Autonomic approach to information discovery in crowd sourced data.

Interim Report

CM0343 - Individual Project

Abstract. This report documents the progress taken so far in demonstrating the applicability of an autonomic information discovery system being viable. This progress includes details regarding: the envisaged solution being of three collaborating parts; an adopted hybrid Waterfall Model/Agile project management strategy; background research into the suitability of the solution and scope for demonstration; the initial approaches for implementation and prototypes of some essential system mechanisms. Finally, a reflection on the report aims and aims detailed in the initial plan is conducted as well as proposed focus going forward.

Student Number: 1029310
Student Name: Liam Turner
Lecturer: Dr S Allen, Dr J Shao

Date: December 13, 2012
Hours Spent on this Exercise: 100
Special Provison:

Acknowledgements

With thanks to Dr S. Allen as project supervisor.
With thanks to Dr M. Chorley for access to crowd sourcing platform CrowdFlower.

Table of Contents

List of Figures

1 Introduction

The purpose of this project is to provide an autonomic solution to the problem of knowledge management of vast data sources such as Twitter. The primary intention of Twitter, as a social network, has arguably resulted in useful and related information being fragmented by the spatial and temporal nature of the social network; which subsequently results in using Twitter as a harvestable knowledge repository difficult. The aims of this report include demonstrating individual competency in gaining a conclusive background research into the problem and produce an envisaged solution to implement. Furthermore, this report aims to illustrate the influence of using appropriate research in gaining insight into the problem and methodologies to satisfy the aims. Lastly, to provide definitive documentation regarding progress of the approach, prototyping and focus going forward in relation to a defined project scope.

2 Envisaged Project

2.1 Project Aims and Objectives

The following aims and objectives provide an assessable means of creating a solution to the problem; these can be considered the initial set of requirements:

- Demonstrate the capabilities and suitability of processing aggregated Twitter content as a knowledge management solution.
- Create an autonomic platform for sustainable use and expansion.
- Allow for system mechanics to be nested under a loosely coupled tool-kit for use in contexts outside of overall solution.
- Investigate suitability of using natural language processing methods as a suitable means of categorising context of Tweets.
- Investigate a suitable means of ranking Tweets taking multiple metrics into consideration.
- Attempt to provide semantically accurate content related to the content of Tweets from external sources.

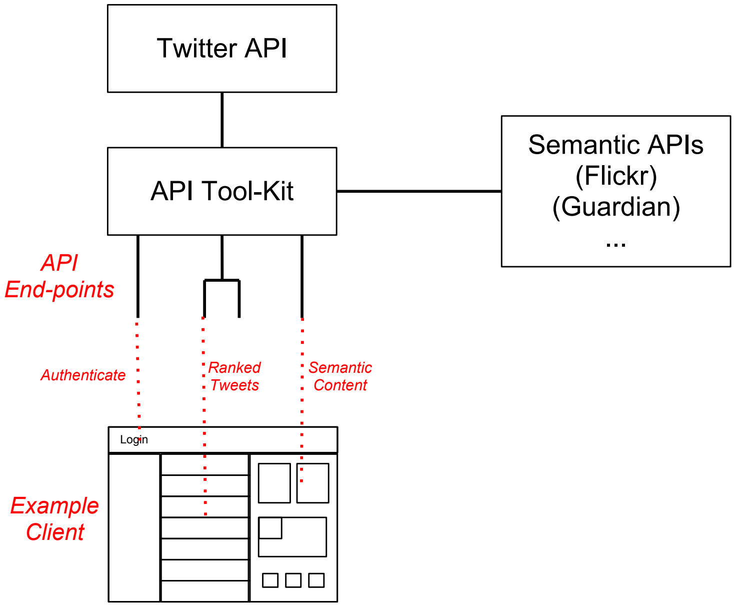

Fig. 1: Simple mock-up of envisaged implementation.

2.2 Methodology

To address the project aims and objectives, the envisaged solution will be split into components that resemble an abstract modelling methodology similar to that of the Model-View-Controller architecture (Reenskaug 1979). These components being Twitter data from the Twitter API (Model), an API tool-kit for processing Twitter data (Controller) and front-end clients visualising the processed data (Views) for consumers. An abstract model of the collaboration between these components is shown above, Figure 1. This methodology provides two main benefits, firstly the internal mechanisms of one component having minimal adverse effects any other component; results in increased development maintainability as any issues that arise will be relatively localised and aids in managing development against milestones. With limited control over access to data with the Twitter API, this approach would maintain a single methodology throughout. An additional benefit to this methodology is that the loose coupling would enable minimal changes to the controller and view components if using Twitter data was replaced with another data backbone containing repositories of sentences.

