbott displays all the hallmarks of a highly
   ↪  delusional...'],
  'datetime': ['29 Jan 2015 3:15:55pm'],
  'name': ['Patrick']
  },
  {
  'other': [],
  'content': ["Every footy team needs a head-kicker but you
   ↪  do not make him captain"],
  'datetime': ['29 Jan 2015 3:47:38pm'],
  'name': ['Tony']
  },
  {
  'other': [],
  'content': ['@Tony:\nTony Abbott displayed all of his
   ↪  head kicking prowess as...'],
  'datetime': ['29 Jan 2015 4:17:03pm'],
  'name': ['JohnC']
  },
  {
  'other': [],
  'content': ['Like'],
  'datetime': ['29 Jan 2015 6:07:58pm'],
  'name': ['Arthur']
  },
]
```

Figure 39: Data extracted from a document using the wrapper generated in Fig. 38

This wrapper can then be used to extract data from other pages made from the same templates. The extraction process automatically injects the data into appropriately-named fields, as seen in Fig. 39. This data can then move on to additional processing, cleaning and use in analysis.

5.4.4 *XPath Extension Types*

Several algorithms detailed in this chapter make use of custom XPath extension types. XPath (Clark and DeRose, 1999) is a language designed to locate and extract elements within structural XML documents, including HTML. It can select the attributes or values of single or multiple tags at any depth within the XML tree, using direct paths or searching by predicates. XPath is a commonly-used scraping tool, and are often used in data extraction wrappers due to their simplicity. This section introduces an extended use of XPath that is particularly useful for data extraction.

5.4.4.1 *Structural XPaths*

A structural XPath can locate elements in a DOM tree based on a minimal set of available elements. This provides the ability to find objects within the DOM tree that match certain tag layouts, which is useful for tree-matching algorithms. An example of a structural XPath is shown in Fig. 40, which also illustrates that the structural XPath can be generated to different tree depths.

```
1 // li [ count(a)=1 and count(h3)=1 and count(p)=3 ]
2 // li [ count(a)=1 and count(h3)=1 and count(p)=3 and p [
  ↪  count(span)=2 ] ]
```

Figure 40: Two example structural XPaths: to depth [1, 2]

An algorithm to generate a structural XPath is presented in Listing 41. At each level of the tree, the tag types of children are counted. By using the XPath predicate 'count()', elements that contain a precise number of certain tag types can be selected. By going deeper into the tree, more complex structures can be selected because only elements that exactly match the predicate in each branch of the tree will satisfy

the search. Leniency is allowed for structures that contain tags not mentioned by the predicates, as only tags mentioned in a 'count()' predicate are checked.

```
1: function RecursiveStructureXPath(tag, depth, level = 0)
2:     if tag.children and level ≠ depth then
3:         child_count ← count of children by tag type
4:         path_parts ← list("count(child.type)=child.count" for
   child in sorted(child_count))
5:         recursed_children ← list(RecursiveStructureXPath(child, depth, level +
   1) for child in tag.children)
6:         children_xpath ← join(path_parts, " and ")
7:         if children_xpath then
8:             children_xpath ← "[ children_xpath ]"
9:         end if
10:    end if
11:    if children_xpath then
12:        return "tag.name children_xpath"
13:    end if
14: end function
15: function StructureXPath(tag, depth)
16:    return "// " + RecursiveStructureXPath(tag, depth)
17: end function
```

Figure 41: Generating a Structural XPath to a specific depth

Once generated, a structural XPath can be given to an XPath engine and used to locate elements that match the structure described. For data extraction, these XPaths are used to locate elements that match the generated wrapper, and fields can then be derived from those elements using relative XPaths, described in the following section.

5.4.4.2 *Relative XPath*

Relative XPath can be used to navigate from one element to another, if elements are ancestrally related. This provides a simple way of expressing particular fields within a parent element, which is useful for data extraction. An example of relative XPaths are shown in Fig. 42.

```
ul[1]/li[1]/h3[1]
li[1]/a[1]
span[2]/p[4]/li[2]
```

Figure 42: Three example relative xpaths, navigating from one point in the tree to another

An algorithm to generate a relative XPath is presented in Listing 43. Starting at the child, the sibling position of the tag is determined before walking up the tree. Once the parent is reached, the XPath is developed. In order to support a broader range of returned data than standard XPath, the retrieval engine is extended to include several more options, such as *inner_text* and *string*.

```
1:  function RelativeXPath(tag, target)
2:      path_components ← list()
3:      loop
4:          if tag == target then
5:              break()
6:          else
7:              if tag.parent then
8:                  path_components += SiblingPosition(tag)
9:                  tag ← tag.parent
10:             else
11:                 path_components += tag.name
12:                 break()
13:             end if
14:         end if
15:     end loop
16:     xpath ← path_components joined by "/"
17:     return xpath
18: end function
```

Figure 43: Generating a relative XPath from one tag to a parent

5.4.4.3 *Identifying XPath*

An identifying XPath can be used to locate a specific element in the DOM tree, even if the element is not otherwise unique. While some wrappers use tag position to select elements, this is not always reliable - selecting the 15th element in a list won't work on a data

structure that only has 13 elements. To work around this, the nearest unique ancestor is located and a relative XPath is derived from that element. A couple of examples are shown in Fig. 44.

```
1 //ul[@id="comments"]/li[3]
2 //ul[contains(@class, "comments-paginate") and
  ↪  contains(@class, "page")]/li[2]
```

Figure 44: Two example identifying XPaths, from one point to its nearest unique parent

Similar to relative XPaths, the creation of an identifying XPath starts at the child and walks up the tree until a unique parent is discovered. This can be an element with an ID attribute (which are unique) or a set of classes that are likely to be unique - or at least fairly discerning. In some cases, this may be the element itself - in which case, the identifying XPath is quite simple. In others, a unique parent may be several levels up. An algorithm to generate identifying XPaths is presented in Listing 45.

```
1: function IdentifyingXPath(tag, target)
2:     path_components ← list()
3:     loop
4:         if tag == target then
5:             break()
6:         else
7:             if tag.id then
8:                 path ← "tag.name[@id='tag.id']"
9:                 path_components += path
10:                break()
11:            else if tag.classes then
12:                class_paths ← list("contains(@class, 'cls')" for cls in
    tag.classes)
13:                path ← "tag.name[class_paths joined by 'and']"
14:                break(
15:            else)
16:                if tag.parent then
17:                    path_components += SiblingPosition(tag)
18:                    tag ← tag.parent
19:                else
20:                    path_components += tag.name
21:                    break()
22:                end if
23:            end if
24:        end if
25:    end loop
26:    xpath ← join elements in path_components with "/"
27:    return xpath
28: end function
```

Figure 45: Generating a identifying XPath from one tag to a parent

Identifying XPaths are particularly useful for finding and locating specific elements, rather than similar structures. They can be used to express and locate buttons, links or pagination elements.

5.5 DISCUSSION

One of the key research areas for potential improvement identified in Section 2.6 is the ability to access currently-inaccessible social data. This data, such as comments on a news website or reviews on an eCommerce site, often contains valuable information - and

without an API provided to access it, remains unusable without the development of manual wrappers. Since these wrappers incur significant development cost and must be individually written per-site, an automated process for extracting this data could provide significant value to both academic and commercial applications.

The CFH-NS algorithm solves this problem through a number of key functions. Firstly, it allows easy access to data on dynamically-generated sites that do not provide APIs by emulating a browser and automatically stepping through any pagination present. Secondly, it infers a data wrapper specifically for social data (though provides expansion to any kind of data). Thirdly, it uses this wrapper to extract social data from the site and returns it in a known structure, which can then be integrated into the SMAAS framework like any other data.

Because the SMAAS framework provides distributed collection, integration and post-processing, the CFH-NS algorithm can be used much like any of the other data sources, operating over unstructured web pages. Hence, any cross-platform linkages (such as users that share links to news websites over Twitter) can be evaluated by CFH-NS and have data automatically extracted - which, in turn, may contain cross-platform linkages to other sources. This recursive seeking and scraping capability allows the SMAAS framework to perform very broad cross-platform searches for data and sources, and provide the entire set of social events back for presentation and post-processing.

5.6 CONCLUSION

This chapter proposed a new process for the automatic extraction of generic social data from the web, involving three main components: a new method for automatically navigating generic sites to extract social data, a new algorithm for discovering and extracting nested social data structures, and a new technique for typing and classifying social data fields. CFH-NS, a social data extraction algorithm, represents a new state-of-the-art algorithm in automatically extracting user-generated content from generic web pages and significantly improves recall compared to existing data extraction algorithms. The user emulation and interaction techniques can significantly increase the amount of social data collected from generic web pages by allowing for generic navigation of the deep web. Finally, the data typing and classification algorithms provide a standardised interface to user-generated content on pages, and can be expanded easily to other types of data structures.

The combined use of these three techniques increases the reach of social data mining into pages containing generic user-generated content, including news articles and comments, forums and other sites that do not provide an API for retrieving data. This data is made accessible through a standard interface that provides simple integration into analytical applications.

The CFH-NS algorithm in this chapter, as well as the intelligent sourcing algorithm from Chapter 4 and the SMAAS framework from Chapter 3 are empirically evaluated in the following chapter.

6

EVALUATION

6.1 OVERVIEW

Chapter 3 described the design of the SMAAS framework, which aims to source, collect, integrate, query and present social data. This framework enables the use of diverse social data platforms as sources for sensor networking and analytics, and enables collection of user-generated content from previously inaccessible sources such as generic Web 2.0 sites. The framework also uses new techniques to source social data in an efficient fashion, intelligently targeting only relevant data and using the machine learning and natural language processing techniques introduced in Section 4.4 to optimise search queries in an iterative fashion.

This chapter presents evaluations of the major contributions of this thesis: the collection, integration and analytics framework defined in Chapter 3, the intelligent sourcing algorithm described in Chapter 4 and the generic user-generated content extraction algorithms described in Chapter 5.

Section 6.2 evaluates each of the requirements in Chapter 3 to determine whether the framework satisifies the original design goals. Section 6.3 performs an in-depth evaluation of the intelligent sourcing algorithm described in Chapter 4. Section 6.4 evaluates the effectiveness of the user-generated content extraction algorithm by proposing a new testbed that more accurately represents the current

state of the Internet and performing an empirical comparison of the proposed algorithm against other web data extraction algorithms. Finally, Section 6.5 evaluate the use of the SMAAS framework in two real-world scenarios: an implementation and evaluation of an event detection algorithm using the real-time analytical pipeline, and using existing analytical tools to perform off-line analysis of a political event.

6.2 REQUIREMENTS EVALUATION

Chapter 3 defined a set of requirements that each represented the need for a solution to the challenges facing the use of social data in sensor networking. To evaluate the proposed framework's suitability for use in social data mining and analytics, each of these requirements must be satisfied. These requirements are composed of both functional and non-functional requirements, and represent a minimum set of functionality required for the SMAAS framework to perform and support collection, integration and analytics of social data.

The following section will discuss how the SMAAS framework satisifies each of the defined requirements.

6.2.1 *Sourcing*

R1. Must be able to locate application-relevant social data sources in an efficient and extensible manner.

Social Networking Sites (SNS) and social media provide an enormous amount of data from users on a broad range of topics. To

perform efficient collection, integration and analysis of this data, a method of intelligently locating relevant data sources was developed and described in Chapter 4.

The evaluation for this component is more fully described in Section 6.3. It was determined that the algorithm provides a significant increase in the number of relevant data sources over existing search methods. It also introduces an amount of irrelevant noise, but significantly less than collecting all available data. After 4 iterations of the sourcing algorithm, approximately 40% of discovered sources are relevant for the specified topic, compared to an estimated <1% relevancy for the collect-all approach.

6.2.2 *Collection*

R2. Must be able to collect data from identified sources (both SNS and other media) in a generic and extensible manner, regardless of platform.

In order to enable generic social data mining and analytics, the SMAAS framework is required to support the integration of disparate social data sources. While some platforms require the development of an API wrapper, others can use the generic user-generated content extraction method described in Chapter 5. The API wrappers are evaluated in this section, while the user-generated content extraction techniques are evaluated in Section 6.4.

Social data sources that provide an API can be accessed by the proposed framework through the use of an API wrapper, as described in Chapter 3. Due to the development lifecycle of APIs

and adherence to backwards-compatibility, these wrappers require minimal development and maintenance. Only three components require development; authentication, API interaction and data integration.

Each wrapper function can interface with existing community-maintained platform libraries, often dramatically reducing the amount of code required. As the platform libraries abstract the API itself, maintenance of the wrappers is simplified and primarily handled by the developers of the platform APIs. By combining the use of existing authentication and platform libraries, each wrapper requires a minimal amount of code to implement all required functionality for each platform. The length of these wrappers is illustrated in Table 1, which shows that a very small amount of code is required to provide access to a platform, greatly reducing complexity of wrappers.

	Auth	Get	Search	Stream	Integrate	Total
Twitter	13	9	18	39	29	108
Facebook	14	66	31	38	35	184
Reddit	14	24	8	37	58	141
Google	13	3	29		[Uses HTML]	45
RSS		8		19	32	59
HTML		24	[Uses Google]	35	26	85

Table 1: Source lines of code per function in each interface wrapper required to establish functionality for use in the SMAAS framework

The framework is also able to generically collect user-generated content and social data from generic data sources, such as HTML pages. By utilising the new algorithms presented in Chapter 5, significant amounts of data can be acquired from generic sources with no prior knowledge of the format. These results are presented in Section 6.4, and show that the CFH-NS algorithm represents the

state-of-the-art at unsupervised collection of user-generated content from generic pages, outperforming existing algorithms.

6.2.3 *Post-processing*

R3. Must support extensible data cleaning and post-processing methods to ensure integrity of collected data.

Post-processing is an important part of the SMAAS framework that allows for additional processes to enrich social events before they are persisted for long-term analysis. Post-processing steps are not performed for real-time analytics, due to the amount of time added by this process. However, fully post-processed data is available to SMAAS filters through an additional pipeline, in addition to any method of serialisation used.

Each post-processing filter is isolated and performs a single enrichment task, so that filters can be executed over a cluster of worker resources. These tasks range from gender detection of social nodes to detailed semantic analysis of text content, but can also include cleaning tasks that would impact performance if they were to be included in the integration process. For example, a post-processing filter can be used to enrich a Facebook event with the user's location from their profile, which is not provided by the Facebook API. To include this in the integration process would dramatically increase the time taken to collect and integrate events and possibly affect real-time analytics, hence it is completed in a delayed fashion.

Due to the bus-based nature of the SMAAS framework, filters are run in sequence, with an event considered "completed" once all

relevant filters have been executed, depicted in Fig. 46. To improve processing efficiency, each post-processing filter can process many events concurrently through the use of worker pools.

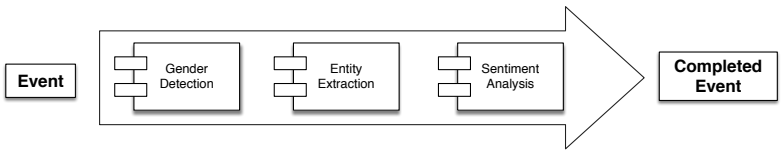


Figure 46: Process flow of post-processing filters within the SMAAS framework.

Fig. 47 shows an event that has undergone post-processing and contains the results of sentiment analysis, entity extraction and gender determination embedded. The gender detection filter operates over the user's given or username. The sentiment dictionary shows positive, negative and neutral sentiment for the whole comment, though other algorithms can break sentiment down into smaller granularity.

```
{
  "origin": {
    "person": {
      "given_name": "Jennifer",
      "gender": "fem"
    }
  },
  "sentiment": {
    "pos": 0.45,
    "neg": 0.23,
    "neu": 0.3
  },
  "entities": ["Abbott", "Liberal", "Australia", "school"]
}
```

Figure 47: A SMAAS event with data enriched through post-processing, including gender, sentiment and entities.

6.2.4 *Integration*

R4. Must support the integration of heterogeneous data sources, such that applications are not required to manually handle data from different sources.

The SMAAS framework handles integration of heterogeneous data sources during the collection phase, as described in Chapter 3. This unified schema is presented to requesting applications, requiring no additional code in client applications to handle data from difference sources. Fig. 48 presents a sample ElasticSearch aggregated output of the "'headers.interface field"', representing the source interface that the event came from, detailing the variety of different interfaces and sources used by the framework when collecting data.

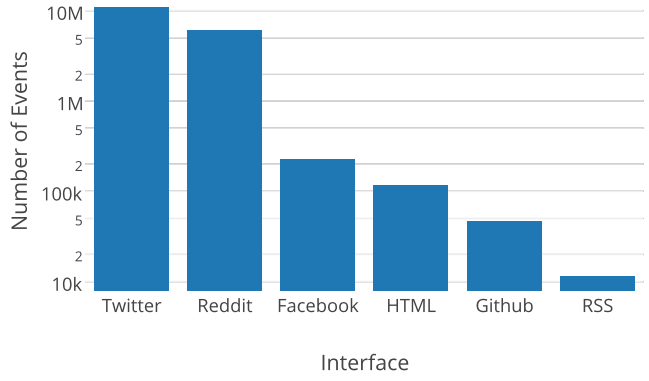


Figure 48: Event count per interface of a sample collection run by the SMAAS framework for three days, consisting of data from approximately 17.5 million unique nodes - each node being a page, a Twitter user, a Facebook page or the like.

Additional collection or cleaning work required for certain sources can be implemented during the post-processing phase, allowing a the integrated set of data to reach the real-time processing pipeline

as quickly as possible while also persisting the most complete version of the event to the chosen serialisation pathway.

To evaluate the ability of the framework to handle the integration of heterogenuous data sources, a pipeline was created. This pipeline consisted of the following:

COLLECTION A set of collection workers were executed and instructed to stream real-time data from all available APIs. This consisted of: Facebook[1], Github, Reddit, Twitter and a recursive intelligent sourcing streaming algorithm, as described in Section 4.4.

INTEGRATION All events produced by the collection workers were integrated into a standard format.

POST-PROCESSING Events underwent sentiment and gender analysis.

SERIALISATION Events were then stored in ElasticSearch.

Some 33 million events were stored as a result of this collection process, running for approximately two days. The integrated data set had a consistent schema as described earlier in the thesis, with variations as expected by variables reported by APIs (e.g. Twitter reports location if enabled, while location will never be present for Facebook messages).

1 While Facebook no longer provides a real-time public Stream API, such functionality can be emulated by actively monitoring popular public Facebook pages, e.g. news pages. In this instance, a number of these pages were monitored: '9news', 'abcnews.au', 'theaustralian', 'news.com.au', 'SkyNewsAustralia', 'SBSWorldNewsAustralia', 'abcnews24.au', 'abc', 'theguardianaustralia', 'mediaspy', 'ConversationEDU', '7NewsAustralia', 'bbcnews', 'nbcnews', 'cbsnews', 'abcnews', 'nytimes' and 'wsj'.

6.2.5 *Real-Time and Delayed Processing*

R5. Must provide unprocessed data in real-time and post-processed data historically, with advanced querying available to historical data.

Both real-time and delayed processing are possible with the SMAAS framework, isolated as two separate processing pipelines.

The real-time processing pipeline is provided with the base social event (prior to any post-processing) immediately after integration, to lessen latency introduced by the post-processing phase. As post-processing can contain a number of network and processing-heavy tasks, the reduction in latency can be significant - particularly if post-processing workers are overtasked, as seen in Section 6.2.7. Real-time analytics can be performed through the use of a filter, as described in Chapter 3 and demonstrated in Section 6.5.

