Gatekeeper: Rebuilding Privacy for Online Social Networks

Oliver Jones

June 16, 2015

Abstract

In order to maintain privacy on online social networks, such as Facebook, users must define a privacy policy through an arduous multi-step process for each status update and each of an average of 338 friends. Despite the functionality to set privacy policies per status update, a recent survey found 36% of Facebook users still strongly disliked others sharing too much about themselves and 27% of users still strongly disliked someone seeing posts or comments they did not mean for them to see. Much of the state-of-the-art has revolved around clustering users based on observable friendships lists, which is no longer feasible; in this work, we present empirical results from our method of clustering users into friendship groups without access to friendship lists. We also show that utilising knowledge about these groups can generate topic models that are competitive with LDA. We present a probabilistic model for predicting whom a user would want to share a real-time status update with, utilising these topic models. Our results show this ordering over a user's friends is accurate 82.6% - 97.4% of the time. We implemented a plugin for Facebook based on our model for audience prediction and topic modelling. Our approach can be generalised to any data with an assumption that authors or publishers know one another. For example, in this work we use data from Facebook, Twitter, and a screenplay.

Acknowledgements

I would like to thank Dr. Emil Lupu, who listened to my ideas nine months ago, saw potential in them, and brought in guidance and support in the form of both Dr. Daniele Sgandurra and Dr. Luke Dickens. I am indebted to all three for the countless hours spent inspiring, encouraging and critiquing.

Thank you to my friends and family for offering support and thanks to those who offered intimate anecdotes, putting personal meaning behind this project.

Contents

1 Introduction

> *"People have really gotten comfortable not only sharing more information and different kinds, but more openly and with more people [...] that social norm is just something that has evolved over time."*
>
> - Mark Zuckerberg 2010 [1]

Zuckerberg's statement caused quite a stir [2, 3], and whilst the online social landscape has changed greatly over the past decade, many were incredulous to believe that new social norms around sharing and privacy have developed, seemingly over night. Indeed research [4, 5, 6] suggests that privacy and over-sharing are still two of the most highly ranked concerns of online social network users.

Users do limit the audiences of posts: for example, Kairam et al. when interviewing users of Google+ discovered that 74.8% of users had shared at least one update in the period of a week with only a selective audience [7]. Ross et al. discovered that, since Zuckerberg's comments, users are being more selective when sharing information on Facebook with only 47.4% of users sharing their friendship list publicly in June 2011, down from 82.7% 15 months previous [8].

The notion of privacy far precedes the invention of online social networks: Warren and Brandeis penned The Right to Privacy in the Harvard Law Review in 1890 [9], predating the notion of computers as anything beyond the gears and cogs of Babbage's Difference Engine. Zuckerberg's comments seem out of place given the century long history of people caring about privacy. For this reason this project contributes to a movement that is unlikely to lose relevance over the course of a decade. Information control theories have developed within multiple fields. Around the same time as the right to privacy was published in the Harvard Law Review, research started to develop around a conceptual approach to measuring and understanding the control someone has over communications within a network, called *centrality* [10]. Centrality developed mainly within the context of graph theory over the next one hundred years and much of the related work in the field of privacy revolves around network properties such as centralities.

The way in which users currently restrict access to content seems to revolve around grouping friends into friendship lists on Facebook [11] and Circles on Google+. However, according to Jones and O'Neill's research on Facebook privacy [12], very few people create or update friendship lists beyond a "Limited Profile" grouping and feel that the automatically created groups are often too incomplete to be useful. However, grouping people is one of the ways we manage to maintain such a high number of friendships in the first place [13].

It is our hypothesis that there is an underlying mapping of friends to groups that share common interests, but the current tools available to users do not meet their expectations around usability. We believe that these groups can be predicted and we can utilise topic modelling between content produced and shared within them to make predictions about the intended audience of a previously unseen status update. Previously research has used highly connected sub graphs or other properties

in the network of friendships that surround a user [14, 15, 16] to help automate the selection of the intended audience of a piece of content. However, this graph is increasingly difficult to build given the increasing trend towards people hiding this information publicly, and Facebook limiting the functionality of this API endpoint [17].