2.3 Project Scope

Whilst the envisaged solution and methodology aim to be as autonomic as possible, numerous restrictions force this instance of implementation to have a more restrictive scope. Firstly, the solution created in this project will aim to demonstrate the capabilities and suitability of the overall aims and objectives. This is due to limitations of time, resources and knowledge available as well as further uncontrollable limitations from restrictions of API calls to the Twitter API. Furthermore, as the project will be a demonstration, the implementation will not include a regulated authentication process for development of 3rd party clients (Views). However it should be noted that this feature could be added if further time was available and a need was evident. Lastly, as this project is experimental in its attempt to satisfy the aims and objectives, further additions to the project scope could be documented if discoveries resulting in limitations are found; through prototyping or complete implementation.

2.4 Project Management

The experimental nature of this project is likely to produce unforeseen conceptual issues and technical challenges to overcome or adapt to. Particularly regarding: Tweet context tagging; multi-objective Tweet ranking; the need for frequent testing and the unreliability of external data sources (Twitter and other chosen APIs for related semantics). Therefore it is proposed that the management of this project use augmented features from the Waterfall Model (Benington 1956) and Agile Methodologies (Highsmith and Fowler 2001). Particular elements of the Waterfall Model to adopt include:

- Loose structure of initial requirements, then design and prototyping, followed by full implementation.

Particular elements of the Agile Manifesto to adopt include:

- Welcoming of changing requirements, even in late development (better mapping to the experimental nature of the project).
- Working software is delivered frequently (better mapping to showing weekly progress to supervisor).
- Working software is the principal measure of progress (better mapping to milestones of initial plan).
- Sustainable development, able to maintain a constant pace (better mapping to the proposed methodology).

- Face-to-face conversation is the best form of communication (better mapping to weekly supervisor meetings).
- Projects are built around motivated individuals, who should be trusted.
- Continuous attention to technical excellence and good design.
- Simplicity - the art of maximizing the amount of work not done - is essential (better mapping to the project scope).
- Regular adaptation to changing circumstances (better mapping to the experimental nature of the project).

Exclusions of features from either methodology were result of either: better perceived effectiveness of a feature the other methodology or that the feature was not applicable given the individual nature of the project and the project scope.

3 Background

3.1 Standalone vs API Based Systems

The methodology used for this project proposes to use an API based system (Section 2.2); this decision was taken primarily due to the the overall greater suitability for the core concept of being an autonomic system, in contrast to a standalone application. Whilst the primary reason for creating a standalone system is arguably simpler due to being tailored for a single platform, removing cross-compatibility issues including: compiled code and display differences. However, the primary negative of this is the limited scalability, especially with the growth in different platforms from new technologies. The majority of platforms accept standardised data encoding methods such as XML and JSON; which would enable these platforms to use the same feed of data. This would create a single maintainable repository of code for these platforms, leaving discrepancies in the visualisation of data the only issue to overcome, maintaining platform flexibility with substantially less effort in maintainability the core functionality. Focus can still be made towards a particular platform for project scope, but the limitations in scalability are minimised in favour of greater complexity, ease of development, maintainability and the ability to introduce new features such as 3rd party clients without freely distributing code for the system mechanics.

3.2 Existing and Partial Solutions

Several of the major knowledge management solutions for aggregated social media data were explored to discover advantages and disadvantages in features. To summarise, firstly Sulia (Sulia 2012) claims to be the world's first subject-based social network and has relevance in the objectives of this project in sorting information by subject rather than by author; as well as provide a means of finding the best sources for each topic. However Sulia is a closed platform with a single web-client and whilst the inner mechanisms of its algorithms cannot be determined, no obvious opportunities exist to easily build upon its features. This is made evident as a Sulia endorsed news article (AllThingsD 2012) stated that the service had to be overhauled following Twitter's decision to drop affiliation and subsequently their custom streams. The user interface of the sole client has benefits of being seamless in scaling to a host of screen resolutions but in my opinion feels cluttered with unexplained features and constant pop-ups which makes it intimidating to the novice user. In terms of the primary partial solutions to similar problems, TweetDeck enables tweets to be grouped for better manageability, however appears to be very limited in scope integrating no ranking mechanisms other than by time. Overall TweetDeck appears to be a utility for a single solution of a single problem. Other partial solutions include Hibari (Wang 2012) which allows for easy muting of uninteresting information but the entire process is still very manual, minus a few steps and is limited to the Mac OSX operating system. ManageFlitter (89n 2012) also attempts to provide a means of using Twitter smarter and easier; but faces much of the same limitations of Hibari in that the process of finding interesting and legitimate information still a very manual time consuming process. Lastly, Twitter's native 'Discover' (Twitter, 2012) feature provides a means of finding potentially relevant recent tweets claimed to tailor to the user. However, the amount of tweets presented is low in regards to the spread of days previous it also covers and after using the service with a personal account proved almost random in what tweets it would display and poor in engaging interest. Other solutions were explored, but produced similar convergences in their advantages and disadvantages. Exploring these and other lesser solutions has influenced the required and desired features of this solution.