Delayed processing is available in two forms: through a filter (as with real-time analytics) or through the available serialisation pathway. Filter-based delayed processing operates in a similar manner to real-time analytics, but all post-processing is completed and the enriched data is available for use. The post-processed data can also be passed through the serialisation pipeline into software suited for analysis, such as Elasticsearch Kuc and Rogozinski (2013) and Kibana Elasticsearch BV (2015a). Understandably, both of these options introduce additional latency compared to real-time analytics.

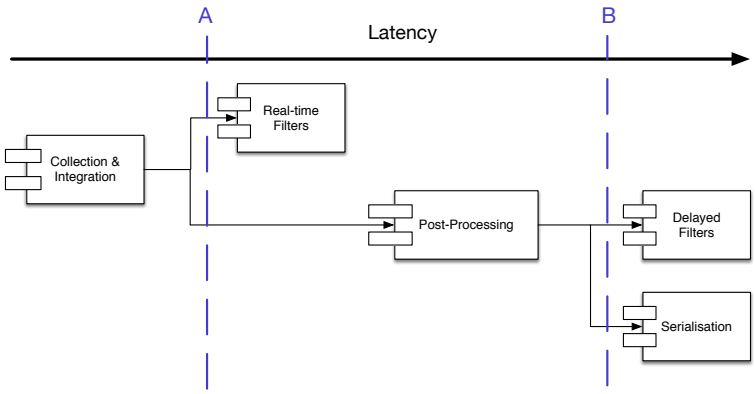


Figure 49: Latency increases at each step of the processing pipeline.

Fig. 49 shows the increase in latency as the pipeline progresses. (A) denotes the phase during which events are provided in real-time, without post-processing. (B) denotes the phase during which fully processed events become available for use in analysis, including during serialisation and delayed filters. For an evaluation of the latency over the different sections of the framework, see Section 6.2.7.

6.2.6 *Presentation*

R6. Must be able to present data in extensible formats for compatibility, including standard formats (XML/JSON) and more complex methods (AMQP, event-passing).

The SMAAS framework supports any kind of data serialiser for querying, storage and presentation by using an intermediary daemon that consumes events from the bus and persists them to the engine of choice. The events can be serialised in any fashion suitable for the data sink - from a relational model for use in RDBMS to a JSON document for Document Storage Engines, as pictured in Fig. 50.

```
{
   "headers": {
      "id": "653964760610926593",
      "locale": "en",
      "interface": "twitter"
   },
   "payload": {
      "content": "Forget Frieze...go to @rownhamshouse nr
      ↪ Soton. Super new work by @Matt_Forster, Alison
      ↪ Orchard and Kate Richardson
      ↪ http://t.co/UfHBorc02c.",
   },
   ...
}
```

Figure 50: Example JSON serialisation of a SMAAS event.

An example serialiser implementation has been provided for ElasticSearch (Kuc and Rogozinski, 2013), which enables persistent clustered document storage and querying. Similarly, serialisers can be written for AMQP, relational databases or NoSQL databases such as MongoDB (Chodorow, 2013), allowing for different types of storage and analysis to be used, and supports integration with existing analytical suites.

During the test run used to collect data in Section 6.2.4, 33 million events were serialised into ElasticSearch and made available for analysis.

6.2.7 *Scalability*

R7. Must be able to scale to a wide range of hardware, from high-resource clusters to commodity hardware.

A major requirement of the SMAAS framework is the ability to scale, from low-throughput deployments on commodity hardware to high-throughput deployments across processing clusters. This

allows for the use of small deployments to monitor a small number of interfaces or a subset of available data from interfaces, while also being capable of handling real-time data firehoses from a number of interfaces simultaneously.

To evaluate the scalability of the SMAAS framework, a number of experiments were performed using simulated social data to test the performance of post-processing - the most processing-intensive task performed by the framework. For the framework to operate at full effectiveness, it must be able to post-process tasks at a rate equal to collection. If it is unable to do so, only unprocessed events are available for use in real-time analytics.

For this experiment, three post-processing filters were enabled: entity extraction using a custom NLTK pipeline (Bird, 2006), gender detection using SexMachine[2] and sentiment analysis using VaderSentiment (Gilbert, 2014). Entity extraction performs natural language processing (NLP) of event content to extract stemmed and lemmatized Named Entities[3]. Gender detection attempts to determine the gender of an event origin by analysing the user's name or identifier. Sentiment analysis determines the event's sentiment towards certain entities within the content.

Table 2: Test platform instance types, running on Amazon Elastic Computing Cloud.

<table>
<tr><th></th><th>Single</th><th colspan="2">Small</th><th>Large</th></tr>
<tr><td>Event Bus</td><td rowspan="5">M4.4XLarge</td><td colspan="2" rowspan="2">M4.2XLarge</td></tr>
<tr><td>Logserver</td></tr>
<tr><td>Serialiser</td><td colspan="2">M4.XLarge</td></tr>
<tr><td>Collectors</td><td colspan="2">4x T2.Large</td></tr>
<tr><td rowspan="2">Post-Proc</td><td colspan="2">4x C4.2XLarge</td></tr>
<tr><td colspan="2"></td><td>4x M4.2XLarge</td></tr>
</table>

2 Available at https://github.com/MarcSalvat/sexmachine

3 Named Entities in this context include nouns, proper nouns and other identifying words, not including any stopwords.

The tests were conducted on three test platforms, shown in Table 2. The first was using a single Amazon EC2 M4.4XLarge instance running Ubuntu 14.04, RabbitMQ 3.5.6 with Erlang R16B03 for an event bus, Redis 2.8.4 as a job queue, ElasticSearch 1.7.2 to serialise and query processed data and the development version of SMAAS using Python 3.4.0. The second was a small cluster, consisting of an M4.2XLarge instance for the event bus running RabbitMQ and Redis, an M4.XLarge instance running ElasticSearch, a T2.Medium instance running a network logserver and 4x C4.2XLarge instances, each running 8 post-processing worker processes. The third was a larger cluster, consisting of the same resources as the small cluster but with an additional 4x M4.2XLarge post-processing instances.

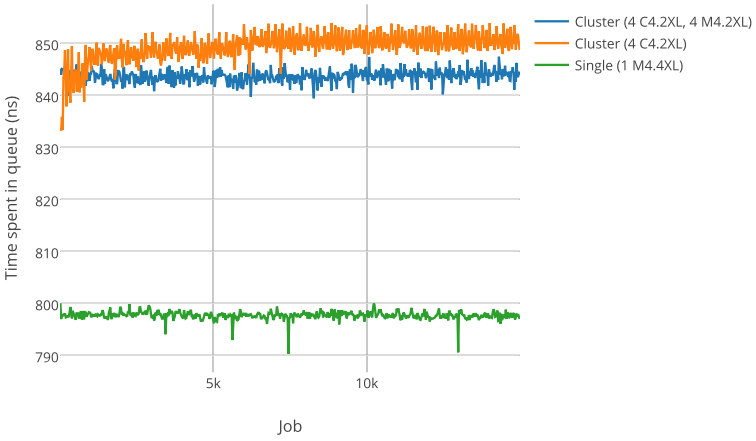
Each test consisted of the following steps:

1. Initialise EC2 resources with default configurations
2. Execute server, logger and worker processes
3. Execute simulate.py script to start pushing events into the bus at a defined rate
4. Log post-process task queue / start / end times
5. Stop simulate.py script after 10 minutes
6. Wait until post-processing has completed
7. Store, parse and analyse logs for performance data
8. Terminate and delete EC2 instances

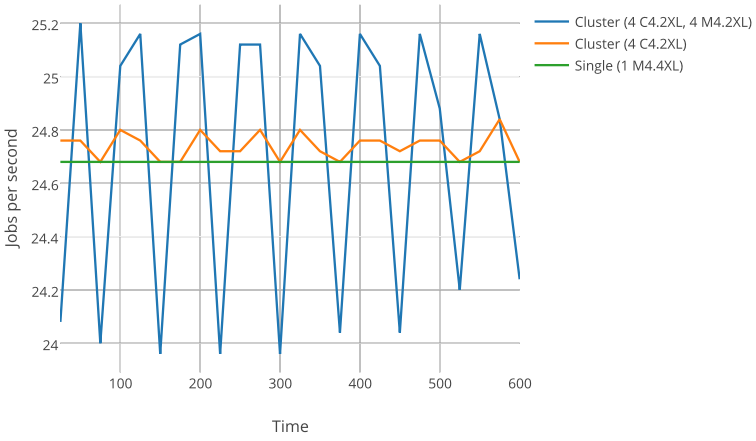
This process, including the initialisation and destruction of EC2 resources, was automated using Python and the Amazon AWS API. The steps were repeated on each platform for different rates of event production: 25, 50, 100 and 1000 events per second (EPS). Hence,

each test resulted in different total numbers of events produced: 15,000, 30,000, 60,000 and 600,000. Data in each event was partially randomised, presenting a similar level of computational difficulty for the post-processing algorithms. Times were logged for a number of checkpoints in the process: the time each task entered the queue (usually just prior to processing), the time it took to start processing the task, and the time taken to complete processing.

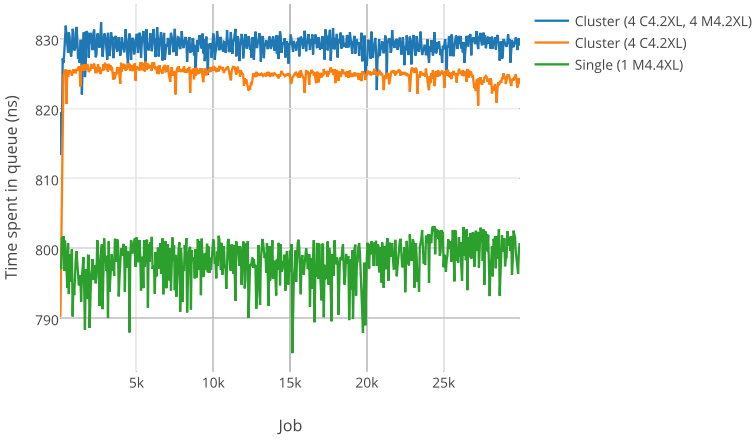
Figures 51a through 52c present the time each task spent in the work queue for each platform and EPS rate, represented as a rolling mean across the lifetime of the processing work. More detailed graphs are available in Appendix A, including processing time.



(a) Time tasks spent in work queue, 25 events per second



(b) Cluster processing speed, 25 events per second

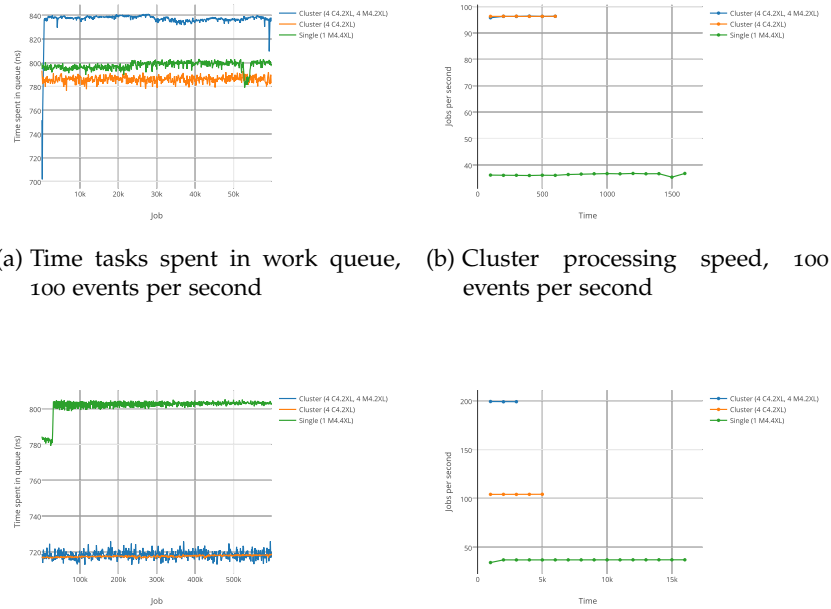


(c) Time tasks spent in work queue, 50 events per second

Unsurprisingly, the queue is susceptible to datacenter and hypervisor-level/virtualisation latency, resulting in better performance for low EPS rates on the single-instance platform, as seen in Fig. 51a. As the event bus and workers exist on the same resource, there is little latency and no network-related latency spikes, with the queue only affected by hypervisor-level lag. By contrast, both clusters have both increased and more variable queue times due to the additional latency involved in workers requesting, retrieving and responding to the event bus present on a different logical (and possibly physical) resource.

Fig. 51b shows that at 25 EPS, all platforms are able to complete assigned tasks at the same rate that they are presented.

At 50 EPS, the single instance is no longer sufficient to process all available tasks in time, shown in Fig. 51d. Both clusters are able to keep up, but the single instance can only process approximately 27 EPS. As a result, the single instance takes nearly 35% longer to process the full set of 30,000 tasks. However, Fig. 51c shows that tasks still spend less time in the queue due to the additional latency and overhead required to use a remote worker pool.



(a) Time tasks spent in work queue, 100 events per second

(b) Cluster processing speed, 100 events per second

(c) Time tasks spent in work queue, 1000 events per second

(d) Cluster processing speed, 1000 events per second

Figure 52: Post-processing performance at different EPS.

At 100 EPS, some of the processing latency resulting from multi-tenancy on EC2 instances appears, present in Fig. 52a and Fig. 52b. The single instance suffers from a drop in processing resources after processing approximately 30% of jobs, and a temporary drop after 80%. The other platforms suffer minor spikes in available resources.

While both clusters are able to handle 100 EPS, the single instance performs poorly. Fig. 52b shows the additional time required for the single instances to complete all assigned tasks, nearly a 160% increase over the clusters.

Generating 1000 EPS shows the true scalability of the framework. While none of the test platforms are able to cope with the enormous load, Fig. 52d shows the maximum processing rates for each. While the small cluster achieves a throughput of approximately 105 EPS, the larger cluster with an additional four worker instances manages an average of 200 EPS. The small drop in efficiency is accounted for

in the difference between resources used - while both use C4.2XL instances (which have 31 Elastic Compute Units [4](ECU) available), the larger cluster also uses four slightly less powerful M4.2XL instances (26 ECU).

Table 3: The scalability of SMAAS components, either by the framework or dependent on choice of implementation.

Component	Scalable by SMAAS	Scalable by Impl.
Sourcing	x	
Collection	x	
Integration	x	
Post-Processing	x	
Event Bus		x
Serialisation		x
Presentation		x
Logging		

The SMAAS framework was designed to be highly scalable, in order to cope with varying levels of data throughput. The ability to scale different SMAAS components is presented in Table 3, which shows components that are scalable as part of the framework, or those that are dependent on the selected implementation. For example, the Sourcing, Collection and Integration components are all scalable through the use of a SMAAS worker pool, such as those evaluated above. The scalability of the Bus, Serialisation and Presentation components depends on the technology selected for those components - the default Bus implementation using RabbitMQ is scalable, for example, as is ElasticSearch for Serialisation and Presentation. The only component of the framework not scalable by default is the Logging server, which has a very limited amount of work - in the event that more processing resources were needed, a

4 A measure of the amount of processing resources available to an Amazon instance. They are an indicator of the maximum processing power attainable, but are not necessarily always available due to shared tenancy. https://aws.amazon.com/ec2/faqs/

more powerful instance could be used before any horizontal scaling was required.

These results indicate that the SMAAS framework scales well to different throughput, with the large cluster able to handle the data collected by all generic streaming interfaces currently implemented[5]. To handle additional data streams, it is trivial to add additional processing resources.

6.3 INTELLIGENT SOURCING EVALUATION

One of the prominent and novel advantages of the SMAAS framework over existing methods of social data collection is the use of on-demand social data sourcing. Effective data analytics relies on the presence of quality data sources, so this is an important goal. The evaluation in this section emphasises data sourcing, and examines the relevance and quality of data sources discovered.

6.3.1 *Experimental Setup*

The sourcing algorithm was implemented as a SMAAS collection task, taking advantage of the Natural Language ToolKit (Bird et al., 2009) library for text mining. The sourcing algorithm was executed on an Amazon EC2 m3.medium instance, and individual experiments were conducted in a single run to ensure that the available social data did not dramatically change between runs. All experiments in this section were conducted in August 2014.

5 While this amount varies depending on the time of day, event throughput from the generic streaming interfaces of Reddit, Twitter, Facebook and a subset of RSS feeds averages approximately 120-180 EPS. Twitter's public streaming API delivers a sample portion of data created, which is variable.

The algorithm used the SMAAS platform APIs to search and collect data, as well as a generic web scraper used to further build keyword lists from sources. Most services are subject to API search throttling limits, and these limits are taken into account during the sourcing process [6].

The process for each experiment consists of a series of iterations. A single iteration consists of the following steps (subject to configuration values):

1. Search using initial seed keyword[7]
2. Generate target source list
3. Retrieve content from list of sources
4. Mine content for commonly-occurring keywords and metatags (URLs, Usernames, Emails)
5. Further add to source and keyword lists
6. If [broad_search]: Search using metatag lists (ie. Twitter users) and add to sources
7. Iterate using expanded keyword set

There are a number of possible configuration options for each experiment, which are explained below:

BROAD_SEARCH Whether the sourcing algorithm should also take sources from collected content

MAX_SEARCH_RESULTS The number of results to return from a search (important to avoid API throttling)

6 Most APIs place limits on searches and other functionality, while not limiting access to real-time streams.

7 It is worth noting that a typical manual search on social media equates to only performing this first step.

The broad search algorithm also collects special metatags from content, such as usernames and hashtags on Twitter, and email addresses or URLs in static content. These metatags are then used to broaden the search scope, discovering additional search vectors and providing a significantly higher number of data sources while being subject to much higher noise levels. These searches are examined in the following section.

6.3.2 *Results*

6.3.2.1 *Relevance*

The relevance of discovered data sources is an important metric in evaluating the usefulness of this sourcing algorithm. In order to evaluate this, the algorithm was given a broad seed keyword ("Australian politics") and let to run over multiple iterations. The source list was exported to a comma-separated value file and each source was manually evaluated for relevance to the topic. This process was executed twice: once as a normal search, and once with the additional broad searching options enabled.

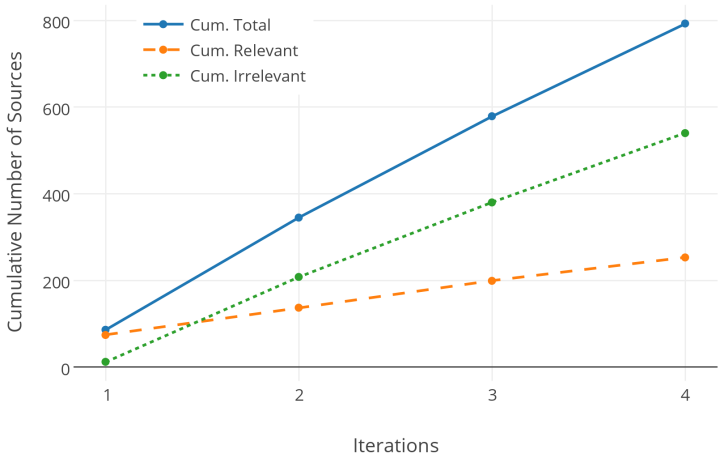
Sources are considered relevant and useful based on the following set of criteria:

- Whether the source provides data containing one of the given query keywords,
- Whether the source will continue to provide links and other sources into the future, and
- Whether the source provides data that is of high enough quality to be useful without requiring significant platform-specific cleaning.