We use datasets from Twitter and Facebook to prove our approach works on a variety of social media content. We use our approach on a screen play in order to evaluate using an established ground truth of groups of characters. We build a plugin for Facebook to evaluate how we can present our audience predictions to a user. Inspired by the concepts of *homophily* and *registers* we jointly consider observable statuses and audiences to provide a novel approach to topic modelling.

1.1 Motivations

During recent personal discussions around the issue of privacy on online social networks, a story came to light that motivates the current situation well. In this anonymised scenario, there is a Facebook user, *Bob*, who has many friendships on Facebook including close family. *Bob* described the situation as thus: "I want to share LGBT[1] related content on my Facebook to start to feel part of the community and to build a support network. However, I don't want to reveal this to my close family". *Bob* is not alone in having privacy concerns: Pew Research [6] found that 27% of those surveyed strongly disliked someone seeing posts or comments they did not mean for them to see, and 36% of people surveyed actually strongly disliked other people sharing too much about themselves.

However, the current multi-step process[2] to change the audience of a post, as in Figure 1, is too cumbersome to go through when posting every status. We give an example *social graph* for Bob's scenario in Figure 2. Vertices represent *Bob*'s friends, and edges represent a confirmed friendship connection on Facebook. We see Friends A, B, C, D, and *Bob* create a highly connected sub graph, which is a property some algorithms look for when automatically grouping a user's friends. We can also see that Family Member A is a friend of Friend D, so *Bob*'s "friends of friends" include a family member, as would a "public" privacy setting, so *Bob* must be careful not to use these audience settings, when posting content intended for one subset of his social graph.

We can only construct such a graph as in Figure 1 with access to everyone's lists of friends on Facebook, which we no longer do through the Facebook API [17].

The problem of selecting the right audience for a post to ensure a users privacy is only going to affect more and more people, with the popularity of Facebook increasing year on year with currently over 1.3 billion monthly active users [18]. However, the demographic of Facebook is slowly shifting towards an older user base [19]. Privacy concerns amongst social network users are correlated with an increase in age [4]. Although Ross et al. [8] have measured a significant shift towards users limiting the audience of their profile information over 15 months for all ages.

[1] LGBT stands for *Lesbian, Gay, Bisexual and Transgender* and is used as a *catch-all* term for people identifying along the queer spectrum

[2] As of 29th January 2015

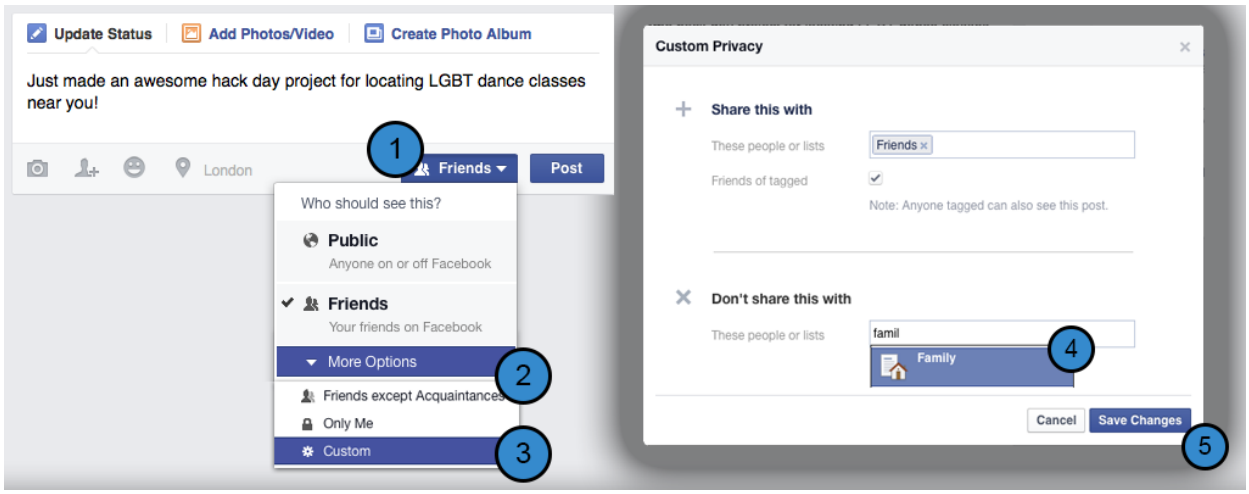

Figure 1: The multiple step process for posting a status Bob's family will not be able to see.