3.3 Use of External Services

As mentioned an objective of the system, highlighted in the methodology, is to be easily adaptable to any similar data backbone to Twitter, such as other social networks. For demonstration purposes however, Twitter was selected due to it being a popular platform that is used worldwide. Twitter also allows free access to its API, however does have restrictions which would need to be addressed if taken beyond the scope of this project's implementation. Furthermore, it is a popular platform used for social computing research, which arguably increases its significance in being an applicable choice to demonstrate with. A reflection on this choice will be made after full implementation in the final report of this project.

An additional aim of the system is to provide user's with relevant semantics to the tweet(s) they are currently viewing. The intention is to provide a similar service to Google's Knowledge Graph (Google Inc 2012) as a feature of this systems tool-kit. Given the scope of the system, features such as augmentation information that Google's Knowledge Graph provides will not be implemented. The purpose of which is to remove the need for having to go from the client to another service such as a search engine to gain full insight into the full context. To retrieve this information, such as related: images, news stories, definitions, videos, apps etc external APIs could be used, however like with the primary data backbone (Twitter) API limitations are also enforced which should also be considered if used outside of the demonstration purpose of this project. A full list of investigated API limits can be found in the Appendices, Figure 2.

4 Approach

4.1 System Architecture

The approach in regards to what features each system component will involve both required features and desired to have will include common features across the system components. Desired features aim to be implemented provided that sufficient progress in the required features are made so that time allows further implemented and that any technical challenges of the desired feature can be overcome. The primary common required feature will be authentication to allow for collaboration between the system components. Twitter requires OAuth (Parecki 2012) authentication by default to access data via its API, the systems own API tool-kit will also have authentication between itself and instances of the demonstration client. The required authentication between the API tool-kit and the demonstration client will predominately be used to differentiate between users of the client rather then as a security measure. As full OAuth authentication between clients and the API tool-kit would need to be implemented if the system was expanded beyond demonstration purposes this is seen as a desirable feature to ensure that arguably more important exclusive features of each component are completed. The second common required feature will be that the exchange of data between components use JSON encoding. The choice of using JSON over XML is arguably relatively arbitrary in the context of data exchange; however JSON will be used as it is arguably better suited for data conversion into a visualised format with a JavaScript based client. XML would arguably be better suited for relatively static data lookups alongside technologies such as RDF, which is not appropriate given in the scope of this project.

API Tool-Kit

The intended API architecture to adopt will be the REST web service architecture (Fielding 2000) over other web service architectures or a streaming API. REST is arguably the best suited due to being considerably more lightweight than SOAP and given the scope of the project, it is quicker to implement than a streaming API. REST is a widely adopted standard, including the endpoints of the Twitter API that this project will connect to. The following is a summary of the required and desired memories for the primary endpoints of the API tool-kit; full documentation of each end-point including input parameters will be included after finalised implementation in the final report. It should be noted that this is subject to change if further requirements or issues arise and these will be highlighted in the final report.

Keyword Extraction Endpoint - The primary required feature of keyword extraction from tweets will be the ability to extract out key words or phrases which define the context of the tweet with at least equal precision than existing free solutions (although better precision would be desirable). The primary reason for is is due to the keyword extraction being a core component which is integrated into the other end-points, access must be quick and reliable. If an external API was used, it would likely cause all tasks in the system to be slowed down due to bandwidth limits between the system and the external service. In terms of desired features, the ability to rank the keywords/phrases by most relevant or attach a relevancy score would be desirable but not essential. Another desired feature would be entity recognition of relevant keywords/phrases including: named entity recognition, event entity recognition and geo-location recognition.