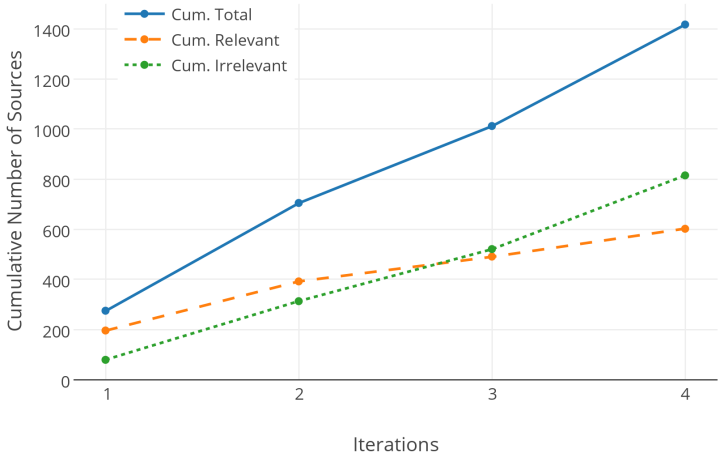
By using these criteria and weighting them depending on application parameters, each potential source can be scored to evaluate relevance. If a source surpasses a predefined relevance score, it is considered relevant. All other sources, including those recorded as a result of algorithmic mishaps[8], are deemed as irrelevant. Relevance is a therefore a boolean result.

Each sourcing iteration consists of a single query to a data source, followed by a period of post-processing. Hence, the cumulative results collected by the algorithm on the fourth iteration have required a total of four queries.

8 This includes results that are misparsed due to malformed HTML and useless sources such as embedded advertising servers, both of which are considered failures of the algorithm.



(a)

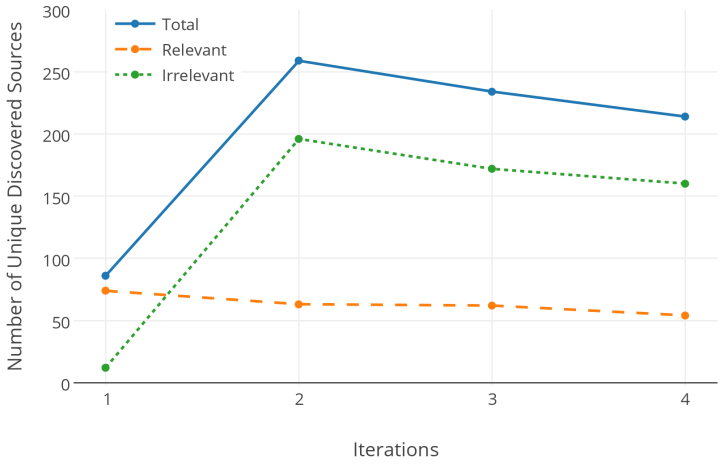


(b)

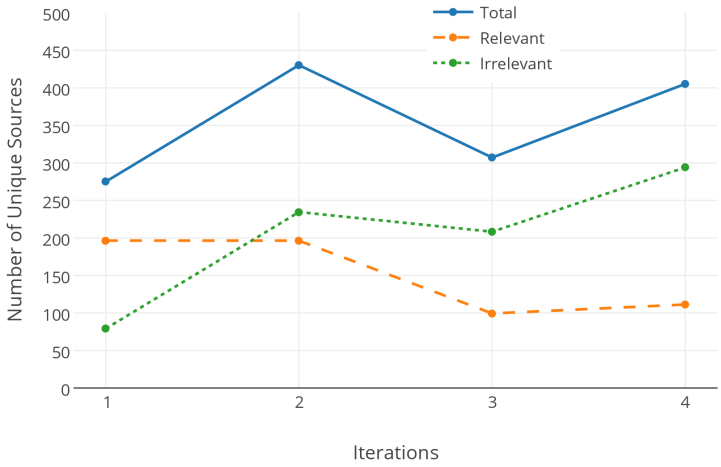
Figure 53: Cumulative relevance of data sources; a) standard, b) broad

The most relevant sources are quickly and easily found by a normal search with a low percentage of irrelevant results in the first sourcing iteration, as seen in Fig. 53a. This is to be expected, as the search APIs provided by platforms are usually quite good at locating the small amount of most relevant information quickly. The broad search finds a higher number of relevant sources, but with a significantly higher number of irrelevant sources, shown in Fig. 53b. In both instances, these sources are expanded and new keywords are

derived, and additional relevant sources continue to be found at a lessening rate.



(a)



(b)

Figure 54: Relevance of data sources; a) standard, b) broad

After the first two iterations, new sources continue to be found at a relatively linear rate. The relevance of discovered sources decline steadily relative to the total after the second iteration, but still maintain an acceptable rate of discovery. Fig. 54b shows a steep increase in irrelevant sources during the fourth iteration, as the search space expands out beyond any semblance of relevance. The

broad search maintains a much more even percentage of relevant results over the search space.

6.3.2.2 *Keywords*

The search keywords (initially seeded as "Australian politics") are expanded based on discovered content and the most relevant keywords float to the top of the list. The first iteration of both searches immediately expand the keyword set to contain mostly relevant keywords, as seen in Table 55. Successive iterations narrow the search space down to a specific set of topics that accurately frame Australian politics. Interestingly, the broad search (which relies more heavily on static page content rather than real-time social networks) contains a higher number of historical keywords. A number of these keywords from a broad search relate to the government as of 2013, whereas those from the normal search relate more to the state of Australian politics, circa 2014.

The difference in keyword sets between the search types also affects the relevancy of discovered sources after successive iterations. The iterative improvement of the search space during broad searchers potentially explains why even the third and fourth iterations of searching still contain a relatively high percentage of relevant results, as seen in Fig. 54b. The normal search relies on a much smaller set of newer content from which to derive keywords, resulting in a lower discovery rate of historical data sources.

Iterations			
1	2	3	4
politics	abbott	abbott	abbott
australia	australia	government	tony
government	government	australia	australia
australian	news	tony	government
university	australian	news	news
party	minister	minister	minister
minister	party	party	party
abbott	politics	people	people
news	tony	pm	politics
media	people	politics	australian

(a)

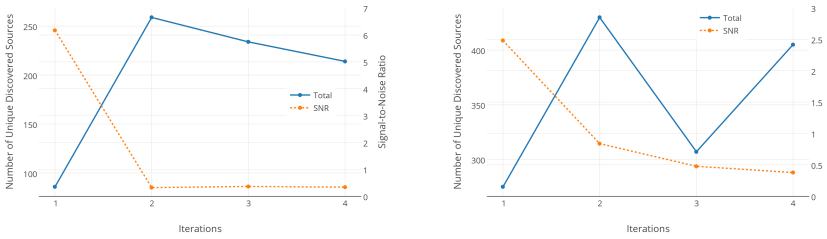
Iterations			
1	2	3	4
politics	australia	australia	abbott
australia	government	abbott	australia
australian	politics	government	rudd
government	party	rudd	labor
party	australian	party	government
news	abbott	minister	party
media	minister	labor	minister
world	labor	australian	news
minister	rudd	politics	australian
abbott	pm	news	election

(b)

Figure 55: Top 10 keywords used for searches; a) standard, b) broad

6.3.2.3 *Signal-to-Noise*

The signal-to-noise ratio of each search type was also examined, relative to the total number of sources discovered. A normal search discovers relevant data sources at a ratio of over 7:1 during the first iteration, seen in Fig. 56a, but immediately drops to a very low success rate in successive iterations. By comparison, Fig. 56b shows that the broader search starts with a much lower success rate of



(a) (b)

Figure 56: Signal-to-Noise Ratio of data sources; a) standard, b) broad

approximately 2.5:1, but maintains a steadier ratio well into later iterations, providing a more steady flow of new relevant sources. As explained in Section 6.3.2.2, this is likely due to the broader search touching on a larger source of historical data sources due to differences in keyword selection.

6.3.2.4 *Analysis*

The results of the sourcing algorithm indicate that the search methods (normal and broad) operate along different parameters. Normal searches rely more heavily on new data provided by real-time sources such as Twitter and Reddit, while the broad searches derive keywords primarily from older data sources such as newsfeeds and articles found by Google. As a result, the training of keyword sets tend toward two different trends: modern and historical, but could also indicate a disconnect in discussion between traditional media and social media. Both of these search types are useful, depending on the desired application. Overall, both search types provided a significant number of relevant data sources and ultimately achieved their goal - to optimise the collection of social media data.

These sources can then be passed directly into the collection and streaming components of the SMAAS framework, allowing for

continuous collection from relevant sources. This ensures that relevant data is collected while not requiring a massive amount of resources to monitor a very wide array of (potentially static) content.

There are a number of improvements that could be made to the sourcing algorithm. One would be to increase the use of training to include potential sources, preferring those new sources that had multiple existing links to discovered sources. A second involves a combination of both approaches, using historical keywords to search social media sources and modern keywords to search historical data sources.

6.3.2.5 *Limitations*

While the intelligent sourcing algorithm performs well at retrieving additional relevant data sources than a normal search while keeping a relatively low signal-to-noise ratio, it has a number of limitations. It relies on the presence of an initial keyword that gathers enough results to be able to iteratively improve the search space. If the initial keyword is rarely used or not well-linked to the desired material, it is unlikely to gather many additional sources. This also implies that an initial focus area can be selected for wide-ranging event detection using generic real-time streams are far more effective. Additionally, due to API limits on most social APIs, using extremely broad seed keywords is likely to gather only a fraction of available data.

Sources that come from a platform API (such as Twitter) can be directly collected by using said API, but there are no APIs provided for static content and web pages heavily featuring user-generated content, such as comment-enabled news articles. The SMAAS framework provides a method to extract data from these pages, which is evaluated in the following section.

6.4 GENERIC COLLECTION EVALUATION

The Canopied Feature Hashing with Nested Structure detection (CFH-NS) algorithm in Section 5.4.2 was developed for use with the SMAAS framework to enable the extraction of user-generated content from static and dynamic pages that are not accessible through an API or other interface. This allows the framework to collect from a much broader range of data sources than previously accessible. This section presents an evaluation of this algorithm, and evaluation of the automated interaction and social filtering techniques presented in the same chapter.

The effectiveness of the CFH-NS algorithm presented in Section 5.4.2 is evaluated by comparing it with similar algorithms from previous research, in keeping with previous evaluations of data extraction algorithms (Liu et al., 2003; Thamviset and Wongthanavasu, 2014a; Jindal and Liu, 2010). To test these algorithms, each was run against a testbed of HTML data and extracted structures were counted to determine recall and precision. To evaluate the effectiveness of automated interaction algorithms, the amount of data collected using traditional HTTP requests was compared with the data collected using the interaction and rendering techniques.

Testbeds for web data extraction algorithms already exist, such as TBDW (Hirokawa Lab, 2004), ViNTs (Liu et al., 2005) and others. Each of these testbeds contains a set of pages that contain data records, such as Search Record Results or List Records. Evaluating an algorithm against these testbeds is relatively simple - the algorithm is run against the pages and the number of detected data

records is compared with the number that are known to exist on each page.

Two testbeds commonly-used in previous research (TBDW and ViNTs) were compiled in 2004 and 2005 respectively, and represented typical web design techniques for that time. However, web design has dramatically changed in the time since then. HTML standards and design paradigms have evolved to encourage better structure while introducing higher complexity, which significantly alters the requirements for web data extraction algorithms. Modern pages also rely heavily on dynamic DOM interaction to render content, making the collection mechanisms implemented in many WDE algorithms non-functional.

These changes have effectively rendered the standard evaluation testbeds inappropriate, as they no longer represent the state of design on the Internet. Regardless, they would be unsuitable for evaluation in this context: user-generated content was seldom embedded into pages during the time period in which the testbeds were compiled. The only sites that tended to contain UGC were message boards or forums, whereas the shift to Web 2.0 approaches has driven the expansion of primarily user-generated content pages. These fundamental changes required not only the development of new extraction techniques, but also of new evaluation techniques.

6.4.1 *Methodology*

To test the CFH-NS algorithm against existing solutions, a new testbed was compiled that is significantly more representative of the current state-of-the-art in web design and development methodologies.

A set of candidate URLs was crowdsourced, with participants directed to provide sites that they commonly visited that used UGC in some way. The candidate URLs provided covered news sites, social media, blogs and online stores. The list was then filtered and URLs excluded based on several criteria:

- the presence of an inappropriate amount of UGC (either not enough to provide an appropriate sample, or so much as to cause difficulty in storage);
- multiple sites with duplicate structures, e.g. blogs operating on the same software with similar design themes;
- very obscure sites or those that are unrepresentative of the state of the Internet, e.g. sites that have not undergone design upgrades in the last decade; and
- sites on unreliable servers that did not always return page data in a timely fashion.

The candidate list was culled in this fashion to a total of 49 URLs. The pages require various amounts of interactivity to retrieve data, consisting of different types of pagination and expand/redirect link types and covering a wide range of design strategies for these elements. The page designs also vary, with some using tag types in structures (such as list elements), and others preferring to use DIVs for everything (which can make extraction significantly more difficult). The site designs represent modern web design, and are representative of current and popular content on the Internet.

Some pages require dynamic DOM parsing, while others provide content statically. This means that algorithms that perform their own data collection must be able to handle dynamic content, else the content is unavailable. In instances where it was possible, we have

directly supplied the data to the algorithm, reducing its responsibility to structure detection.

While the older testbeds are unsuitable for use in this evaluation, the algorithms themselves are not. Although they are designed to detect data records such as search results and e-commerce data tables, UGC constructs share many of the same qualities and WDE algorithms should be able to detect (but not necessarily isolate) UGC. Hence, we only evaluate the effectiveness of structure detection for other algorithms.

The full list of URLs selected is presented in Appendix A.

6.4.2 *Results*

6.4.2.1 *Social Data Extraction*

Web Data Extraction algorithms come in two broad types: those designed to operate on single pages, and those designed to operate on sets of similar pages. As the CFH-NS algorithm is designed to operate on a single page, it was benchmarked against other algorithms of this type.

The acquisition of the code for several WDE algorithms was attempted by directly contacting the authors of previous research, but limited replies were received. Few algorithms are provided online, and a number of those are now non-functional. Hence, the algorithms being evaluated are DEPTA (Zhai and Liu, 2005) and BUW (Thamviset and Wongthanavasu, 2014a), both of which are publicly available.

CFH-NS is able to split up fields into their appropriate type and identify fields relevant to social data, which neither DEPTA nor BUW was designed to do, so this functionality was not evaluated.

The DEPTA and BUW algorithms are evaluated slightly differently. Neither algorithm provides the ability to filter social data, so this process was performed manually upon the various record types returned by each algorithm. If a discovered record represented a single social comment that contained the necessary fields outlined in Section 5.4.3, it was deemed a successful extract. Any records that contained nested data, did not contain the necessary fields or represented a field within a record were considered a failed extract. Records that were unrelated to social data were discarded and not included in the evaluation, and are not reflected in recall or precision.

DEPTA provides a Java binary that can be operated on a set of collected data, but BUW only provides a live web interface and performs its own collection.DEPTA was able to be passed a full set of post-rendered and interacted content, but BUW was unable to perform these duties on its own. To account for these problems, two sets of tests were completed. Using data procured, interacted and rendered by CFH-NS, performance of both CFH-NS and DEPTA were evaluated for extracting social data structures. Then, using their own collection mechanisms, DEPTA and BUW were evaluated. In keeping with evaluation practices in previous research, two sets of results were provided for recall and precision: "All Results" describes recall and precision if including collection or WDE failures as part of the results, while "Success Only" excludes failures and only determines recall and precision for pages in which at least one result was found.

Recall represents the number of relevant records available on a page that were successfully discovered, while precision represents the number of discovered elements that were relevant. F-score is

Table 4: Summary of Experimental Results

	AR-P	CFH-NS	DEPTA	AR	DEPTA-L	BUW-L
Total	3155	3454/2981	853/764	1675	149/124	704/593
All Results:						
Recall		94.5%	24.2%		7.4%	35.4%
Precision		89.2%	89.6%		83.2%	84.2%
F-score		**0.92**	0.38		0.14	0.50
Success Only:						
Recall		94.5%	48.1%		48.1%	71.6%
Precision		89.2%	89.6%		83.2%	84.2%
F-score		**0.92**	0.63		0.61	0.77

calculated using a standard balanced algorithm that represents the harmonic mean of precision and recall, seen in Eq. 1.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

A summary of these results are presented in Table 4, while the full results table can be viewed in Appendix A, in Section A.1. AR-P is the number of available results from the paginated dataset, produced by CFH-NS. AR is the number of available results without any interaction required. The figures in each algorithm's column represent the total number of results found and the number of relevant results. DEPTA-L and BUW-L are the "live" versions of each algorithm, and represent the results if the algorithms perform their own data collection rather than using CFH-NS's collected set.

As seen in the table, CFH-NS provides a significant increase in recall compared to DEPTA using the paginated set, while retaining similar precision. This also represents a significant improvement in both recall and precision compared to BUW, though BUW performs well when only counting successful tests. When comparing all results, CFH-NS represents a major improvement in F-score.

A large part of the difference in recall between CFH-NS and the live versions of DEPTA and BUW is due to dynamic DOM parsing. Neither algorithm is able to render the DOM dynamically, and are unable to retrieve the social data on pages. By counting the URLs in which both DEPTA and BUW fail to retrieve data, the number of pages that use dynamic DOM injection to render social content can be determined. 51% of sites tested require dynamic DOM parsing to retrieve social data, indicating that algorithms without the ability to deal with this problem are unable to collect significant amounts of social data.

Each algorithm presented was built with a specific purpose in mind, but have often been cross-applied to different usages. The DEPTA algorithm was designed primarily to extract eCommerce data from table structures, while BUW was designed more generically to read complex tabular data from pages. CFH-NS was designed primarily to extract social data, and hence performs much better at this task than previous algorithms. The principles used in CFH-NS apply fairly generically, and it should perform well for extracting eCommerce and tabular data from pages - however, this functionality was outside the scope of this research and not tested.

CFH-NS represents a major improvement in the ability to collect social data and user-generated content from rich-text web pages. Integrated into the SMAAS framework, this improves the accessibility of data that was previously unavailable for use, and broadens the scope of data available for social data analytics. While manually-written wrappers are far more accurate at performing this data extraction, the cost of developing them is prohibitive - this automated algorithm significantly reduces cost while providing a significant amount of data.

6.4.2.2 *Dynamic Interaction*

To test the effectiveness of automatic dynamic interaction, the number of records available on a page without interaction was compared against the number of records available using interaction. Some pages in the dataset normally used pagination, but there was not enough social data present to be paged. Of 49 pages, 10 pages (20.5%) had data hidden behind pagination or expansion links.

Table 5: Additional data retrieved through automated interaction

URL	AR	AR-P	Increase
newsfeed.gawk...	20	42	210.0%
the-toast.net...	27	41	151.9%
www.adelaiden...	50	101	202.0%
www.cracked.c...	34	592	1741.2%
www.dailytele...	50	72	144.0%
www.destructo...	50	219	438.0%
www.dpreview....	176	193	109.7%
www.smh.com.a...	10	47	470.0%
www.theage.co...	10	545	5450.0%
www.tripadvis...	10	20	200.0%
Totals	437	1872	428.4%

These pages and the results of automated interaction are presented in Table 5. AR-P represents the number of results available using automated interaction, while AR represents the number of results available without interacting with the page. As presented in the table, using automated interaction provided 428% more social data with these pages than without using interaction, a substantial increase.

Pages that take advantage of pagination often do so by necessity, due to heavy traffic or an abundance of present social data. These sites, by offering platforms for large amounts of user-generated

content, thereby represent very useful data sources for social data mining. Being able to access this previously-unavailable data is a significant step forward, and allows for more extensive mining of user-generated content across the deep web.

6.4.2.3 *Limitations*

The three techniques presented in this chapter all have limitations that would provide a good basis for future work.