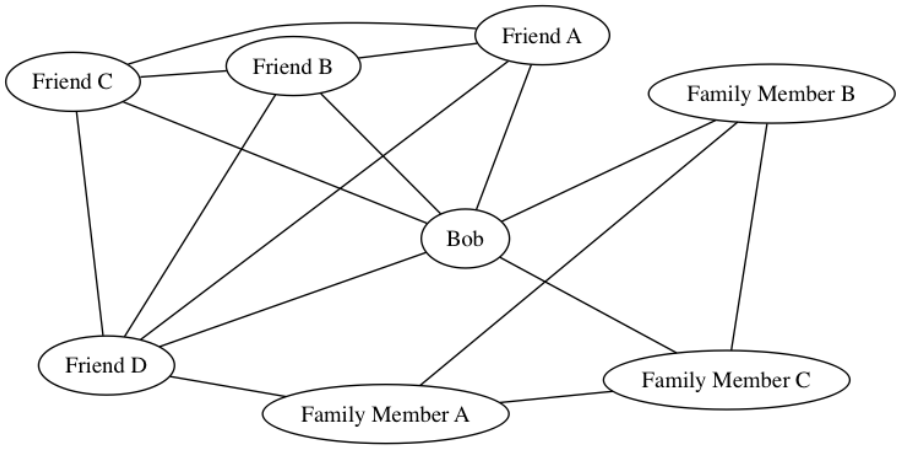

Figure 2: *Social graph* for *Bob*

At the end of 2010, Dunbar reported the average user had 120-130 Facebook friends [20]. Just over 3 years later Pew Research stated that the mean number of Facebook friends per user has risen to 338 [6] (roughly 70 more connections per year). We each have multiple social identities [21] and the more friends we have, and the more social groups we participate in, the more social identities we have to maintain. As the trend in growth of the number of friendships continues it will only become more necessary to control information dissemination across our network of friends; thus giving us more control over the ways we portray our multiple identities and ensuring that we can continue to maintain the stability of all our friendships.

We still have the problem that even with automatic segmentation of Bob's social graph in order to give groupings for his friends and family, Bob still needs to classify each status. In order to improve this situation we need a method that can build a user's *social graph* without access to a list of their friendships, and a method that can identify topics not just in a user's status update,

but also within the friendships groups we have identified from their *social graph*.

Topic modelling performs badly in the domain of online social network content as outlined by [22]. Status updates as a corpus, are short, for example on Twitter status updates are a maximum of 140 characters or less, and often a large number of characters are taken up by URLs, tags, and Twitter names. Both entity extraction and topic modelling have to deal with informal language, misspellings, acronyms and slang. Zhao et al. [23] show that this reduces the performance of LDA.

We have a situation where much of the previous state-of-the-art has relied on the Facebook "/user/friends" API behaviour that no longer exists, and the content we do have access to is unstructured natural language, of which a document is very short, access is still rate limited and the informality of the language makes processing particularly difficult. In this paper we solve both problems in parallel.