Tweet Ranking Endpoint - A required feature of ranking of tweets will be to extract meta-data from tweets relevant to the integrity and interest of both the tweet and it's author. Another requirement will be to use the keyword extraction end-point in generating additional meta-data regarding the integrity and interests of tweets/authors such as cosine similarity against other similarities. Another required feature will be to use a multi-objective ranking algorithm with the objectives of integrity and interest to rank a group of tweets. The desirable features include creating an optimum formula for calculating objectives or integrating machine learning for interpreting a users perception of interest and integrity. Finally, a desired feature would include the ability to choose to rank by author objectives and then by tweet objectives or vice versa.

Semantic Insights Endpoint - A required feature of fetching relevant semantic insights is using the keyword extraction endpoint as input parameters of external APIs. A further required feature is calculate, where able, the relevancy of textual semantics with the set of input parameters and the keyword extraction endpoint on the semantic data. All textual semantics of a particular type (e.g. news stories) would then be ranked for output by percentage intersection between the input parameters and the keywords of the text. A desirable feature would be to evaluate the effectiveness of assessing the cosine similarity between similar text semantics (e.g. news articles) in order to detect and avoid duplicates. Depending on the implemented keyword extraction features, filtering of semantics could be performed using entity recognition to remove differences in ambiguous terms. Due to the unreliable nature of depending on external APIs features are subject to change if necessary; while the autonomic nature of the system aims to provide seamless error recovery, planned semantic content may become unavailable.

Demonstration Client
Although the intention of the system is to allow for multiple clients on any JSON compatible platform to be created, given the scope of the project a single client will be created; a cross-platform rich internet application. The required features of client beyond the common features will include a multiple pane graphical user interface: one containing navigation to display tweets using various Twitter API end-points as the input data for the API tool-kit; another will contain ranked tweets and another containing semantic information regarding the tweet(s) currently being looked at. Additional required features of the primary client include: a landscape mobile friendly interface; one-click sign in using a Twitter account and brief documentation. Desired features will include: portrait screen size user interface support; management of Twitter account within the client; personalisation of aesthetics using a user's personal Twitter design; swipe gesture support for mobile devices and buttons as a fallback. As with required and desired features of the API tool-kit, the demonstration client's features should be considered as initial ideas; any changes as a result of new or adapted requirements or technical challenges will be documented in the final report.

4.2 Technologies and Mechanisms

In terms of technologies to implement the features of each system component, Python will be the intended programming language used to create the tweet processing API tool-kit; more specifically the Python web framework Django (Django Software Foundation 2012). Further libraries will be utilised where appropriate including the REST API framework TastyPie (Lindsley et al. 2012) for creating standard consistent and maintainable API end-points; as well as other features such as authentication. This will leave more time for development of this components more novel features. The choice to use Python is a personal choice as a result of good feedback from previous users of Django and TastyPie; however the extent at which these were suitable and effective will be evaluated after full implementation in the final report.

Additional libraries will be used for internal processing Tweet data for the API-endpoints; a conclusive list of libraries will be stated after full implementation in the final report. For prototyping, the Python natural language tool-kit, NLTK (NLTK Project 2012) will be used for the keyword extraction and in elements of creating objectives for tweet ranking. The NLTK is widely adopted and arguably the most substantial in providing a means to perform natural language processing in Python; the extent of its suitability and effectiveness will be fully evaluated in the final report. The final primary mechanism intended to use for tweet ranking in the API tool-kit is the fast-non-dominated-sort algorithm nestled under the NSGA-II algorithm (Deb 2002), which allows for multi-objective sorting (of tweet/author interest/integrity objectives). Likewise to the other technologies and mechanisms of the API tool-kit, the suitability of the algorithm will be evaluated in the final report.

In regards to the technologies and mechanisms of the demonstration client, backbone.js (Ashkenas 2012) will be used as the primary framework, incorporating other yet to be finalised libraries for aesthetics and desirable features such as swipe gesture detection. The choice to use backbone.js was made after investigating the suitability of a Django client against JavaScript driven clients (see Appendices, Figure 31, Project Meeting Notes - Week 10) which found backbone to be faster than Django and relatively seamless integration with TastyPie, this systems chosen API structure framework. The framework also claims to be lightweight in that it has only a single library dependency to function. As the intention of the client is fort demonstration purposes, this will allow for focus to be made on the projects novel features.