The automated interaction algorithm expects certain phrases or design styles being used in the source code of the pages, and does not function well outside these general cases - a more robust method of detecting expansion and redirection links could provide access to significantly more data. For example, locating a set of pagination controls involves looking for elements that use words related to pagination, followed by a search for a certain tag structure indicating a pagination toolbar (e.g. a series of links that consist only of consecutive numbers), followed finally by searches for a dropdown box containing a series of consecutive numbers - all of which would indicate page controls.

The CFH-NS algorithm does not perform well with very loose tag structures, particularly those that rely heavily on inline HTML tags (such as B, P and SPAN tags). It was developed to work with well-designed page structures, which causes performance to suffer on legacy designs. Combination of the algorithm with those that perform text-based context extraction (such as BUW (Thamviset and Wongthanavasu, 2014a)) could improve recall and precision on certain sites.

Finally, the probabilistic datatype determination described in Section 5.4.3 could benefit from the use of more reliable statistical

methods, though the accuracy of the current model performs quite well.

Despite these limitations, the use of these algorithms to improve collection and integration of user-generated content in the SMAAS framework assists in enabling the analysis of social data. The use of this data in real-world scenarios is demonstrated in the following section.

6.5 SCENARIO EVALUATIONS

6.5.1 *Scenario 1: Burst Feature Detection in Generic Social Streams*

A common practical application of analysing social data sources lies in event detection and emergency management. Social media has previously been used in emergency detection, management and response (Sakaki et al., 2010; Cameron et al., 2012; Robinson et al., 2013), using a number of different event detection algorithms (Amruthalingam, 2015; Fung et al., 2005; Ozdikis et al., 2012; Weng and Lee, 2011; Guille and Favre, 2014). Early warnings can make a significant difference to disaster mitigation (de León et al., 2006), and hence there is a need for event detection with the lowest possible latency - requiring optimised frameworks that prioritise real-time event detection.

As presented in Chapter 3, the SMAAS framework is designed to be used in both delayed and real-time scenarios. All events that come in are immediately integrated and passed to the real-time event bus as quickly as possible, allowing for immediate use in event detection algorithms prior to post-processing, which can take additional time.

This prioritisation significantly lowers the latency involved in event detection by entirely bypassing post-processing.

This section presents two evaluations. The first evaluates the improvement in latency using the real-time prioritisation pipeline within the SMAAS framework, in comparison to the standard pipeline that includes post-processing. The second is an implementation of bursty feature detection in integrated social data streams, demonstrating the SMAAS framework's ability to perform event detection generically across multiple social networks in a simple fashion, dramatically improving accessibility of social data.

6.5.1.1 *Real-Time Prioritisation*

Real-time data processing can be a significant challenge when dealing with social data, particularly due the variance in flow rates. Significant public events can cause social media usage to spike dramatically, potentially overwhelming server resources allocated to processing the incoming data. Twitter noted that during some nation-wide events, incoming data rates can spike to 25 times greater than normal (Kirkorian, 2013). Hence, event detection systems operating on social data streams are required to operate even under heavy (and unexpected) load, rendering traditional collect-process-analyse sensor systems vulnerable to significant latency at the times they could be needed most. In this section, the SMAAS framework is evaluated to determine its ability to retain functional real-time processing during periods of unexpected loads.

In this scenario a single node setup was used to drive the SMAAS framework [9], As previously noted, the single-node setup was able to process approximately 36-37 events per second (EPS). In order to

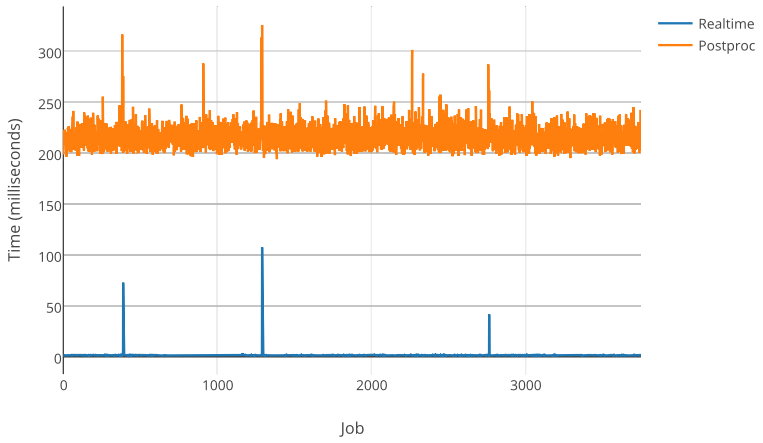
9 The single-node setup consists of an Amazon EC2 M4.4XLarge instance running all required software, including the SMAAS framework, RabbitMQ, ElasticSearch, Redis and the logging server.

facilitate repeatable results, simulated social data was used for the following experiments.

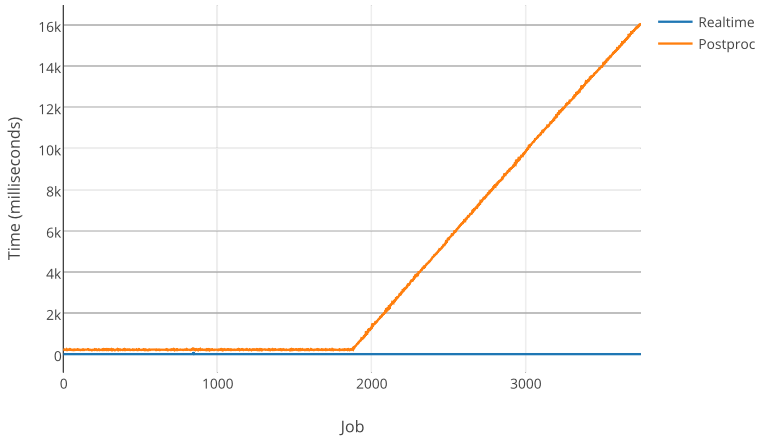
Each of the following tests was operated using the same methodology:

1. Execute realtime processing filters
2. Execute simulate.py script to start pushing events into the bus at a defined rate
3. Log event creation time, and the time taken for the event to reach real-time analysis and serialisation
4. Stop simulate.py script after 150 seconds
5. Wait until post-processing has completed
6. Store, parse and analyse logs for performance data
7. Clear accumulated data and restart all worker processes and filters

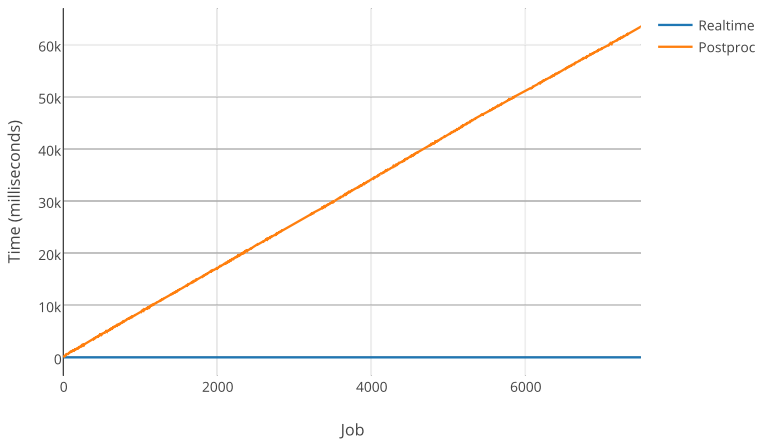
The EPS rate for each test was modified for each test, with every second test featuring a data rate "burst", in which the incoming event rate was increased temporarily. This burst rate was set at 200% - while Twitter noted a 25,000% burst at certain times, such bursts are exceedingly rare and are unlikely to represent a real-world emergency scenario. Emergency events tend to be significantly more localised to the region they occur, while high-burst events tend to be global and visible events (such as the Football World Cup Finals).



(a) Latency comparison at 25 events per second, near the processing limit of available resources.



(b) Latency comparison at 25 events per second, bursting to 200%.



(c) Latency comparison at 50 events per second, beyond the capability of available resources.

Fig. 57a shows that at 25eps, the server is capable of processing incoming data with approximately 33% resources free. This test gives some indication of the overhead involved in post-processing and serialisation while the server is not under load. Even during normal operation, use of the prioritised real-time analysis pipeline results in a 99.28% reduction in latency over the standard post-processed pipeline. While latency is still relatively low for emergency event detection, applications requiring much lower latency to operate can take full advantage of this. One such application is low-latency trading (Hasbrouck and Saar, 2013), which could use the SMAAS framework to take advantage of low-latency social media sensing.

When under burst conditions pushing the server approximately 30% beyond its allocated resources, the serialisation time suffers. Fig. 57b shows the dramatic increase in post-processing latency to over 4 seconds during the burst period, while real-time analytics remains under 2 milliseconds. This level of latency may be acceptable for some emergency scenarios, but excludes its use for any low-latency applications.

Increasing the baseline event rate to 50 EPS causes the post-processing time to increase to 32 seconds, as shown in Fig. 57c. In an emergency scenario, a 30 second early warning can make a significant difference to disaster mitigation (Sakaki et al., 2010).

Under burst conditions at 50 EPS, Fig. 57d shows the serialisation time reach 41.2 seconds, while real-time analysis sits just under 2 milliseconds - a very small increase of latency over the 25 EPS test. These results show that the scalability of the real-time analysis pipeline retains functionality even under heavy load, making it suitable for low-latency sensing applications.

Table 6: Mean time for an event to move from collection to real-time event detection and serialisation.

	Real-Time	Serialisation
25 EPS	1.56ms	216.01ms
25 EPS w/ burst	1.62ms	4222.39ms
50 EPS	1.81ms	32004.26ms
50 EPS w/ burst	1.89ms	41259.3ms

Table 6 summarises the results of the latency tests. As noted, the real-time prioritisation pipeline results in a dramatic decrease in latency for event detection during bursty periods, at the cost of having unprocessed (but fully integrated) data. This ensures that the SMAAS framework is useful for low-latency event detection scenarios, and enables it for use with low-latency applications.

6.5.1.2 *Bursty Feature Analysis*

Social media can be very useful for discovering events in real-time by observing user-generated content, as discussed in Chapter 2. However, while much research mentions applicability to multiple data sources (Xu et al., 2016; Rosser et al., 2017; Amento et al., 2003), very few utilise more than a single source in experiments. To verify that the SMAAS framework can be used in this capacity, a real-time bursty feature detection filter was developed based on the Parameter-Free Bursty Event Detection algorithm (Fung et al., 2005). This algorithm has previously been used in a number of other studies, such as for emergency response (Cameron et al., 2012; Robinson et al., 2013). Both the algorithm and filter are described below, and the ability of the filter to detect events in real-time is evaluated.

Feature Detection

The Parameter-Free Bursty Event Detection algorithm (Fung et al., 2005) detects events by modelling the probability that a given feature appears within a windowed subset of the data. Hence, windows in which the feature is more likely to occur are considered to be "bursting", and signify that the feature represents part of an event. Each feature is a token from event payloads, representing a single word within a social media post that is stemmed or lemmatised [10].

Each feature, f_j is contained within a number of time-interval windows, each denoted as W_i, where the number of documents containing f_j within a window is denoted as $n_{i,j}$. Using a generative probabilistic model, the probability of the number of documents containing f_j in W_i can be computed as $P_g(n_{i,j})$.

$P_g(n_{i,j})$ can be modeled in a computationally efficient manner using a *binomial distribution*, as seen in Equation 2.

$$P_g(n_{i,j}) = \binom{N}{n_{i,j}} p_j^{n_{i,j}} (1-p_j)^{N-n_{i,j}} \tag{2}$$

N is the number of documents in a time window. p_j is the expected probability of the documents that contain the feature f_j in a random time window, and is therefore the average of the observed probability of f_j in all time windows containing f_j, shown in Equation 3 and 4.

$$P_o(n_{i,j}) = \frac{n_{i,j}}{N} \tag{3}$$

10 Stemming is a heuristic process used to truncate the ends of words in an attempt to reduce them to their base form. Lemmatising is a natural language processing technique that uses morphological analysis to more accurately identify the base form of each word, at significant performance expense. Both methods are used in an attempt to group together feature words (so that "bicycle" and "bicycles" are considered to be the same feature).

$$p_j = \frac{1}{L'} \sum_{i=0}^{L'} P_o(n_{i,j}) \tag{4}$$

The distribution is then divided into three main regions across the x-axis (the number of documents): R_A, R_B and R_C. R_A is from 0 to the x value where $P_g(x)$ is the maximum, R_B is from the x value where $P_g(x)$ becomes zero again, and R_C is the region following R_B. For efficiency purposes, the region line dividing R_B and R_C is calculated using SciPy's (Jones et al., 2014) fast binomial interval function, which locates the x values that bound α percent of the distribution - for these purposes, α is defined as 99%.

The position on this distribution $P_g(n_{i,j})$ that $n_{i,j}$ falls defines the probability of whether the word is non-bursty, bursty or a stopword [11]. Hence, the probability of feature f_j bursts in the window W_i is $P_b(i, f_j)$, dependent on the three options below:

- When $n_{i,j}$ is in R_A, it suggests that the probability of feature f_j in W_i is less than or equal to the probability that f_j is drawn randomly. This is considered to be non-bursty, and $P_b(i, f_j) = 0$.
- When $n_{i,j}$ is in R_C, it suggests that f_j exhibits an abnormal behaviour in W_i and is considered bursty, letting $P_b(i, f_j) = 1$.
- When $n_{i,j}$ is in R_B, there are three possibilities. When $n_{i,j}$ approaches the boundary of R_B and R_C, the feature f_j will be bursty; when it approaches the boundary of R_B and R_A, f_j will be non-bursty; and when $n_{i,j}$ is on the mid-point of R_B, f_j can be bursty or non-bursty.

11 A stopword is a frequently-occurring word in a language that is of little significance and not included in indexing operations.

A *sigmoid function* can be used to determine whether f_j is bursty or not when $n_{i,j}$ is in the region R_B, as shown in Equations 5 and 6.

$$x = P_g(n_{i,j}) \cdot \theta - q \tag{5}$$

$$P_b(i, f_j) = \frac{1}{1 + e^{-x}} \tag{6}$$

When $P_b(i, f_j) \geqslant 0.5$, feature f_j is considered to be bursting in window W_i, and both variables are logged for further analysis.

Experiment

For this experiment, the SMAAS framework was run as a large cluster [12] set to generically collect streaming data. The burst detection filter was installed and enabled as part of the low-latency real-time processing pipeline and operated on a dedicated Amazon EC2 C4.XLarge instance. The filter was configured to group features into time windows at an interval of one minute, designed to provide better response time than an hourly window. The bursty feature algorithm was set to run once each time a new window was created - once an entire minute-long window was completed, the algorithm would run over that window using all previous windows as its dataset. The filter is shown in Fig. 58.

12 A large cluster consists of an M4.2XLarge instance running RabbitMQ and Redis, a T2.Medium running the logging server, an M4.XLarge running ElasticSearch, 4x T2.Large instances as collector workers, and both 4x C4.2XLarge instances and 4x M4.2XLarge instances operating as post-processing workers.

```
WINDOWS = collections.defaultdict(lambda:
↪  collections.defaultdict(int))

def process(event):
    global WINDOWS
    # accumulate buckets to date
    bucket = event.time.detected.replace(second=0,
    ↪  microsecond=0)
    # tokenise and grab unique tokens
    for word in set(event.payload.content.split(' '):
        if bucket not in WINDOWS:
            old_bucket =
            ↪  event.time.detected.replace(second=0,
            ↪  microsecond=0) -
            ↪  datetime.timedelta(minutes=1)
            if old_bucket in WINDOWS:
                for word in WINDOWS[old_bucket]:
                    feature_detection(old_bucket, word)
        # automatically create a new bucket
        WINDOWS[bucket][word] += 1
    return event
```

Figure 58: The bursty feature detection filter, based off the algorithm described in (Fung et al., 2005).

Data was collected between 13th October 2015 12:00:00 UTC and 16th October 2015 6:00:00 UTC (approximately 66 hours), covering a number of national-level events. One of the prominent events that occurred within the period was the US Democratic Party presidential debate, on October 13th 2015. The debate involved five candidates; Hillary Clinton, Bernie Sanders, Jim Webb, Martin O'Malley and Lincoln Chafee.

Fig. 59 shows the bursting nature of the candidates and the debate itself, and also illustrates the nature of conversation surrounding the debate. The algorithm grouped burst values per hour, with a value of 60 implying that a feature was bursting for every single minute in an hour window. The stemmed feature "debat" was extremely bursty during the period of the debate, achieving burst values of 58, 60 and 55 for the three hours of the debate, respectively. All candidates

were considered bursty during the duration of the debate, to varying extents - while candidates Clinton and Sanders were bursting for the majority of the debate at values between 40-55 periods per hour, the other candidates sat significantly lower at 20-35 periods per hour. This reflects the tendency of online conversation to favour campaign front-runners in ongoing discussion, while the other candidates were only discussed during periods that they were actively speaking.

While candidates Webb, O'Malley and Chafee are discussed thoroughly during the debate itself, interest on social media fades quickly upon its conclusion. By contrast, candidates Clinton and Sanders both enjoy significant bursty online discussion for the subsequent days, implying that the two candidates are front-runners in the Democratic Presidential race, and enjoy similar levels of discussion online.

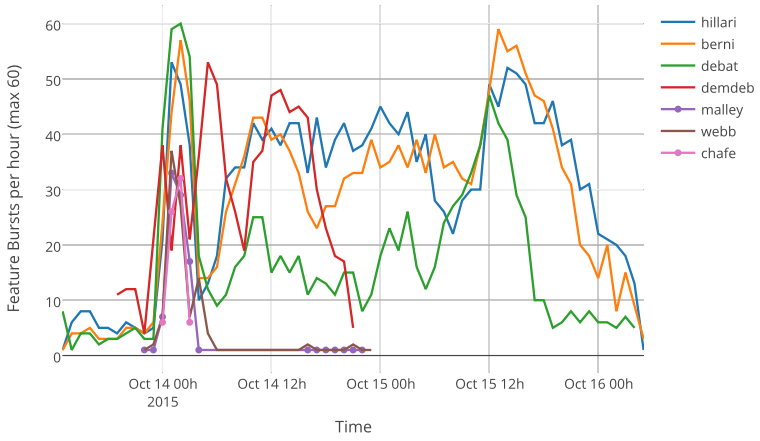


Figure 59: Aggregated bursty feature detection results capture the US democratic debate, clearly showing the significant spike in activity upon commencement of the debate and the ensuing conversation over the subsequent days.

Another significant event detected was the reaction to a media release by Australian Minister for Environment Greg Hunt,

declaring that the controversial Adani Carmichael coal mine had been approved for a second time. Fig. 60 shows the dramatic spike in online conversation related to the minister shortly after the release, wherein the Minister's name bursts between 50-60 periods per hour for the following 3 hours, but conversation quickly fades off as the news cycle progresses.

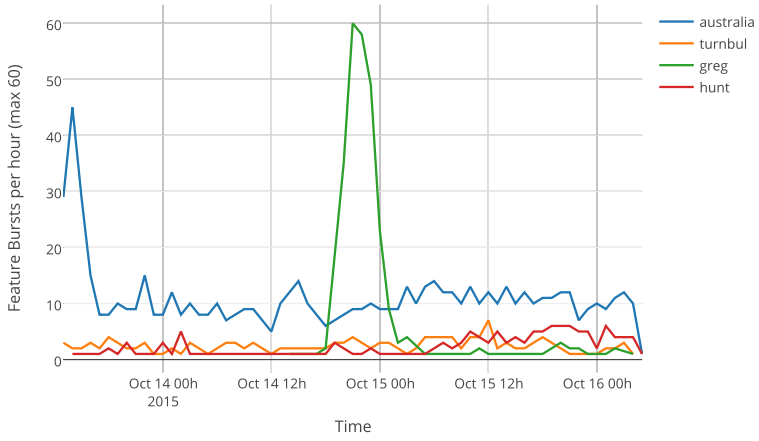


Figure 60: Aggregated bursty feature detection results demonstrating the large spike in activity upon Australian Minister for Environment Greg Hunt's approval of the controversial Carmichael Adani coal mine.

The filter is able to detect a range of different bursty topics, including popular culture events. Fig. 61 pinpoints the moment that former LA Lakers basketball player Lamar Odom was discovered unconscious in a Nevada brothel. The event featured prominently across traditional and social media, in large part due to Odom's former marriage to reality television star Khloe Kardashian. The lower burst values (approximately 20 periods per hour) reflect the lower level of interest compared to the more significant national events seen previously.

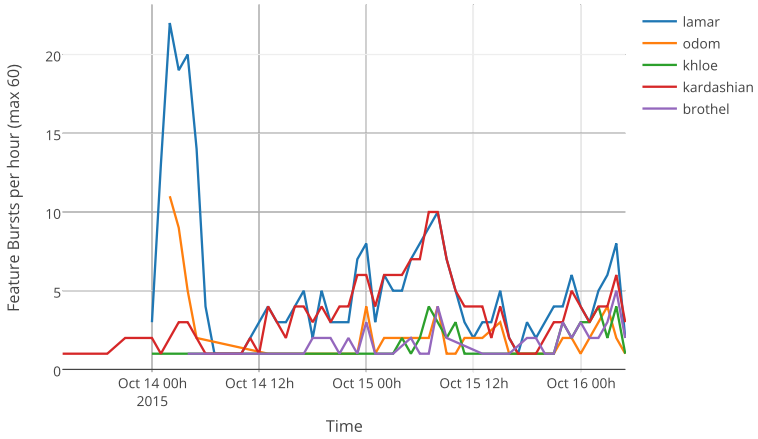


Figure 61: Aggregated bursty feature detection results illustrate the reaction to former basketball player Lamar Odom being discovered unconscious in a brothel in Nevada.

These results show that the SMAAS framework is capable of detecting a range of bursty events across different cultural subdivisions and nations. The selected event detection algorithm operates minute-to-minute in real-time, but the filter shows how easily other algorithms can be used. The filter operates in its own thread, using the real-time event processing pipeline. This means that virtually any algorithm (including very elaborate and complex event-detection frameworks) can be used to enable the SMAAS framework to detect events in real-time by simply cloning the event pipeline and using it to feed additional filters.

6.5.2 *Scenario 2: Political Sentiment Analysis*

The SMAAS framework is designed to allow for collected, integrated and post-processed data to be serialised and analysed in a delayed fashion. This section demonstrates how the SMAAS framework can

be used to collect and process a greater range of generic social data for use in sentiment analysis.

The framework, running on a large cluster as per Section 6.2.7, was set to collect social data covering the US Democratic Party presidential debate, on October 13th 2015. 52,172 relevant social events were collected from multiple social platforms [13] that contain viewer reactions to candidates, topics and general chatter. This number is limited to events that occurred during the debate (between approximately 00:48am to 3:12am UTC) and are related to either a candidate or the debate in general.

The ability of the SMAAS framework to serialise and integrate with any analytical tool dramatically simplifies analysis. To analyse the data in this scenario, the framework serialised all processed data to Elasticsearch (Kuc and Rogozinski, 2013), and Kibana (Elasticsearch BV, 2015a), Python (Python Software Foundation, 1990) and Plotly (Plotly, 2012) were used for analysis. The footage of the debate was manually analysed to determine "points of interest" within the timeline, which could then be cross-analysed with the collected data. For reference, this timeline is available in Appendix 9.

Table 7 shows the results of a "Significant Term" [14] query on the collected data, which shows that the most popular significant terms are candidate names and metatags used to discuss the debate (such as the #demdebate hashtag on Twitter). This query can be used to detect trending terms within a given timeframe in a simplified manner, allowing for more complex follow-up queries.

13 32,772 from Twitter, 19,273 from Reddit and 127 from Facebook. Facebook does not provide a real-time event stream, meaning that streamed collection comes from a number of pre-defined Facebook Pages, so all events on personal pages or pages not in the predefined list were missed.

14 Elasticsearch's significant term query uses a modified version of the TF-IDF (Neto et al., 2000) relevance algorithm called JLH, which adds additional features to scoring such as field length normalisation and query clause boosting. (Elasticsearch BV, 2015b)

Table 7: Most popular significant terms used in social events about the debate.

Term	Number of Events
demdebate	16,857
bernie	11,645
hillary	8,642
sanders	7,199
webb	6,894
debate	6,568
clinton	5,280
chafee	2,925
jim	2,537
o'malley	2,073

Sentiment analysis is used to determine whether the opinions in a piece of text can be categorised as positive, neutral or negative. The SMAAS framework has a sentiment analysis post-processing filter that assesses social event content in this manner. By locating spikes in event activity and referencing the timeframe of these spikes against both the debate topic timeline in the Appendix A and the average sentiment of posts, general viewer opinion towards certain topics can be inferred. This is made more effective by the SMAAS framework's ability to broaden data demographics and collect data from a wider range of sources, lessening any potential bias that could be incurred from using a single social data source.

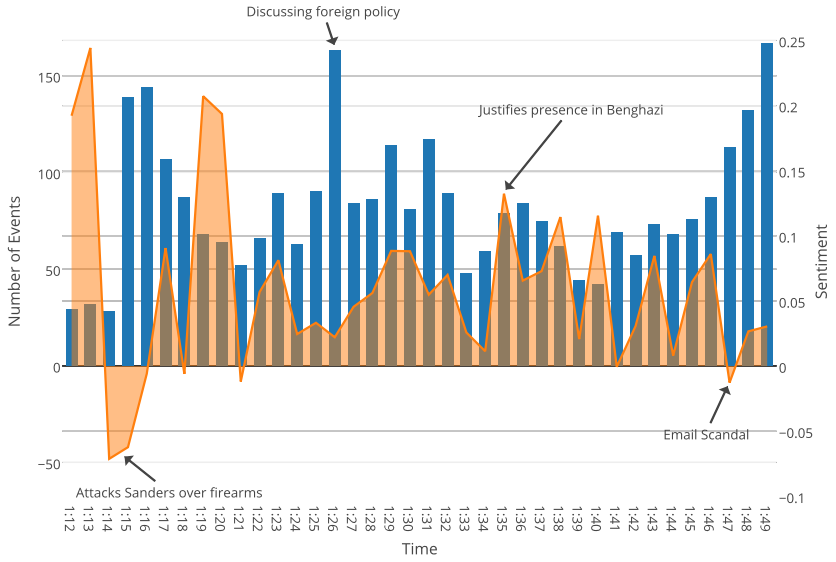


Figure 62: Aggregated event count and associated sentiment related to Hillary Clinton during the debate, per minute.

Fig. 62 shows the number of events collected and average sentiment related to Hillary Clinton, grouped per minute. This allows us to see spikes in activity, which can then be cross-referenced against the debate topic timeline.

At 1:15-1:17am, Clinton attacks Sanders in relation to his history on gun control legislation, causing a significant spike in traffic for Clinton. While activity increases, sentiment dramatically decreases - indicating that many observers take issue to the line of attack and support Sanders. While discussing foreign policy at 1:22-1:35am (historically a strong point for Clinton, as the Secretary of State) she sees increased activity and sentiment as viewers respond to her experience in the area. Discussion of the Clinton email scandal (Hartmann, 2015) results in a small drop in sentiment and a large increase in activity for both Clinton and Sanders.

The ability of the SMAAS framework to provide high-quality, timely data allows for easy synchronisation of data to the debate

timeline, enabling simple identification of popular topics and associated sentiment.

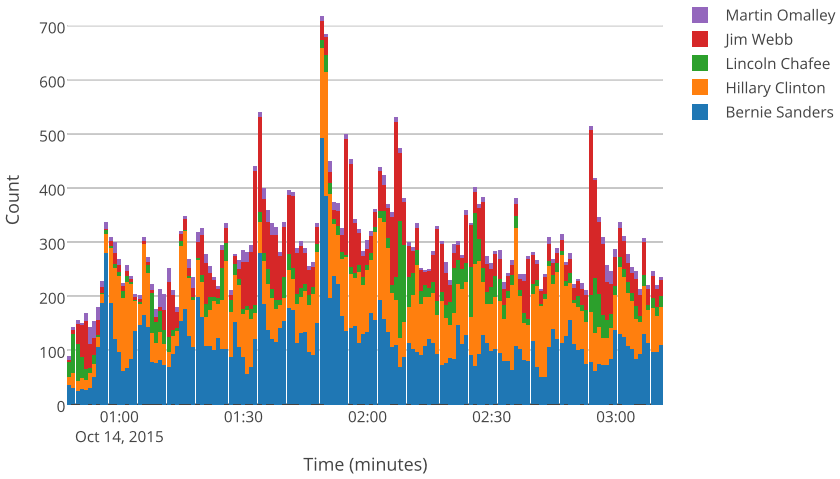


Figure 63: Count of events for each candidate, per minute.

Similar patterns are observed for Sanders. A noted highlight of the debate (Swanson, 2015) is seen at 1:48am, with both Sanders and Clinton's activity dramatically spiking (seen in Fig. 63), marking the moment Sanders said "I think the American people are sick and tired of hearing about your damned emails," to a standing ovation and handshake between the candidates. These spikes in activity and sentiment can be automatically detected using burst detection filters similar to those described in the previous section, facilitated by the SMAAS framework. This would allow for real-time feedback on discussion topics during events such as these, even allowing candidates to receive feedback and tailor their performances in real-time.

As demonstrated, the SMAAS framework is able to collect, integrate, serialise and present data in formats able to be integrated into specialised analytics tools. This allows us to perform useful

analysis related to current events, and can illustrate online sentiment towards topics - a process useful for a range of fields, including political analysis and marketing. This also enables studies that would otherwise operate on a single platform (due to integration difficulties) to instead use data from multiple platforms, vastly expanding demographics and alleviating geographical and cultural barriers to analytics.

6.6 CONCLUSION

This chapter presented an evaluation of the SMAAS framework against defined requirements and in real-world scenarios to determine practical usability. Firstly, it was determined that each of the requirements in Chapter 3 was met. More detailed evaluations were presented for the intelligent sourcing techniques presented in Chapter 4 and the user-generated content extraction algorithms in Chapter 5. Finally, the framework was used in two real-world scenarios to detect events across integrated social streams and also perform more detailed analytics for a specific event.

The SMAAS framework was used several times to collect large amounts of social data, integrate it and serialise it for later presentation. Running on 15 Amazon EC2 instances of various types, it collected social data from 5 heterogeneous data sources and performed entity detection, gender and sentiment analysis across 23 million events, serialising the results to ElasticSearch. During this process, various benchmark data was collected - from the latency of events passing the real-time and serialisation marks, to how the cluster handled scalable workloads.

The results indicate that the SMAAS framework is useful for a range of potential applications, with low latency during processing enabling its use in low-latency trading algorithms, to emergency crisis management and political analytics using sentiment analysis. The framework is scalable to cope with very high data throughput while also retaining the ability to run as a small single-instance setup for bespoke applications and analysis. It is able to extensibly handle data inputs from a range of sources (with seven data interfaces provided) and perform any type of post-processing - with sentiment, gender and entity analysis provided. Finally, it is able to source, collect, integrate, clean, serialise, query and present social data from disparate sources in a generic manner, greatly easing the accessibility of social data as a broader medium.

The next chapter summarises the contributions of the thesis to the field of data integration, mining and sensor networking and concludes.

7

CONCLUSION

7.1 OVERVIEW

This chapter concludes the research into social sensing (Ali et al., 2011) described in this thesis. Social sensing attempts to repurpose data already created and made available by social media users for use in sensor networking. This has a wide range of possible applications in environmental monitoring, event detection, disaster mitigation, urban planning and more, and is able to act as a sole data source or augment bespoke deployments.

Accessing, collecting and integrating social data for use in sensor networking can be extremely difficult, considering the sheer scale of data. The work described in this thesis aims to alleviate these issues and significantly ease the accessibility of social data for use in sensor networking. This PhD thesis has made several contributions to the field of social data mining and sensing by developing a framework to ease the sourcing, collection, integration, cleaning and serialisation of social data, and developing new algorithms for efficient social data sourcing and user-generated context extraction.

The following sections summarise the main research contributions and how the main research aims of this thesis were met, followed by a discussion of the limitations of the research. Finally, the implications of the findings are discussed.

7.2 DEVELOPMENT OF THE SMAAS FRAMEWORK

Social media is a plentiful source of relevant real-time data that can be used for sensor networking. However, while Social Networking Sites (SNS) often provide APIs to enable data access, standard web pages containing user-generated content have no accessible interface for data collection. Even for sites that provide APIs, integration of disparate social data sources is a complex task. Commercial systems exist that provide this service (Gnip, 2008; DataSift, 2010), but operate as large Extract-Transform-Load (ETL) warehouses that often do not provide processed data in real-time. This is primarily due to the scale of social data being created, and the level of processing resources needed to collect, integrate and clean the data in real-time, particularly during bursty periods. As such, many of these systems cannot be used for low-latency real-time sensing applications. Additionally, the services that provide social data are can be prohibitively expensive depending on the type and amount of data being accessed and are unable to be modified to suit bespoke usages.

In Chapter 3, a set of functionality requirements was defined that aim to solve these problems. The Social Media As A Sensor (SMAAS) framework was then proposed as a design to meet those requirements. The framework supports extensible sourcing, collection, post-processing, enrichment, cleaning and integration, with each component replaceable. It divides real-time and delayed processing into individual pipelines, enabling low-latency applications to receive events as soon as they are integrated rather than having to wait for them to undergo post-processing. The

framework is scalable to all levels of hardware and can automatically scale up and down if required.

In Chapter 6, it is demonstrated that it also improves collection efficiency by implementing the intelligent sourcing algorithm described in Chapter 4, enabling it to handle a higher throughput of data without sacrificing a significant amount of accuracy. It improves genericity through the use of the CFH-NS user-generated content extraction algorithm described in Chapter 5, which enables the unsupervised collection of social data from identified sources. It leaves serialisation, querying and presentation to existing data management platforms (such as RDBMS or Document Storage Engines) by providing an extensible output interface, which also allows for integration with existing sensor networking implementations and data analysis suites.

The SMAAS framework was empirically evaluated in Chapter 6 to determine its conformity to the defined requirements. It was determined that the framework is able to source, collect, integrate, clean and serialise social data from generic sources extensibly. It provides the ability to perform low-latency analysis even in high-stress and bursty environments. The framework was then used in two real-world demonstrations, when it was determined that the framework is useful for real-time event detection in the manner of a sensor network, and delayed analysis. It could also be used in further research to evaluation new techniques for social data mining and analytics, such as new data extraction and analysis algorithms, in a repeatable manner operating over disparate data sources. This allows research to be completed with reduced demographic and cultural associated with the use of a single social platform.

7.3 INTELLIGENT SOURCING

Social media is a plentiful source of data for sensor networking and analysis, with an enormous amount of data being produced daily. However, integrating, processing and analysing this data presents a number of challenges, primarily due to scale. To alleviate these problems, it is desirable to filter the initial set of data as much as possible to remove irrelevant data and reduce the amount of unnecessary integration work performed.

While a substantive amount of research has been performed into using query expansion for improving precision in social data searches (Lau et al., 2011; Massoudi et al., 2011; Bandyopadhyay et al., 2012), the primary focus of the previous research is to improve both recall and precision for social searches - specifically, microblog (e.g. Twitter) searches. However, using social data as a basis for sensor networking requires that the incoming dataset be quite broad, rather than extremely precise - there's little point in improving precision for a search that is intentionally wide. Hence, some method of expanding recall at the expense of precision was necessary.

This filter was developed as an intelligent social data sourcing module for the SMAAS framework, detailed in Chapter 4. The intelligent sourcing algorithm uses natural language processing techniques (such as part-of-speech tagging (Ratnaparkhi and others, 1996)) to iteratively train search queries in order to find additional relevant and semi-relevant social data. The algorithm intentionally sacrifices precision for a significant expansion in recall, to provide a useful base for sensor networking while still reducing the incoming dataset. Once queries have been formed, the SMAAS framework

platform APIs described in Chapter 3 are used to execute the queries and collect resulting data.

The efficacy of the intelligent sourcing algorithm was evaluated in Chapter 6, which determined that it significantly increased the amount of relevant social data collected over a simple search, leading to an increase in recall and associated drop in precision. It also significantly reduced the signal-to-noise ratio of the collect-all approach, providing a marked reduction in the amount of social data requiring collection, integration and post-processing. This opens up new applications in social sensing, as it significantly reduces the amount of processing resources required.

7.4 USER-GENERATED CONTENT EXTRACTION

Social data is currently collected using a number of different methods. Typical collection involves locating a source that provides an API and writing a wrapper to interface with the API. However, the API for nearly every site is different and return results in different formats, requiring manual wrapper development. In addition, many sites containing social data (such as news sites and blogs) do not provide APIs to access user-generated content featured on the site. This data is inaccessible unless an interface wrapper is written, which requires the development of a new wrapper for every individual site being used as a source. This is generally infeasible unless only a small number of sources are being used. To counter these problems, unsupervised data extraction algorithms have been developed, but none collect social data or user-generated content (Liu et al., 2003, 2010; Ferrara et al., 2014; Kao and Chen, 2010; Xia,

2009; Arasu and Garcia-Molina, 2003; Muslea et al., 1999; Cao et al., 2008).

Chapter 5 presented a new unsupervised user-generated content extraction algorithm: Canopied Feature Hashing with Nested Structure Detection (CFH-NS). This algorithm is designed to intelligently access user-generated content on web pages in an unsupervised manner. It also handles pages that dynamically inject content using AJAX by expanding methods developed by previous research (Mesbah et al., 2012; Baumgartner and Ledermiiller, 2005; Xia, 2009), which make up an increasing proportion of pages on the Internet. It uses heuristics to locate repeating structures that contain user-generated content and extracts the data from them before integrating the result into a common format. It also contains heuristic-based methods for automatically navigating and interacting with pages in order to expose the greatest possible amount of social data, previously hidden behind pagination systems.

The user-generated content extraction algorithm is empirically evaluated in Chapter 6. A new testbed is developed that is more consistent with current Internet design and development standards. The CFH-NS algorithm is determined to provide best-in-class performance for extracting user-generated content from generic web pages. The automated interaction component significant improves access to additional data, gathering more data per source than previously available. Additionally, the dynamic rendering components used in the algorithm allow data to be collected from modern pages that require client-side scripting to be enabled, which is rarely implemented in web scraping solutions.

The CFH-NS algorithm provides a consistent interface to generic web pages, regardless of their design or the format of the user-generated content. No manual wrapper development is required, nor any regular maintenance when page designs change. This means that access of social data on these pages is through a single interface. The SMAAS framework implements this algorithm as a collector, which can then be used to collect data from any web-based sources identified by the intelligent sourcing algorithm described in the previous section without any need for additional code.

7.5 RESEARCH AIMS

In Section 1.3, a number of research aims are introduced that outline a high-level set of goals that this thesis aims to address. Each of these aims is addressed below:

AIM 1: **Develop and evaluate methods of efficiently collecting and integrating generic social data for analysis.**

In Chapter 4, an intelligent sourcing algorithm is described that aims to reduce the entry requirements for large-scale social data mining by isolating relevant data sources, rather than adhering to a collect-all approach. In Section 6.3, this algorithm is evaluated and determined to provide a large amount of relevant data while significantly reducing noise, satisfying this research aim to a high level of success.