Several other technologies will be used to support the system components including a database. Whilst the majority of user tasks will be performed in real-time, for efficiency, consumers Twitter OAuth credentials will be stored after initial authentication from the client through the API toolkit to Twitter. This will allow for one-click Twitter login after initial authentication. The presence of a connected database would also allow for simpler integration of further desirable efficiency features such as caching of searches. The chosen DBMS for this database will be PostgreSQL (The PostgreSQL Global Development Group 2012); a native supported DBMS for use with Django. Whilst many other DBMS are natively available, as well as multiple community created modules for NOSQL systems such as CouchDB, the scope in the use of a database being so small the difference is arguably trivial. Therefore PostgreSQL will be used as it is the default support in Django, and will be used in a relational sense and this choice will be evaluated in the final report.

Another supporting mechanism will be the use of a version control system to allow for: simple maintainability of stable/unstable implementations; back-ups of data and provide a means of assessing and documenting progress towards project management milestones. These features are only a subset of general features of the version control systems and therefore the particular choice of using of one over another is also relatively trivial in this project's context and scope. It should be noted that the use of these features to aid in the project management are not a necessity but are desirable and quickly implementable. Therefore, the software Git (GitHub Inc. 2012) will be used to fulfil these desirable features due to a seamless integration with the development environment used for implementation (Linux). Three main 'branches' of versions will be used, each containing sequential snapshots of the entire selected code base at a given date and time, known as 'builds'. One for the prototyping of features, one for unstable development (untested code) and a branch for stable development milestones.

Lastly, it is proposed that a two-stage testing strategy be used to compliment the predominately Agile driven project management approach (see Section 2.4). Whilst a full presentation and evaluation of the testing strategy will be evaluated in the final report, a proposal is to perform frequent testing myself on the 'unstable builds' of the version control system and periodically perform more substantial testing with a larger user set on the 'stable builds' (every 6 weeks if time is available). A means of maximising the effectiveness of testing of the 'stable builds' could be to utilise crowd-sourcing platforms such as CrowdFlower (CrowdFlower 2012); but the applicability will be researched at a later date.

5 Prototyping and Design

5.1 Initial API End-point Structure Design

As stated the intention of the API is to be as loosely coupled as possible so that it can act as a tool-kit for performing tweet processing tasks for contexts and projects outside, much like the demonstration client will. Therefore it is proposed that a tree design be used to determine which Twitter API data end-point to use as the input data, followed by the type of tweet processing that is required for output (e.g. Get Ranked Tweets for User Timeline or Get Semantics for term x). Access to the API endpoints (both branches and leaf nodes) is achieved through a URI detailing the traversal through the API tree structure using a forward slash as a separator of branch sequences e.g. 'example.com/api/discover/personal/timeline/feed' or 'example.com/api/discover/Indiana%20Jones/insights'. An initial design for the API structure can be found in the Appendices, Figure 11; any changes made to this design will be highlighted in the final report.

5.2 Prototype of Keyword Extraction Algorithm

The approach taken in extracting keywords that define the context of a tweet is to attempt to produce a method which is better than free services such as Alchemy (Orchestr8 2012). A method was devised to evaluate the precision effectiveness of Alchemy and any devised algorithms in extracting out keywords against those which a human would; using a set of 50 randomly selected tweets on various topics. The set of human keywords was created by combining 500 total sets of unique keywords with between 9-11 sets for each of the 50 tweets. This was gathered using the crowd sourcing platform CrowdFlower (Screenshots in Appendencies, Figures 3 through 5) with people either in the UK or the US for the first 450 results; although the results from 2 individuals had to be removed due to unusable responses. A focus group of 5 individuals was then used to produce the final sets up to 500 results which were then aggregated into a set of unique keywords for each tweet.

Analysing the precision of keywords/phrases Alchemy extracted in the set of human extracted keywords produced an average of 0.222; with the minimum and maximum in range being 0 and 0.5 respectively. An average of a 22.22% match per tweet using the Alchemy API keyword extraction endpoint resulted in the desirability of a new devised algorithm to be better at extracting keywords in terms of precision than Alchemy. An initial devised algorithm for extracting keywords in the system API was:

```
Input: A Tweet $t$, NLTK Brown Corpus Tagger
Output: A list of all types of nouns and verbs in the Tweet as $extractedKeywords$
$TokenisedSet =$ Tokenise $t$ by white space;
$TaggedSet =$ Tag each member of Tokenised Set with a part of speech classification.;
foreach word $w$ in $TaggedSet$ do
    if $w$.tag is a Noun OR $w$.tag is a Verb then
        append $w$ to $extractedKeywords$
    end
end
Return $extractedKeywords$;
```

Algorithm 1: 'Brown Tagger' - Basic Keyword Extraction Algorithm using Python NLTK.