AIM 2: **Develop new techniques to access alternative and untapped social data sources for use in social data mining.**

In Chapter 5, the CFH-NS algorithm is described, which is designed to automatically and without supervision develop data extraction wrappers that can extract social data from previously-unavailable sources. In Section 6.4, the performance of CFH-NS is evaluated against several current state-of-the-art algorithms and is determined to be a significant improvement for the extraction of social data, satisfying this research aim to a high level of success.

AIM 3: **Develop and evaluate a framework that integrates work from the previous two aims to provide an extensive and generic social data sourcing, collection and querying framework.**

In Chapter 3, the SMAAS framework is described, which aims to provide an extensive toolkit for social data mining. It allows the construction of a pipeline taking social data from sourcing and collection through cleaning and integration and into presentation. By combining both intelligent sourcing and the CFH-NS algorithm, cross-platform social data mining can be quickly and easily performed for academic and commercial applications with a high degree of flexibility. In Section 6.5, the SMAAS framework is used to perform social sensing in two real-world scenarios and is highly successful, satisfying this research aim to a high level of success.

7.6 LIMITATIONS AND FUTURE RESEARCH

The SMAAS framework was designed to be fully extensible, and acts as both a practical platform for operating social analytics as well as a testbed for evaluating algorithms in each extensible space, such as sourcing, integration or event detection experiments. Future research can be put into each of these challenges, examining better ways to source data, generically collect it, automatically integrate it or perform analysis upon it.

The intelligent sourcing algorithm could be expanded to more fully direct searches across different networks, rather than relying on search interfaces on individual networks. While the SMAAS framework provides search functionality to most platforms, the users of each platform may use slightly different terminology or metatag formats, so using a singular search query across all interfaces may not produce an optimal result. Additionally, more research effort could be put into identifying relevant data within collected content, rather than analysing the entire set of content retrieved - this could significantly reduce noise.

The user-generated content extraction algorithm could be improved to increase precision and recall, and the automated interaction algorithm could definitely be improved through more complex methods of locating redirection and expansion links. Additionally, the type discovery algorithm used within CFH-NS could use more complex and accurate probabilistic models to determine field types. This algorithm, while highly experimental, was successful at extracting UGC from most page designs - but had

limited success operating on pages that heavily relied on in-line HTML tags [1] or used simple DIV elements for all page structures.

There would be significant value in the automatic generation of social graphs from social events collected, especially in visualisation. Academic applications spend quite some time creating basic visualisations to illustrate links between social events. This has the additional benefit of being a solid analytical base to perform machine learning upon; by using ML to cluster users, can we identify users more likely to respond in certain ways to certain events? This has large implications for some applications, particularly in health/mental health, and is an exciting prospective research field.

The SMAAS framework itself also has room for improvement - auto-scaling of workers would significantly reduce cost while not affecting search/collection efficiency. Likewise, the interface to the system could be improved and integrate with existing analytical tools, which could provide significant business value.

7.7 RESEARCH IMPLICATIONS

The development of the SMAAS framework and of the user-generated content extraction algorithm CFH-NS both have significant implications for the research in the fields of social data mining and social sensing, as well as social data extraction.

In Section 1.1, it was stated that only 4% of papers citing three well-known social event detection papers used more than a single SNS in their experiments. These studies that currently operate on a single platform to alleviate collection and integration difficulty are

1 Such as paragraph (<p>), span () and line-break (</br>) tags.

now able to use the SMAAS framework and abstract away the difference between social platforms. This aids in reducing bias between platform demographics and also opens up the possibility of cross-platform research - e.g. studies that are designed to merge social graphs across networks.

The SMAAS framework also allows for new applications in sensor networking backed by social data sources without bespoke deployments, reducing the barrier to entry for new research into social sensing. The framework eliminates any effort required by researchers to find, collect and integrate social data, and provides access to previously-inaccessible social data sources such as comment sections from news websites and blogs.

There are additional research opportunities that this thesis makes available, particularly in automated interaction, social data extraction and query expansion. Section 6.4.2.2 notes that 20% of sites tested had additional social data made inaccessible without the use of automated interaction techniques. There are also further improvements to be made to social data extraction in both recall and precision, and the query expansion algorithms can be improved to provide additional relevant data.

7.8 PRACTICAL IMPLICATIONS

On a practical level, the SMAAS framework can be used to perform industrial analysis of social data for marketing, sentiment analysis or event detection. This does not break new ground - solutions exist already that are able to do so. However, the intelligent sourcing and CFH-NS algorithms are both entirely new work that do not currently exist in systems, and both provide substantial new data sources to

work with over existing solutions. This expands the reach of social analytics into sources of social data not currently accessible, which opens up new possibilities in real-time analysis and social sensing.

In conclusion, the research described in this thesis represents significant progress towards making social data more accessible and thereby opening new opportunities for the use of social sensing and its integration into society. Social sensing can harness data already being produced by everyday people to investigate, describe and improve many aspects of our environment, lifestyle and infrastructure.

Part I

APPENDICES

A

APPENDIX A

A.1 USER-GENERATED CONTEXT EXTRACTION RESULTS

Table 8: Experimental results.

URL	AR-P	CFH	D	AR	D	B-L
9to5mac.com/2...	20	13/13	10/8	20	0/0	6/6
bbs.boingboin...	18	18/18	6/6	18	5/2	0/0
clubtroppo.co...	30	30/30	25/10	30	0/0	19/11
games.on.net/...	13	13/13	13/13	13	13/13	13/13
homeservershо...	60	55/55	21/21	60	0/0	44/34
larvatusprode...	220	240/220	27/27	220	0/0	220/220
news.national...	50	51/50	15/13	50	0/0	0/0
newsfeed.gawk...	42	82/42	0/0	20	0/0	0/0
packetlife.ne...	18	12/12	10/6	18	10/6	0/0
pivotallabs.c...	5	5/5	16/4	5	16/4	5/5
submicron.dev...	8	8/8	4/4	8	4/4	0/0
techcrunch.co...	18	10/10	0/0	18	0/0	0/0
theconversati...	115	115/115	52/46	115	0/0	111/111
the-toast.net...	41	26/26	0/0	27	0/0	5/5
www.adelaiden...	101	57/52	0/0	50	0/0	0/0
www.amazon.co...	3	3/3	3/3	3	3/3	3/3
www.autocar.c...	7	7/7	6/6	7	6/6	7/6
www.canonrumo...	15	24/15	5/5	15	5/5	26/14
www.courierma...	45	45/45	0/0	45	0/0	0/0

Continued on next page

Table 8 – *Continued from previous page*

URL	AR-P	CFH	D	AR	D-L	B-L
www.cracked.c...	592	543/543	123/123	34	0/0	0/0
www.crymore.n...	78	109/70	55/23	78	0/0	26/12
www.dailytele...	72	54/54	0/0	50	0/0	0/0
www.destructo...	219	220/219	72/72	50	0/0	0/0
www.dpreview....	193	205/193	72/72	176	0/0	0/0
www.engadget....	4	4/4	0/0	4	0/0	0/0
www.escapistm...	5	10/5	0/0	5	0/0	0/0
www.gamespot....	24	24/24	9/9	24	0/0	24/0
www.gizmodo.c...	10	10/10	3/3	10	0/0	9/8
www.joystiq.c...	45	51/45	18/15	*		
www.kotaku.co...	23	23/23	5/5	23	0/0	13/8
www.layer9.or...	23	23/23	0/0	23	0/0	0/0
www.macrumors...	10	10/10	0/0	10	0/0	10/10
www.macworld....	36	44/36	8/8	36	0/0	0/0
www.mortgageb...	2	4/2	0/0	2	0/0	0/0
www.neatorama...	7	7/7	10/5	7	10/5	5/5
www.pcworld.c...	12	20/12	4/4	12	0/0	9/7
www.rocketthe...	10	20/10	5/5	10	5/5	10/10
www.rockpaper...	64	64/64	16/16	64	16/16	43/18
www.smh.com.a...	47	47/47	25/25	10	2/2	9/9
www.theage.co...	545	545/545	138/138	10	3/3	9/9
www.tripadvis...	20	20/20	0/0	10	0/0	10/10
www.urbanspoo...	26	22/22	6/6	26	6/6	0/0
www.windowsce...	27	64/27	8/8	27	8/8	18/9
gigaom.com/2...	9	17/9	2/2	9	0/0	0/0
productforum...	21	42/21	3/3	21	0/0	0/0
vimeo.com/98...	17	17/17	3/3	17	0/0	0/0
www.indiegog...	5	5/5	12/5	5	0/0	0/0

Continued on next page

Table 8 – *Continued from previous page*

URL	AR-P	CFH	D	AR	D-L	B-L
www.kickstar...	50	45/45	25/25	50	25/25	50/50
www.ozbargai...	44	44/44	12/11	44	12/11	0/0
www.theguardi...	86	107/86	6/6	86	0/0	0/0

A.2 ADDITIONAL SCALABILITY GRAPHS

Time Tasks Spent in Queue

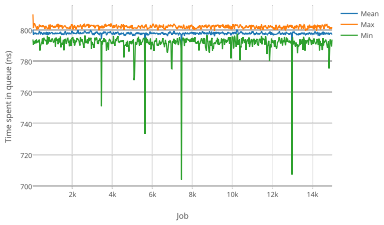


Figure 64: Single EC2 M4 4XLarge, 25eps

Time Tasks Spent Processing

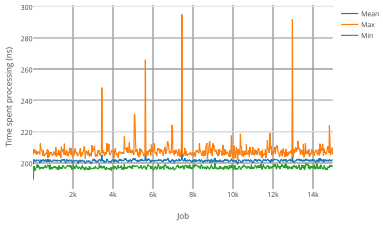


Figure 65: Single EC2 M4 4XLarge, 25eps

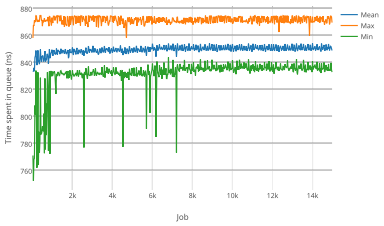


Figure 66: 4x EC2 C4 2XLarge, 25eps

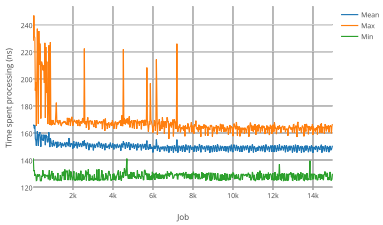


Figure 67: 4x EC2 C4 2XLarge, 25eps

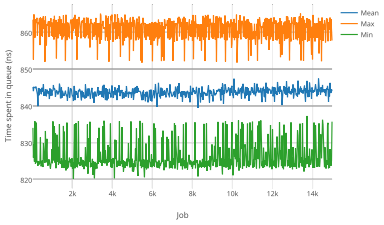


Figure 68: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 25eps

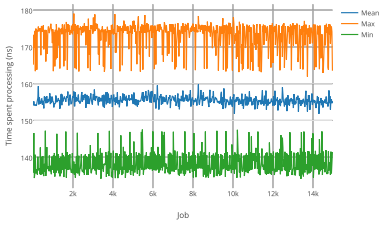


Figure 69: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 25eps

Time Tasks Spent in Queue

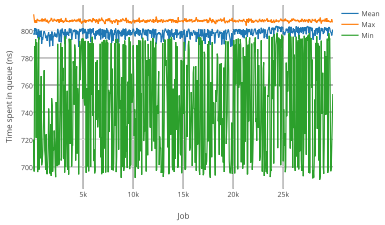


Figure 70: Single EC2 M4 4XLarge, 50eps

Time Tasks Spent Processing

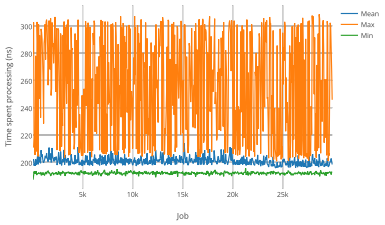


Figure 71: Single EC2 M4 4XLarge, 50eps

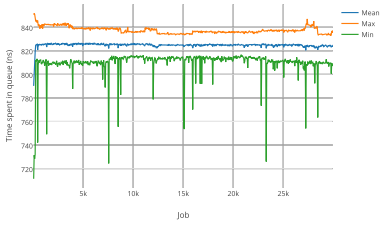


Figure 72: 4x EC2 C4 2XLarge, 50eps

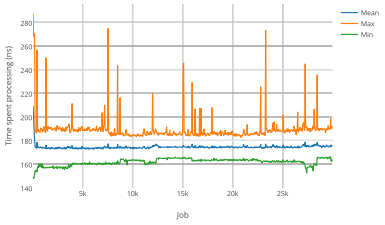


Figure 73: 4x EC2 C4 2XLarge, 50eps

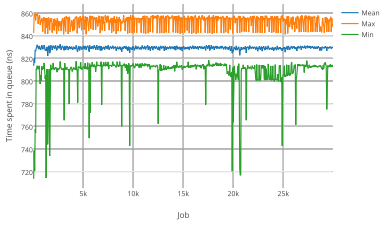


Figure 74: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 50eps

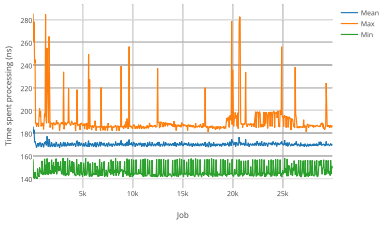


Figure 75: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 50eps

Time Tasks Spent in Queue

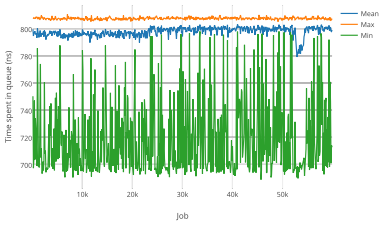


Figure 76: Single EC2 M4 4XLarge, 100eps

Time Tasks Spent Processing

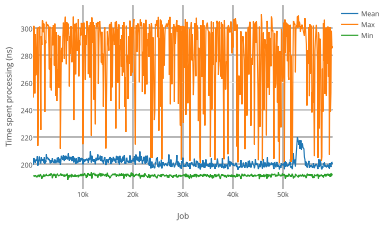


Figure 77: Single EC2 M4 4XLarge, 100eps

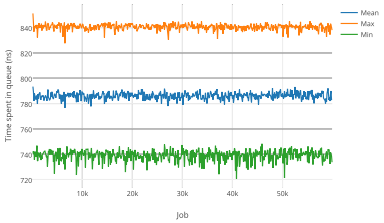


Figure 78: 4x EC2 C4 2XLarge, 100eps

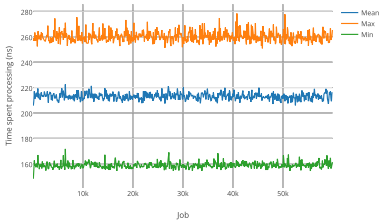


Figure 79: 4x EC2 C4 2XLarge, 100eps

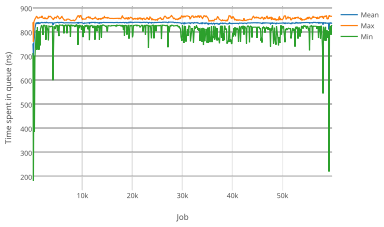


Figure 80: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 100eps

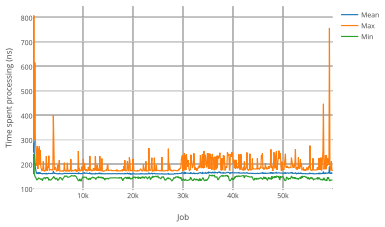


Figure 81: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 100eps

Time Tasks Spent in Queue

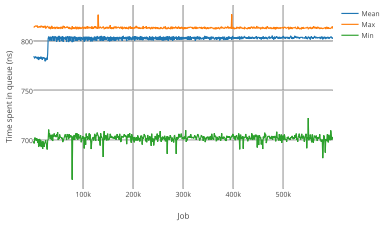


Figure 82: Single EC2 M4 4XLarge, 1000eps

Time Tasks Spent Processing

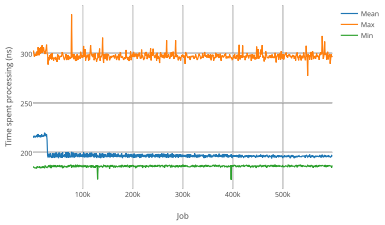


Figure 83: Single EC2 M4 4XLarge, 1000eps

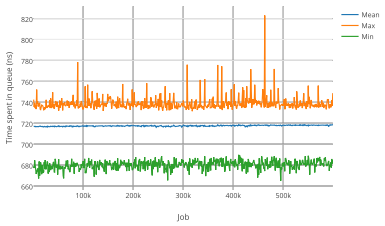


Figure 84: 4x EC2 C4 2XLarge, 1000eps

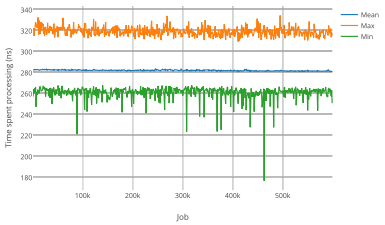


Figure 85: 4x EC2 C4 2XLarge, 1000eps

Figure 86: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 1000eps

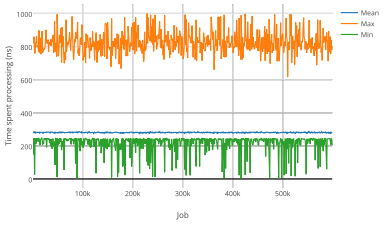


Figure 87: 4x EC2 C4 2XLarge and 4x M4 2XLarge, 1000eps

A.3 CFH-NS TESTBED URLS

1. http://9to5mac.com/2015/03/08/apple-campus-2-drone-flyover/
2. http://bbs.boingboing.net/t/act-now-congress-wants-to-fast-track-the-trans-pacific-partnership/52776
3. http://clubtroppo.com.au/2015/02/12/stem-part-culture-war-part-cargo-cult-my-latest-fin-column/
4. http://games.on.net/2015/03/league-of-legends-ai-toxic-players/
5. http://homeservershow.com/hp-proliant-n40l-microserver-build-and-bios-modification-revisited.html
6. http://larvatusprodeo.net/archives/2011/01/brisbane-flood-maps-and-up-to-date-flood-information/
7. http://news.nationalgeographic.com/news/2015/03/150302-black-hole-blast-biggest-science-galaxies-space/
8. http://newsfeed.gawker.com/terror-owl-gleefully-tearing-apart-its-victims-in-the-1688525383/+jparham
9. http://packetlife.net/blog/2014/nov/18/mac-address-aggregation-and-translation/
10. http://pivotallabs.com/rspec-elasticsearchruby-elasticsearchmodel/
11. http://submicron.deviantart.com/art/Soulsequencer-Series-Hemocyte-Green-481438108?q=gallery%3Asubmicron%2F23534542&qo=8
12. http://techcrunch.com/2014/05/15/soldsie-the-service-that-lets-you-shop-via-facebook-and-instagram-comments-raises-4-million/
13. http://theconversation.com/we-are-all-suspects-now-thanks-to-australias-data-retention-plans-38223
14. http://the-toast.net/2015/02/10/gal-science-ant-lab/
15. http://www.adelaidenow.com.au/news/south-australia/sa-public-servant-paid-out-to-quit-at-age-of-76/story-fni6uo1m-1227255760836
16. http://www.amazon.com/dp/B00I15SB16
17. http://www.autocar.co.uk/car-news/detroit-motor-show/next-mercedes-benz-glk-spotted-ahead-2015-launch
18. http://www.canonrumors.com/forum/index.php?topic=25434.0

19. http://www.couriermail.com.au/news/mh370-disappearance-towelette-that-washed-up-on-wa-beach-being-tested-for-connection-to-missing-malaysia-airlines-plane/story-fnii5s41-1227256204320
20. http://www.cracked.com/article_22116_6-popular-games-that-were-meant-to-be-totally-different.html?wa_user1=2&wa_user2=Weird+World&wa_user3=article&wa_user4=feature_module
21. http://www.crymore.net/2015/02/06/the-4-best-animes-youre-missing-out-on-winter-2015-edition/
22. http://www.dailytelegraph.com.au/news/national/toxic-tuna-daily-telegraph-investigation-reveals-the-smelly-and-messy-conditions-at-thai-tuna-factories-linked-to-poisoned-fish/story-fnpn0zn5-1227255781943
23. http://www.destructoid.com/police-officer-killed-during-gamestop-robbery-288794.phtml
24. http://www.dpreview.com/reviews/sony-alpha-7-s?utm_campaign=internal-link&utm_source=features-default&utm_medium=homepage-block&ref=features-default
25. http://www.engadget.com/2015/03/09/apple-macbook/
26. http://www.escapistmagazine.com/articles/view/video-games/columns/extra-punctuation/13550-When-Remastering-Classic-Games-Stay-True-to-the-Originals
27. http://www.gamespot.com/videos/sid-meier-s-starships-ori-and-the-blind-forest-hot/2300-6423773/
28. http://www.gizmodo.com.au/2014/09/kindle-voyage-this-is-what-a-200-e-reader-looks-like-its-gorgeous/
29. http://www.joystiq.com/2015/02/03/there-is-no-end/
30. http://www.kotaku.com.au/2015/03/remember-this-735/
31. http://www.layer9.org/2014/10/session-3-paper-1-reclaiming-brain.html
32. http://www.macrumors.com/2015/03/06/hands-on-with-the-textblade-keyboard/
33. http://www.macworld.com/article/2894575/apples-radical-12-inch-macbook-air-is-the-slimmest-lightest-mac-ever.html

34. http://www.mortgagebusiness.com.au/breaking-news/8218-frydenberg-concerned-about-smsf-lending?utm_source=Mortgage+Business&utm_campaign=Mortgage_Business_Newsletter02_03_2015&utm_medium=email

35. http://www.neatorama.com/2015/02/27/The-Girl-Who-Gets-Gifts-from-Crows/

36. http://www.pcworld.com/article/2893647/gnome-2-is-back-ubuntu-mate-is-now-an-official-flavor.html

37. http://www.rockettheme.com/forum/general-discussion/147147-page-with-modules-only-no-mainbody-apart-from-home-page

38. http://www.rockpapershotgun.com/2015/02/26/rimworld-alpha-review/

39. http://www.smh.com.au/federal-politics/political-opinion/inequality-at-the-heart-of-rejection-of-gonski-program-20131130-2yi54.html

40. http://www.theage.com.au/federal-politics/political-news/asylum-seeker-torture-report-united-nations-special-rapporteur-juan-mendez-responds-to-tony-abbott-criticism-20150310-13zrwz.html

41. http://www.tripadvisor.com.au/Restaurant_Review-g1076168-d3418528-Reviews-Blend_Cafe_and_Pizza_Bar-Melville_Greater_Perth_Western_Australia.html

42. http://www.urbanspoon.com/r/338/1430837/restaurant/Perth/Blend-Cafe-and-Pizza-Bar-Melville

43. http://www.windowscentral.com/gdc-2015-quick-look-siegecraft-commander-coming-xbox-one-summer

44. https://gigaom.com/2015/03/08/how-i-manage-my-creative-process-on-an-11-inch-macbook-air/

45. https://productforums.google.com/forum/#!category-topic/hangouts/Xfh-RfpGBP4%5B1-25%5D

46. https://vimeo.com/98201214

47. https://www.indiegogo.com/projects/rocketbook-cloud-integrated-microwavable-notebook#comments

48. https://www.kickstarter.com/projects/1853707494/pancakebot-the-worlds-first-pancake-printer/comments

49. https://www.ozbargain.com.au/node/185655

50. http://www.theguardian.com/australia-news/2015/mar/15/crossbenchers-rebuff-pyne-ultimatum-on-university-reforms-and-research

A.4 DEBATE TOPIC TIMELINE

Table 9: US Democratic Party presidential debate timeline of issues, 13th October 2015. Times in UTC.