The above algorithm uses the Brown corpus along with Python's NLTK library and consists of a multi-level fallback method for classifying the part of speech of words; in that it will first assess context of a word based on the 3 previous words, falling back to the previous 2, then previous word, then irrelevant of surrounding words, if a part of speech cannot be determined. The results of extracting keywords from the same 50 tweets showed that on average the algorithm produced 0.12 better precision than Alchemy; totalling 34.24% accuracy on average, with a 0 minimum score and 0.75 maximum score, also equal or better than Alchemy. Whilst the result is positive, due to the white-space tokenising of the algorithm key phrases containing multiple words are not possible and further precision could potentially therefore be achieved. Other limitations of the current algorithm include: punctuation is still attached to the beginning and ends of keywords and that duplicate keywords are possible. The following is an simplistic overview of an improved algorithm attempting to address these issues:

```
Input: A Tweet $t$, NLTK Brown Corpus Tagger
Output: A list of all types of nouns and verbs in the Tweet as $extractedKeywords$
$tokenisedSet$ = Tokenise $t$ by white space;
$textSegments$ = An empty list;
$keywordBuffer$ = An empty list;
foreach word $w$ in $tokenisedSet$ do
    Remove any prefix of affix sentence structure punctuation from word;
    if Sentence breaking punctuation was found in prefix then
        Append contents of $keywordBuffer$ as a single list to $textSegments$;
        $keywordBuffer$ = An empty list;
    end
    else if Sentence breaking punctuation was found in suffix then
        Append $w$ to $keywordBuffer$;
        Append contents of $keywordBuffer$ as a single list to $textSegments$;
        $keywordBuffer$ = An empty list;
    end
end
foreach segment $s$ in $textSegments$ do
    Tag each member of $s$ with a part of speech classification;
    $parsedTree$ = Parse each newly tagged member of $s$ against a grammar to join relevant
    consecutive words as a tree parent node of those members as children nodes.;
    traverse node $n$ in $parsedTree$
    if $n$ contains/is a Noun OR $n$ contains/is a Verb AND the Levenshtein edit-distance of $n$
    against current members of $extractedKeywords$ >2 then
        Append $n$ to $extractedKeywords$;
    end
end
Return $extractedKeywords$;
```

Algorithm 2: 'Brown Parser' - Advanced Keyword Extraction Algorithm using Python NLTK.

As the above algorithm is subject to changes based in performance in full implementation, a full breakdown will be provided in the final report. In terms of results, The improved algorithm produces a 0.1388 increase in average precision from the basic algorithm and a 0.259 average precision increase against Alchemy at 0.4812; with minimum precision of 0.11 (better than Alchemy) and a maximum precision of 1 (equal to Alchemy). However the final prototype algorithm does have its limitations including: a worst runtime complexity than the basic algorithm and lack of desirable features such as ranking or scoring each keyword/phrase in terms of relevancy and entity recognition. To assess the reasons why the precision was less than 50% on average, the 500 human result sets were examined and it was discovered that where multiple consecutive keywords were chosen, these was great variation with words included either side of a central keyword or group of keywords. Therefore arguably, these variations of the same context was effecting the precision scores. Taking this into consideration, along with further analysis showing that on average the human participants agreed with each other 32.45% of the time and therefore the algorithm in its current state was deemed appropriate given the scope of the project. Full visualisations of results and method, including: details of the Recall and F1 measure scores (15); further experimentations using co-occurrence human keyword/phrases (16 and (17) as well as analysis between the derived algorithms and Alchemy are all available in the Appendix, Figures 13 through 18, Project Meeting Notes - Week 7.

As well the improvements to the runtime complexity of the algorithm, and implementations of further desired features, alternatives to the concept of exploding tweets and rejoining relevant words together have surfaced in academic literature which could be explored; such as (Li et al. 2012) which proposed the concept of calculating a probability that splitting a string makes sense in order to create a list of keywords/phrases.

5.3 Prototype of Tweet Ranking Algorithm

In regards to Tweet ranking, the fast-non-dominated-sorting algorithm as part of the NSGA-II algorithm (Deb 2002) provides an appropriate multi-objective ranking algorithm to use. However, the values of the corresponding objectives: Tweet Interest, Tweet Integrity, Author Interest and Author Integrity required creation of bespoke formulas using combinations of retrievable variables (e.g. Tweet Retweet count) and calculated variables (e.g. Cosine similarity of a tweet against others tweets in a group). A full list of initial objective formulas and each variable/metric involved can be found in the Appendices, Figure 19, Project Meeting Notes - Week 8. The metrics used were influenced from (Uysal and Croft 2011); it should be noted that these are subject to change following experimentation during full implementation. It is recognised that machine learning could have been used however, given the scope of the project formulas was chosen instead. This decision will be reflected upon in the final report.