Time	Candidate	Topic
0:48:00	Chafee	Opening statements.
0:50:00	Webb	Opening statements.
0:52:30	O'Malley	Opening statements.
0:55:00	Sanders	Opening statements.
0:57:30	Clinton	Opening statements.
1:00:30	Clinton	Flip-Flop, Expediency, "Stand my ground!"
1:03:00	Sanders	"Socialist in the White House", How can you win?
1:05:45	Clinton	Follow-up: Is anyone else here not a capitalist? Save capitalism from itself.
1:06:45	Sanders	Re-follow-up: "Entrepeneurial nation!", "Growth doesn't mean anything if it's inequal."
1:07:15	Chafee	"You've been a republican, etc, why should voters trust that you won't change again?"
1:08:30	O'Malley	"Baltimore blames your zero-tolerance policy for the riots."
1:11:15	Webb	Affirmative Action, "Aren't you out of step?"
1:12:45	Sanders	Guns. You supported gun owners, but now say otherwise? Do you want to shield gun companies from prosecution?
1:14:45	Clinton	Follow-up: Is Sanders tough enough on guns? "No." Attack on Sanders by Clinton.

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
1:15:45	Sanders	Re-follow-up: "All the shouting in the world isn't going to keep guns out of the hands of people that shouldn't have them."
1:16:30	O'Malley	Obama couldn't pass gun laws, how did you? "We didn't look at polls."
1:18:00	Sanders	Follow-up: "I think O'Malley gave a very good example of weaknesses in the laws, but rural and urban disagree on this issue."
1:18:45	O'Malley	Re-follow-up: "It's not about rural vs urban."
1:19:00	Sanders	Re-re-follow-up: "You haven't been in congress!"
1:19:15	Webb	You had an A rating from the NRA.
1:20:45	Chafee	You have an F rating from the NRA. "Gun lobby!"
1:21:30	O'Malley	"I wrote back to the NRA! They didn't dare petition it to referendum."
1:22:00	Clinton	Russia. Foreign policy. Did you underestimate them? "Stop bullying us and others. We need to take leadership."
1:23:15	Sanders	"Syria? Quagmire."
1:24:30	Chafee	Only republican in senate to vote against Iraq. Clinton said her support was a mistake - why isn't that enough?
1:25:15	Clinton	He's questioning your judgement. "Obama valued my judgement. Namedrop situation room."
1:26:30	Sanders	Voted against wars. Under what circumstances would you use force? "No-fly is dangerous." "Kosovo I voted for."
1:28:30	O'Malley	Is Clinton too quick to use force? "No commander-in-chief should take military action off the table."

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
1:30:15	Clinton	Follow-up: "Happy when O'Malley endorsed me. Consider him a friend. We're already flying in Syria. Tough decisions."
1:31:45	Webb	"Three strategic points. Iraq. Arab Spring. Iran fascination."
1:33:30	Sanders	"I think Putin will regret what he's doing."
1:34:15	Clinton	Should you have seen Benghazi coming? "There was about to be a massacre. Europe was begging us."
1:36:15	O'Malley	"There are lessons to be learned from Benghazi. We need better HUMINT."
1:36:45	Webb	"This was not about Benghazi, it was about the inevitability of something occurring."
1:37:30	Webb	Sanders was a conscientous objector. Can he be commander in chief? "Everyone should follow the legal process and I respect that."
1:38:30	Sanders	Why can you be CiC as a conscientous objector? "Thanks Webb."
1:40:00	Chafee	"Iran deal. No Webb, you're wrong."
1:40:15	Webb	"The signal we sent was that we are accepting Iran's greater position in the balance of power."
1:41:00	Chafee	Greatest threat to national security? "Chaos in the middle east."
1:41:10	O'Malley	"Nuclear Iran."
1:41:20	Clinton	"Spread of nuclear proliferation."
1:41:30	Sanders	"Climate change."
1:41:40	Webb	"Cyberwarfare."
		Break.

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
1:45:30	Clinton	Emails. "I've taken responsibility and said it was a mistake. What I did was allowed."
1:48:00	Sanders	"I think the American people are sick and tired of hearing about your damned emails." *handshake* *standing ovation*
1:49:15	Chafee	You said email was a huge issue. "Absolutely. American credibility."
1:49:45	Clinton	Do you want to respond? "No." *cheers*
1:50:00	O'Malley	Emails. "I believe that we don't care anymore because debates. We can talk about other issues."
1:51:15	Sanders	"Black lives matter. We need to combat institutional racism." *cheers*
1:52:15	O'Malley	"We have undervalued black lives."
1:53:00	Clinton	"I think Obama has been a visionary on this issue. Reform criminal justice."
1:54:30	Webb	*complaint about time* "All lives matter. I've had a long history working with African-Americans."
1:55:45	Sanders	How can you combat inequality when Obama couldn't? "We've created jobs. We need to create more by building public infrastructure, raise minimum wage, etc."
1:57:00	Clinton	You're part of the 1%, how can you represent the middle class? "We've worked really hard. I want to make sure everyone has the same opportunity."
1:57:45	O'Malley	"We raised the minimum wage and invested in infrastructure. We need Glass-Steagall."
1:59:00	Clinton	Glass-Steagall? "We need to deal with banks too big to fail. Shadow banks. Put attention on them, empower regulators to break them up."

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
2:00:00	Sanders	"Greed and reckless behaviour of Wall St, where fraud is a business model, helped destroy the economy."
2:01:00	Clinton	Follow-up: "I respect passion and intensity. I told Wall St. to cut it out! I have thought deeply about what we want to do."
2:02:00	Sanders	"Congress does not regulate Wall St, Wall St regulates Congress."
2:02:30	Clinton	"I think Dodd Frank was a very good start."
2:03:00	O'Malley	"After the repeal of Glass-Steagall, the big banks went from controlling 15% of our GDP to 65%."
2:03:30	Clinton	"Everyone changes their mind."
2:04:30	Sanders	Told without the 2008 bailout, you were told the country could collapse. You voted against it. "How about the people that caused the problem pay to fix it? (Goldman Sachs)
2:05:30	Webb	Is the system rigged? *complaining about time*
2:07:00	Chafee	You voted to make banks bigger. "My dad had just died and I just started."
2:08:45	Sanders	Do you think taxpayers should pay for rich kids' degrees? "They'll be paying more tax. Every kid should get it, regardless of parental income."
2:10:00	Clinton	Sanders says college and medicare should be free. "We want to reduce interest rates on student loans. Want students to work 10h/wk."
2:11:15	Clinton	"We fully support social security."
2:12:15	Sanders	"When you have poor pensioners, you lift social security, not lower it."

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
2:13:15	Sanders	Immigration. Why should latino voters trust you when you betrayed them? "SPLC said the provisions were semi-slavery."
2:14:30	Clinton	"I want immigrants to buy in to exchanges under the ACA."
2:15:15	O'Malley	"We are a nation of immigrants and made stronger by them."
2:16:00	Webb	"I wouldn't have a problem with immigrants getting Obamacare."
2:17:00	Clinton	Follow-up: "The republicans demonise immigrants."
2:17:45	Clinton	Undocumented immigrants should get state education? "Yes."
2:18:00	O'Malley	"We did this. It worked well. Trump is a clown."
2:19:00	Sanders	Why did it take so long for the Senate to respond to VA problems? "We put 15b into it."
2:20:00	Chafee	You voted for the Patriot act? "99-1 vote. It got broadened out of scope. I'd repeal parts of it."
2:20:30	Clinton	Regret vote on Patriot? "No. It was necessary."
2:21:30	Sanders	Only one to vote against. Would you shut down the NSA surveillance program? "Yes. We have the right to be free."
2:22:15	Chafee	Snowdon: traitor or hero? "I'd bring him home."
2:22:30	Clinton	"He broke the laws."
2:23:00	O'Malley	"He broke the law."
2:23:15	Sanders	"He broke the law, but what he did for us should be taken into account."
2:23:30	Webb	"I'd leave his judgement to the legal system, but the NSA have overreached."

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
2:24:30	Chafee	Name the way your administration would not be a third Obama term? "I'd change our approach to the middle east."
2:25:00	O'Malley	"I'd follow through on the promise to protect the mainstream economy, Glass-Steagall."
2:25:30	Clinton	"First female president." Is there a policy difference? *non-answer*
2:26:15	Sanders	"Corporate America is so great that the only way to transform is through a political revolution."
2:26:45	Webb	"Executive authority."
2:27:45	Sanders	Revolution? "We need a larger voter turnout. Improve transparency. Improve engagement."
2:28:15	O'Malley	"Green Energy Revolution!"
		Break.
2:33:30	Clinton	Why should democrats embrace an insider? "First female president. Lifetime of experience. I get things done."
2:34:30	O'Malley	"I've travelled everywhere, people say two things: New leadership. Get things done."
2:35:00	Clinton	Follow-up: "Don't vote based on name, look at what I've accomplished."
2:35:45	Sanders	"Only one without a SuperPAC."
2:36:30	O'Malley	Climate change? "Green energy!!"
2:37:45	Webb	You're pro-coal, oil, etc. Out of step? "I voted for all kinds of energy including renewable."

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
2:39:00	Sanders	Better than Clinton? "We need to move boldly. Introduced Carbon Tax legislation. Campaign finance reform - fuel industry funding republicans."
2:39:45	Clinton	"We've been trying to get to the Chinese about climate change."
2:41:00	Clinton	Paid family leave? "Republican scare-tactics. We can design and pay for a system that doesn't burden small business. Make the wealthy pay for it."
2:43:15	Sanders	"We are an international embarassment that we don't pay family leave. "
2:44:00	O'Malley	"In my state, we did this."
2:44:45	Sanders	Recreational marijuana? "I would vote yes. Too many lives destroyed for non-violent offenses. Wall St CEOs walk away yet we give jail to young people that smoke weed."
2:45:45	Clinton	"Not ready to take a position. Wait and see."
2:47:00	Sanders	How will you work with republicans? "They're total obstructionists. Get millions of people to vote for us, so republicans don't control the house or senate."
		Break.
2:53:00	Chafee	Enemy you're most proud of? "Coal lobby."
2:53:15	O'Malley	"NRA."
2:53:30	Clinton	"Everyone, basically. Republicans."
2:53:45	Sanders	"Everyone. Wall St and Pharma."
2:54:00	Webb	"Enemy soldier that wounded me but he's not around."

Continued on next page

Table 9 – *Continued from previous page*

Time	Candidate	Topic
2:54:15	Chafee	Closing statements.
2:55:45	Webb	Closing statements.
2:57:00	O'Malley	Closing statements.
2:58:45	Sanders	Closing statements.
3:00:15	Clinton	Closing statements.

REFERENCES

Agichtein, E., Castillo, C., Donato, D., Gionis, A., and Mishne, G. Finding high-quality content in social media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 183–194. ACM, 2008.

Aki, K. Earthquake prediction, societal implications. *Reviews of Geophysics*, 33(S1):243–247, 1995. ISSN 1944-9208. doi: 10.1029/95RG00396.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. Wireless sensor networks: A survey. *Computer networks*, 38(4):393–422, 2002.

Ali, R., Solis, C., Salehie, M., Omoronyia, I., Nuseibeh, B., and Maalej, W. Social sensing: When users become monitors. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 476–479. ACM, 2011.

Amento, B., Terveen, L., Hill, W., Hix, D., and Schulman, R. Experiments in social data mining: The TopicShop system. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(1):54–85, 2003.

Amruthalingam, L. *Early Detection of Real-World Events with Twitter*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, April 2015.

Arasu, A. and Garcia-Molina, H. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348. ACM, 2003.

Arasu, A., Babu, S., and Widom, J. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal—The International Journal on Very Large Data Bases*, 15(2):121–142, 2006.

Atzeni, P. and Mecca, G. Cut and paste. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 144–153. ACM, 1997.

Audeh, B., Beigbeder, M., Zimmermann, A., Jaillon, P., and Bousquet, C. Vigi4Med Scraper: A Framework for Web Forum Structured Data Extraction and Semantic Representation. *PloS one*, 12(1): e0169658, 2017.

Bandyopadhyay, A., Ghosh, K., Majumder, P., and Mitra, M. Query expansion for microblog retrieval. *International Journal of Web Science*, 1(4):368–380, 2012.

Bartman, B., Newman, C. D., Collard, M. L., and Maletic, J. I. srcQL: A syntax-aware query language for source code. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2017*, pages 467–471. IEEE, 2017.

Baumgartner, R. and Ledermiiller, G. Deepweb navigation in web data extraction. In *International Conference on Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, volume 2, pages 698–703. IEEE, 2005.

Baumgartner, R., Gottlob, G., and Herzog, M. Scalable web data extraction for online market intelligence. *Proceedings of the VLDB Endowment*, 2(2):1512–1523, 2009.

Bille, P. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1):217–239, 2005.

Bird, S. NLTK: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, pages 69–72. Association for Computational Linguistics, 2006.

Bird, S., Klein, E., Loper, E., and Baldridge, J. Multidisciplinary instruction with the natural language toolkit. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 62–70. Association for Computational Linguistics, 2008.

Bird, S., Klein, E., and Loper, E. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.

Blanvillain, O., Kasioumis, N., and Banos, V. BlogForever Crawler: Techniques and Algorithms to Harvest Modern Weblogs. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, page 7. ACM, 2014.

Breslin, J., Bojars, U., Passant, A., Fernandez, S., and Decker, S. Sioc: Content exchange and semantic interoperability between social networks. In *W3C Workshop on the Future of Social Networking*. Citeseer, 2009a.

Breslin, J. G., Decker, S., Hauswirth, M., Hynes, G., Le Phuoc, D., Passant, A., Polleres, A., Rabsch, C., and Reynolds, V. Integrating social networks and sensor networks. In *W3C Workshop on the Future of Social Networking*, volume 2009, 2009b.

Brickley, D. and Miller, L. The Friend of a Friend (FOAF) project, 2000.

Burke, J. A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. B. Participatory sensing. *Center for Embedded Network Sensing*, May 2006.

Buscaldi, D. and Hernández-Farias, I. Sentiment Analysis on Microblogs for Natural Disasters Management. Florence, Italy, May 2015. Association for Computing Machinery.

Cameron, M. A., Power, R., Robinson, B., and Yin, J. Emergency situation awareness from twitter for crisis management. In *Proceedings of the 21st International Conference Companion on World Wide Web*, pages 695–698. ACM, 2012.

Cao, D., Liao, X., Xu, H., and Bai, S. Blog Post and Comment Extraction Using Information Quantity of Web Format. In *Information Retrieval Technology*, pages 298–309, Harbin, China, January 2008. Springer. ISBN 978-3-540-68633-0.

Cassa, C., Chunara, R., Mandl, K., and Brownstein, J. S. Twitter as a Sentinel in Emergency Situations: Lessons from the Boston Marathon Explosions. *PLoS Currents*, 2013. ISSN 2157-3999.

Chang, C.-H., Kuo, S.-C., and others. OLERA: Semisupervised web-data extraction with visual support. *IEEE Intelligent systems*, 19(6): 56–64, 2004.

Chen, L. and Roy, A. Event detection from flickr data through wavelet-based spatial analysis. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 523–532. ACM, 2009.

Cheong, H. J. and Morrison, M. A. Consumers' reliance on product information and recommendations found in UGC. *Journal of Interactive Advertising*, 8(2):38–49, 2008.