Several existing Python libraries were explored for a fast-non-dominated-sort implementation, however no simple usable implementation was found as the method of the sort was tightly coupled with other processes in the NSGA-II algorithm as a whole. Therefore, the sorting algorithm was implemented by myself in Python from the pseudo code in the original literature (Deb 2002, p. 184). A simple outline of the tweet ranking process is as follows:

```
Input: A set of Tweet Objects, $T$
Output: A set of sorted Tweet Objects, $sT$
foreach tweet $t$ in $T$ do
    Extract out useful metrics directly from $t$ (e.g. Retweet Count);
    Calculate additional metrics (e.g. Author Tweets Per Day);
    Calculate and attach Tweet Integrity, Tweet Interest, Author Integerity and Author Interest
    scores from metrics to $t$;
end
Perform fast-non-dominated-sort on $T$ for Tweet Integrity and Tweet Interest and store new rank
index for each tweet;
Perform fast-non-dominated-sort on $T$ for Author Integrity and Author Interest and store new rank
index for each tweet;
Sort Tweet Objects by Author Rank then by Tweet Rank (or vice versa) as $sT$;
Return $st$;
```

Algorithm 3: Tweet ranking using fast-non-dominated-sort with the Python NLTK.

The above algorithm intentionally performs the sorting twice to create a Tweet rank as well as an Author rank, rather than using all 4 objectives for a single rank. Whilst the runtime complexity is increased, it provides benefits such as two-level sorting i.e. Sorting by Author Rank, then by Tweet Rank, or vice versa (See Appendices, Figure 21, Project Meeting Notes - Week 8). A desirable feature of the tweet ranking will be to allow the user of the client to select the order of rank, which this method leaves open for simple implementation.

Several initial ranking tests were performed to ensure that appropriate values were being generated for each objective so that values can be relatively evenly correlated so that appropriate ranking of tweets are made in the sort; see Appendices, Figures 22 through 30, Project Meeting Notes - Week 8/10. The results of these showed that the fast-non-dominated-sort was implemented as intended, however the values for Tweet Integrity were typically either 0 or 1 and modifications will therefore need to be in the focus going forward to create a wider range of values for better differentiation.

5.4 Envisaged Semantic Insights Algorithm

A prototype was not performed for this algorithm due to its lack of novel required features, however prototypes of desirable features will be performed later in full implementation on the 'unstable' development branch (see Section 4.2). An abstract algorithm for retrieving and processing semantics is as follows:

```
Input: Input Parameters, $P$, external API, $A$
Output: A set of formatted semantic objects, $sO$
Query $A$ using parameters $P$, storing response if valid as $R$;
foreach object $o$ in $R$ do
    Process $o$ depending on media type (e.g. image, news article etc) (e.g. create thumbnail, remove unncessary content etc.) store in $sO$.;
    if $o$ is a text object (e.g. news article) then
        Extract keywords using advanced keyword extraction algorithm.;
        Calculate precision against $P$;
    end
end
if $R$ contained texts objects then
    Sort $R$ by precision scores.
end
Return $sO$;
```

Algorithm 4: Tweet ranking using fast-non-dominated-sort with the Python NLTK.

Additional calculations for text objects could be implemented, such as cosine similarity could be used to determine integrity from multiple sources; Levenstein edit distance could be used to determine and remove duplicates.

5.5 Initial Demonstration Client Design

The demonstration client aims to demonstrate the entire capabilities of the API tool-kit in synergy and illustrate this on as wide set of platforms as possible. Given the time-frame of this report a clear defined design has not been completed. Initial wire-frame designs however can be found in the Appendices, Figures 6 through 10, but are subject to change and any changes made will be documented in the final report. CSS3 media queries will be used to modify the aesthetics according to screen-resolution and pixel density, these ranges will differentiate between smart phone, tablets and laptop/desktop computer screens. Further details regarding the demonstration client will be stated in the final report.

6 Conclusions

In conclusion of the progress made towards the background understanding, approach, and prototyping documented in this report, on reflection the report aims and objectives as well as the intended content of the interim report defined in the initial plan have been satisfied overall. In regards to the milestones stated in the individual report, all milestones set to be achieved at this stage have been, however it should be noted that the week by week plan did have some alterations and the designated week for overflow was used. Whilst minimal issues arose during the research and prototyping, it is recognised and envisaged that more issues and greater technical challenges are likely to arise through implementation on a larger scale and introduction of rigorous testing. It also became apparent that the creation of multiple clients for various platforms is foreseen to not be applicable or neccessary given the time-constraints; therefore the chosen demonstration client will aim to cover as many platforms as possible in a single implementation.