Chew, C. and Eysenbach, G. Pandemics in the age of Twitter: Content analysis of Tweets during the 2009 H1N1 outbreak. *PloS one*, 5(11): e14118, 2010.

Ching, A., Murthy, R., Molkov, D., Vadali, R., and Yang, P. Under the Hood: Scheduling MapReduce jobs more efficiently with Corona, November 2012.

Chodorow, K. *MongoDB: The Definitive Guide*. " O'Reilly Media, Inc.", 2013.

Citron, D. K. and Franks, M. A. Criminalizing revenge porn. 2014.

Clark, J. and DeRose, S. *XML Path Language (XPath) Version 1.0*. 1999.

Crescenzi, V. and Mecca, G. Grammars have exceptions. *Information Systems*, 23(8):539–565, 1998.

Crescenzi, V., Mecca, G., Merialdo, P., and others. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.

Crockford, D. The application/json media type for javascript object notation (json). 2006.

Cunningham, H. GATE, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

DataSift. DataSift. http://datasift.com/, August 2010.

DataSift. Historical Architecture - Data Mining Billions of Tweets, December 2011.

De Choudhury, M., Counts, S., and Horvitz, E. Predicting postpartum changes in emotion and behavior via social media. In *Proceedings of the 2013 ACM Annual Conference on Human Factors in Computing Systems*, pages 3267–3276. ACM, 2013a.

De Choudhury, M., Gamon, M., Counts, S., and Horvitz, E. Predicting depression via social media. In *AAAI Conference on Weblogs and Social Media*, 2013b.

de León, J. C. V., Bogardi, J., Dannenmann, S., and Basher, R. Early warning systems in the context of disaster risk management. *Entwicklung and Ländlicher Raum*, 2:23–25, 2006.

Deligiannakis, A., Stoumpos, V., Kotidis, Y., Vassalos, V., and Delis, A. Outlier-Aware Data Aggregation in Sensor Networks. In *IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008*, pages 1448–1450, 2008.

Demirbas, M., Bayir, M. A., Akcora, C. G., Yilmaz, Y. S., and Ferhatosmanoglu, H. Crowd-sourced sensing and collaboration using twitter. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1–9. IEEE, 2010.

Doan, A., Domingos, P., and Halevy, A. Y. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM, 2001.

DOMO and Column Five Media. Data Never Sleeps, June 2012.

Driscoll, K. and Thorson, K. Searching and Clustering Methodologies Connecting Political Communication Content across Platforms. *The ANNALS of the American Academy of Political and Social Science*, 659 (1):134–148, May 2015.

Edison. The Social Habit, June 2012.

Elasticsearch BV. Kibana. https://www.elastic.co/products/kibana, September 2015a.

Elasticsearch BV. Significant Terms Aggregation. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-significantterms-aggregation.html, 2015b.

Facebook. Facebook APIs. https://developers.facebook.com/docs/reference/apis/, 2007.

Facebook. Login. https://developers.facebook.com/docs/facebook-login/, 2014.

Ferrara, E. and Baumgartner, R. Automatic wrapper adaptation by tree edit distance matching. In *Combinations of Intelligent Methods and Applications*, pages 41–54. Springer, 2011.

Ferrara, E., De Meo, P., Fiumara, G., and Baumgartner, R. Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70:301–323, 2014.

Fielding, R. *Representational State Transfer: An Architectural Style for Distributed Hypermedia Interaction*. PhD thesis, PhD Thesis, University of California, Irvine, 2000.

Florescu, D., Levy, A. Y., and Mendelzon, A. O. Database techniques for the World-Wide Web: A survey. *SIGMOD record*, 27(3):59–74, 1998.

FormulaPR. Social Networking in Summary, June 2011.

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L. *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617, June, 1999.

Fung, G. P. C., Yu, J. X., Yu, P. S., and Lu, H. Parameter free bursty events detection in text streams. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 181–192. VLDB Endowment, 2005.

Furche, T., Gottlob, G., Grasso, G., Schallhart, C., and Sellers, A. Oxpath: A language for scalable, memory-efficient data extraction from web applications. *Proceedings of the VLDB Endowment*, 4(11): 1016–1027, 2011.

Garrett, J. J. Ajax: A New Approach to Web Applications, February 2005.

Gilbert, C. H. E. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. 2014.

Gnip. Gnip. http://gnip.com/, March 2008.

Gogar, T., Hubacek, O., and Sedivy, J. Deep Neural Networks for Web Page Information Extraction. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 154–163. Springer, 2016.

Google. The bright side of sitting in traffic: Crowdsourcing road congestion data, August 2009.

Greenwood, S., Perrin, r., and Duggan, M. Social Media Update 2016, November 2016.

Grossman, L. Iran protests: Twitter, the medium of the movement. *Time Magazine*, 17, 2009.

Guerini, M., Strapparava, C., and Özbal, G. Exploring Text Virality in Social Networks. In *ICWSM*, 2011.

Guille, A. and Favre, C. Mention-anomaly-based Event Detection and tracking in Twitter. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 375–382, August 2014.

Gupta, K., Mittal, V., Bishnoi, B., Maheshwari, S., and Patel, D. AcT: Accuracy-aware crawling techniques for cloud-crawler. *World Wide Web*, pages 1–20, February 2015.

Guy, I., Avraham, U., Carmel, D., Ur, S., Jacovi, M., and Ronen, I. Mining expertise and interests from social media. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 515–526. International World Wide Web Conferences Steering Committee, 2013.

Hall, B. Social media trawled as government spends $4.3 million on research contracts. http://www.smh.com.au/federal-politics/political-news/social-media-trawled-as-government-spends-43-million-on-research-contracts-20140215-32snf.html, February 2014.

Han, J., Haihong, E., Le, G., and Du, J. Survey on NoSQL database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366. IEEE, 2011.

Hardt, D. The OAuth 2.0 authorization framework. 2012.

Harper, F. M., Moy, D., and Konstan, J. A. Facts or friends?: Distinguishing informational and conversational questions in social Q&A sites. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 759–768. ACM, 2009.

Hartig, O. and Pérez, J. Ldql: A query language for the web of linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 41:9–29, 2016.

Hartmann, M. Could Hillary Clinton Face Criminal Charges Over Emailgate? http://nymag.com/daily/intelligencer/2015/08/hillary-clinton-legal-emailgate.html, August 2015.

Hasbrouck, J. and Saar, G. Low-latency trading. *Journal of Financial Markets*, 16(4):646–679, 2013.

Heydon, A. and Najork, M. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

Hidayat, A. Phantomjs. *Computer software. PhantomJS. Vers*, 1(7), 2013.

Hirokawa Lab. Testbed for Information Extraction from Deep Web. http://daisen.cc.kyushu-u.ac.jp/TBDW/, June 2004.

Holmes, A. and Kellogg, M. Automating functional tests using selenium. In *Agile Conference, 2006*, pages 6–pp. IEEE, 2006.

Huang, C. Facebook and Twitter key to Arab Spring uprisings: Report. *The National. Abu Dhabi Media*, 6, 2011.

Ihm, S. and Pai, V. S. Towards understanding modern web traffic. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, pages 295–312. ACM, 2011.

Jeffery, S., Alonso, G., Franklin, M., Hong, W. H., and Widom, J. A Pipelined Framework for Online Cleaning of Sensor Data Streams. In *Proceedings of the 22nd International Conference on Data Engineering, 2006. ICDE '06*, pages 140–140, 2006.

Jindal, N. and Liu, B. A Generalized Tree Matching Algorithm Considering Nested Lists for Web Data Extraction. In *SDM*, pages 930–941. SIAM, 2010.

Jones, E., Oliphant, T., and Peterson, P. SciPy: Open source scientific tools for Python. 2014.

Jurgenson, N. Twitter Revolution. *The Wiley-Blackwell Encyclopedia of Globalization*, 2012.

Kao, H.-A. and Chen, H.-H. Comment Extraction from Blog Posts and Its Applications to Opinion Mining. In *LREC*, 2010.

Kautz, H. Data Mining Social Media for Public Health Applications. 2013.

Khoury, R., Dawborn, T., Gafurov, B., Pink, G., Tse, E., Tse, Q., Almi'Ani, K., Gaber, M., Röhm, U., and Scholz, B. Corona: Energy-efficient multi-query processing in wireless sensor networks. In *Database Systems for Advanced Applications*, pages 416–419. Springer, 2010.

Kietzmann, J. H., Hermkens, K., McCarthy, I. P., and Silvestre, B. S. Social media? Get serious! Understanding the functional building blocks of social media. *Business horizons*, 54(3):241–251, 2011.

Kirkorian, R. New Tweets per second record, and how! https://blog.twitter.com/2013/new-tweets-per-second-record-and-how, August 2013.

Klyne, G. and Newman, C. Date and time on the internet: Timestamps, 2002.

Kuc, R. and Rogozinski, M. *ElasticSearch Server*. Packt Publishing Ltd, 2013.

Lau, C. H., Li, Y., and Tjondronegoro, D. Microblog Retrieval Using Topical Features and Query Expansion. In *TREC*, 2011.

Lauder, S. Government's facial recognition system sparks privacy concerns. http://www.abc.net.au/news/2015-12-17/governments-facial-recognition-system-sparks-privacy-concerns/7035980, December 2015.

Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 233–246, New York, NY, USA, 2002. ACM.

Li, J. and Cardie, C. Early Stage Influenza Detection from Twitter. *arXiv preprint arXiv:1309.7340*, 2013.

Liakos, P., Ntoulas, A., Labrinidis, A., and Delis, A. Focused crawling for the hidden web. *World Wide Web*, pages 1–27, May 2015.

Liu, B., Grossman, R., and Zhai, Y. Mining data records in Web pages. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–606. ACM, 2003.

Liu, W., Meng, X., and Meng, W. ViNTs: Visual information aNd Tag structure based wrapper generator. http://www.data.binghamton.edu/vints/testbed.html, May 2005.

Liu, W., Meng, X., and Meng, W. Vide: A vision-based approach for deep web data extraction. *Knowledge and Data Engineering, IEEE Transactions on*, 22(3):447–460, 2010.

Ma, L., Goharian, N., Chowdhury, A., and Chung, M. Extracting unstructured data from template generated web documents. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 512–515. ACM, 2003.

MacCaw, A. *JavaScript Web Applications*. "O'Reilly Media, Inc.", August 2011.

Madden, M. Privacy management on social media sites. *Pew Internet & American Life Project*, 24, 2012.

Massoudi, K., Tsagkias, M., De Rijke, M., and Weerkamp, W. Incorporating query expansion and quality indicators in searching microblog posts. In *European Conference on Information Retrieval*, pages 362–367. Springer, 2011.

Mayfield, T. *A Commander's Strategy for Social Media*. V1, 2008.

Maynard, D., Bontcheva, K., and Rout, D. Challenges in developing opinion mining tools for social media. *Proceedings of@ NLP can u tag# user_generated_content*, 2012.

McCreadie, R., Macdonald, C., Ounis, I., Osborne, M., and Petrovic, S. Scalable distributed event detection for twitter. In *IEEE International Conference on Big Data, 2013*, pages 543–549. IEEE, 2013.

Meneghello, J. XPath Extension Types: Web Scraping. `https://murodese.github.io/xpath-scraping/`, April 2015.

Meneghello, J., Lee, K., and Thompson, N. Towards Social Media as a Data Source for Opportunistic Sensor Networking. Conferences in Research and Practice in Information Technology, Brisbane, Australia, November 2014. Australian Computer Society.

Meraz, S. Is there an elite hold? Traditional media to social media agenda setting influence in blog networks. *Journal of Computer-Mediated Communication*, 14(3):682–707, 2009.

Mesbah, A., van Deursen, A., and Lenselink, S. Crawling Ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB)*, 6(1):3, 2012.

Miao, G., Tatemura, J., Hsiung, W.-P., Sawires, A., and Moser, L. E. Extracting data records from the web using tag path clustering. In *Proceedings of the 18th International Conference on World Wide Web*, pages 981–990. ACM, 2009.

Mikroyannidis, A. Toward a social semantic web. *Computer*, 40(11): 113–115, 2007.

Mishne, G. and Glance, N. Leave a reply: An analysis of weblog comments. In *Third Annual Workshop on the Weblogging Ecosystem*. Edinburgh, Scotland, 2006.

Müller, H. and Freytag, J.-C. *Problems, Methods, and Challenges in Comprehensive Data Cleansing*. Professoren des Inst. Für Informatik, 2005.

Muslea, I., Minton, S., and Knoblock, C. A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 190–197. ACM, 1999.

Neto, J. L., Santos, A. D., Kaestner, C. A., Alexandre, N., Santos, D., and others. Document clustering and text summarization. 2000.

Noy, N. F. Semantic integration: A survey of ontology-based approaches. *ACM SIGMOD Record*, 33(4):65–70, 2004.

OpenSignal. WeatherSignal. http://weathersignal.com/about/, August 2013.

O'reilly, T. What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1):17, 2007.

o'Reilly, T. *What Is Web 2.0*. O'Reilly Media, Inc., 2009.

Ozdikis, O., Senkul, P., and Oguztuzun, H. Semantic Expansion of Tweet Contents for Enhanced Event Detection in Twitter. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 20–24, August 2012.

Parker, J., Wei, Y., Yates, A., Frieder, O., and Goharian, N. A framework for detecting public health trends with Twitter. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 556–563, August 2013.

Perrin, A. Social media usage. *Pew Research Center*, 2015.

Pipino, L. L., Lee, Y. W., and Wang, R. Y. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.

Plotly. Plot.ly, 2012.

Pol, K., Patil, N., Patankar, S., and Das, C. A Survey on Web Content Mining and extraction of Structured and Semistructured data. In *Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on*, pages 543–546. IEEE, 2008.

Poria, S., Cambria, E., and Gelbukh, A. Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108:42–49, 2016.

Python Software Foundation. Python Programming Language. `http://www.python.org/`, 1990.

Rahm, E. and Do, H. H. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

Ramos, J. Using tf-idf to determine word relevance in document queries. In *Proceedings of the First Instructional Conference on Machine Learning*, 2003.

Ratnaparkhi, A. and others. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, volume 1, pages 133–142. Philadelphia, USA, 1996.

Raza, M. and Gulwani, S. Automated Data Extraction using Predictive Program Synthesis. *Microsoft Research*, January 2017.

Richardson, L. Beautiful soup. *Crummy: The Site*, 2013.

Robinson, B., Power, R., and Cameron, M. A sensitive Twitter earthquake detector. In *Proceedings of the 22nd International Conference on World Wide Web Companion*, pages 999–1002, 2013.

Rosser, J. F., Leibovici, D. G., and Jackson, M. J. Rapid flood inundation mapping using social media, remote sensing and topographic data. *Natural Hazards*, pages 1–18, January 2017.

Russell, M. A. *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*. " O'Reilly Media, Inc.", 2013.

Sakaki, T., Okazaki, M., and Matsuo, Y. Earthquake shakes Twitter users: Real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web*, pages 851–860, 2010.

Sayyadi, H., Hurst, M., and Maykov, A. Event Detection and Tracking in Social Streams. 2009.

Sleiman, H. A. and Corchuelo, R. Trinity: On Using Trinary Trees for Unsupervised Web Data Extraction. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1544–1556, June 2014.

Stelzner, M. A. Social media marketing industry report. *How Marketers Are Using Social Media to Grow Their Businesses.*, 2011.

Strazdins, G., Mednis, A., Kanonirs, G., Zviedris, R., and Selavo, L. Towards vehicular sensor networks with android smartphones for road surface monitoring. In *2nd International Workshop on Networks of Cooperating Objects, Chicago, USA*, 2011.

Subrahmanyam, K., Reich, S. M., Waechter, N., and Espinoza, G. Online and offline social networks: Use of social networking sites by emerging adults. *Journal of Applied Developmental Psychology*, 29 (6):420–433, 2008.

Swanson, I. Sanders: Americans 'sick and tired' of Clinton emails. http://thehill.com/blogs/ballot-box/presidential-races/256860-sanders-americans-sick-and-tired-of-clinton-emails, October 2015.

Tang, L., Ni, Z., Xiong, H., and Zhu, H. Locating targets through mention in Twitter. *World Wide Web*, 18(4):1019–1049, July 2014.

Thamviset, W. and Wongthanavasu, S. Bottom-up region extractor for semi-structured web pages. In *Computer Science and Engineering Conference (ICSEC), 2014 International*, pages 284–289, July 2014a.

Thamviset, W. and Wongthanavasu, S. Information extraction for deep web using repetitive subject pattern. *World Wide Web*, 17(5): 1109–1139, 2014b.

Thapen, N., Simmie, D., and Hankin, C. The Early Bird Catches The Term: Combining Twitter and News Data For Event Detection and Situational Awareness. *arXiv preprint arXiv:1504.02335*, 2015.

Tuten, T. L. and Solomon, M. R. *Social Media Marketing*. Sage, 2014.

Ubermotive. The Ubermotive Guide to Media Influence, October 2012.

Van Rossum, G. and Drake, F. L. *Python Language Reference Manual*. Network Theory, 2003.

Vasalou, A., Joinson, A. N., and Courvoisier, D. Cultural differences, experience with social networks and the nature of "true commitment" in Facebook. *International Journal of Human-Computer Studies*, 68(10):719–728, 2010.

Vassiliadis, P. A survey of Extract–transform–Load technology. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(3): 1–27, 2009.

Voskresensky, M. Nielsen Research: Mobile consumer report 2013, March 2013.

Wandhöfer, T., Taylor, S., Walland, P., Geana, R., Weichselbaum, R., Fernandez, M., and Sizov, S. Determining citizens' opinions about stories in the news media: Analysing Google, Facebook and Twitter. *eJournal of eDemocracy & Open Government (JeDEM)*, 4(2):198–221, 2012. ISSN 2075-9517.

We Are Social Singapore. Social, Digital & Mobile in APAC, January 2014.

Weng, J. and Lee, B.-S. Event Detection in Twitter. In *ICWSM*, 2011.

Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., and Welsh, M. Monitoring volcanic eruptions with a wireless sensor network. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005*, pages 108–120, 2005. doi: 10.1109/EWSN.2005.1462003.

White, T. *Hadoop: The Definitive Guide: The Definitive Guide*. "O'Reilly Media, Inc.", May 2009. ISBN 978-0-596-55136-0.

Xia, T. Extracting structured data from Ajax site. In *Database Technology and Applications, 2009 First International Workshop on*, pages 259–262. IEEE, 2009.

Xu, L. and Embley, D. W. Combining the Best of Global-as-View and Local-as-View for Data Integration. In *ISTA*, volume 48, pages 123–136, 2004.

Xu, Z., Zhang, H., Sugumaran, V., Choo, K.-K. R., Mei, L., and Zhu, Y. Participatory sensing-based semantic and spatial analysis of urban emergency events using mobile social media. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):44, December 2016. ISSN 1687-1499. doi: 10.1186/s13638-016-0553-0.

Yardi, S., Romero, D., Schoenebeck, G., and Boyd, D. Detecting spam in a Twitter network. *First Monday*, 15(1), December 2009. ISSN 13960466.

Zauberman, G. The intertemporal dynamics of consumer lock-in. *Journal of consumer research*, 30(3):405–419, 2003.

Zhai, Y. and Liu, B. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web*, pages 76–85. ACM, 2005.

Zhao, S. *Detecting Events from Twitter in Real-Time*. M.s., Rice University, United States – Texas, 2013.

Zimmer, M. "But the data is already public": On the ethics of research in Facebook. *Ethics and information technology*, 12(4):313–325, 2010.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 12, 2017 (`classicthesis` version 1.0).