At this time, oo changes are needed to be made to the time plan going forward into the second semester, including the sequential order of the week by week focus remainingas defined; although conclusions from the prototypes have given clearer focus of the tasks to perform in the initial weeks. On reflection the integration of overflow weeks in the second semester are likely to be used and on reflection was a good precaution to integrate. At this stage the final implementations of prototypes will be used going forward into full implementation, however as stated any changes and implemented desirable features will be documented in the final report.

7 References

89n, 2012. *ManageFlitter* [Online]. Available at: http://manageflitter.com/ [Accessed: 12 December 2012].

AllThingsD. 2012. *Life After Twitter: Sulia Builds Its Own Social Network* [Online]. Available at: http://sulia.com/news [Accessed: 12 December 2012].

Ashkenas, J. 2012. *Backbone.js* [Online]. Available at: http://backbonejs.org/ [Accessed: 12 December 2012].

Benington, H.D. 1956. Production of Large Computer Programs. In: Navy Mathematical Computing Advisory Panel. *Proceedings of the Symposium on Advanced Programming Methods for Digital Computers*. Washington, D.C., 28-29 June, 1956. Washington, D.C: Office of Naval Research, pp. 15-27.

CrowdFlower. 2012. *CrowdFlower The Worlds Largest Workforce* [Online].
Available at: http://crowdflower.com/ [Accessed: 12 December 2012].

Deb, K. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evoluationary Computation* 6(2), pp. 182-197.

Django Software Foundation. 2012. *The Web framework for perfectionists with deadlines.* [Online].
Available at: https://www.djangoproject.com/ [Accessed: 12 December 2012].

Fielding, T.R. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* PhD Thesis, University of California.

Google Inc. 2012. *The Knowledge Graph Learn more about one of the key breakthroughs behind the future of search.* [Online]. Available at: http://www.google.com/insidesearch/features/search/knowledge.html [Accessed: 12 December 2012].

GitHub Inc. 2012. *github* [Online]. Available at: https://github.com/ [Accessed: 12 December 2012].

Highsmith, J. and Fowler, M. 2001. The Agile Manifesto. *Software Development Magazine* August 2001, pp. 29-30.

Li, C. et al. 2012. TwiNER: named entity recognition in targeted twitter stream. In: *SIGIR'12: Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval.* Portland, Oregon, USA, 12-16 August, 2012. New York: ACM, pp. 721-730.

Lindsley, D. et al. 2012. *Welcome to Tastypie!* [Online]. Available at:
http://django-tastypie.readthedocs.org/en/latest/ [Accessed: 12 December 2012].

NLTK Project. 2012. *NLTK 2.0 documentation* [Online]. Available at: http://nltk.org/ [Accessed: 12 December 2012].

Orchestr8, LLC. 2012. *AlchemyAPI Keyword / Terminology Extraction* [Online]. Available at:
http://www.alchemyapi.com/api/keyword/ [Accessed: 12 December 2012].

Parecki. A. *An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.* [Online]. Available at: http://oauth.net/ [Accessed: 12 December 2012].

Reenskaug, T. 1979. Thing-model-view-editor-an example from a planningsystem. *Xerox PARC technical note* May 1979.

Sulia. 2012. *Welcome to Sulia, the Subject-Based Social Network* [Online]. Available at: http://sulia.com/about [Accessed: 12 December 2012].

The PostgreSQL Global Development Group. 2012. *http://www.postgresql.org/* [Online]. Available at: http://www.postgresql.org/ [Accessed: 12 December 2012].

Twitter, 2012. *Tweets What's happening now, tailored for you.* [Online]. Available at: https://twitter.com/i/discover [Accessed: 12 December 2012].

Uysal, I. and Croft, B.W. 2011. User Oriented Tweet Ranking: A Filtering Approach to Microblogs. In: Berendt B. et al. eds. *CIKM'11: Proceedings of the 20th ACM international conference on Information and knowledge management.* Glasgow, Scotland, UK, 24-28 October, 2011. New York: ACM, pp. 2261-2264.

Wang, V. 2012. *hibari the cleanest tweet stream* [Online]. Available at: http://hibariapp.com/ [Accessed: 12 December 2012].3