1.2 Contributions

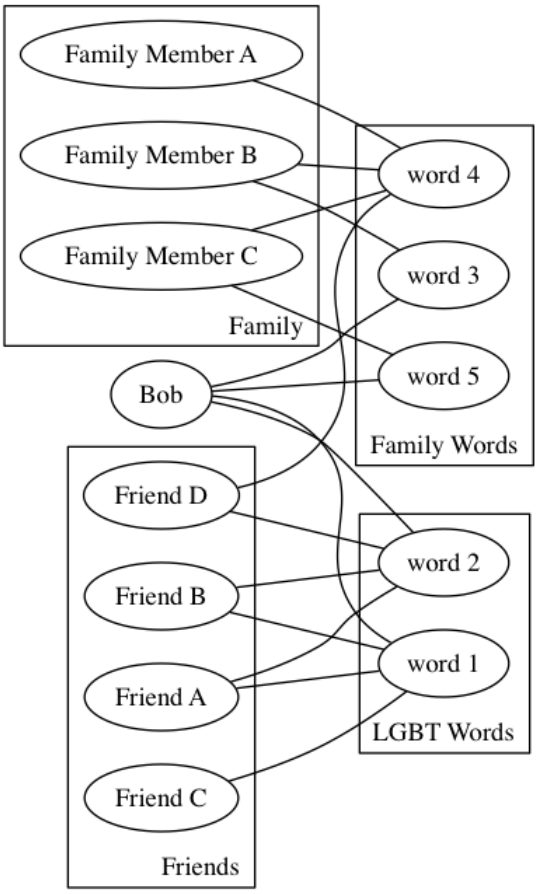

Figure 3: A *register graph* for Bob.

We explore a novel method of topic modelling using a graph based approach utilising online social network conversations where we consider both the recipients of messages and the content of the

messages jointly. We evaluate this against the existing solutions in the literature. Figure 3 shows an example of this approach. In this figure, we have edges that join words to the people that utter them, thus building a *register graph*. We have clustered together words and people to illustrate how our approach discovers these highly connected communities, and thus friendship groups and the vocabulary used within them. In this trivial instance we can see Friends A, B, C, D, and *Bob* all speak about the LGBT topic, with Friend D also talking about the Family topic (this cluster would become more complex as we reason about more topics). We can also see that *Bob* and family members B and C talk about family matters and no other topics.

We have built a solution that as *Bob* writes a status on Facebook, we can analyse the topics the status is discussing and automatically suggest an audience built from this *register graph*. In contrast with the current process on Facebook, with our work *Bob* no longer has to remember to update these settings per message, but instead to just verify and amend the suggestions. We also show that the process is applicable to Twitter and other conversational datasets such as screenplays, but focus on Facebook for our plugin implementation due to API support for posting status updates with a specified audience.

We named our plugin from Barzilai-Nahon's survey of Gatekeeping concepts [24], which explores the many ways in which data is altered, removed or restricted as it passes through a network. We focus explicitly on *withholding* information as it passes through a social network. Gatekeeping theory is much broader than *withholding* information, which reflects that this work is a small part of a much larger narrative of bringing privacy in the online world up to speed with the expectation of privacy we have in the offline world.

We present our solution in six sections. In each we focus on the three main challenges this project represents: entity extraction, topic modelling, and audience prediction.

1. The next section, Section 2 details related work.
2. In Section 5 we outline our proposed method.
3. In Section 6 we step through the implementation details of our plugin.
4. In Section 7 we detail our results.
5. Finally, in Sections 8 and 9 we detail our discussion and conclusions respectively.

2 Related Work

In this section we focus on the five most relevant pieces of related work whose results have shaped our approach, or have outlined methodologies that parallel our own and serve as a basis for evaluating our solution. However, there is lots of literature in this field and throughout the report we make reference to several other relevant papers. Our five most relevant pieces of work are roughly clustered into two parts:

1. Audience prediction, with *Identifying Hidden Social Circles for Advanced Privacy Configuration* by Squicciarini et al. [25]; *Predicting Sharing Policies for Text-based Content on Facebook* by Sinha et al. [26]; and *Learning to Rank Audience for Behavioral Targeting in Display Ads* by Tang et al. [27].

2. Topic modelling, with *Enhancing Text Clustering by Leveraging Wikipedia Semantics* by Hu et al. [28]; and *A Graph Analytical Approach for Topic Detection* by Sayyadi and Raschid [29].

Audience prediction and topic modelling are the two main components of our solution.

2.1 Identifying Hidden Social Circles for Advanced Privacy Configuration

Squicciarini et al. [25] propose a methodology for discovering hidden social groups within a user's online social network friendships, based on an adaptation of the Apriori algorithm [30]. The Apriori algorithm groups together transactions. Squicciarini et al. structure a transaction as a set of features that occur on a user's friends profile. They suggest features such as the ages of the users, the educational institute, the hobbies a user has listed on their profile, and their relationships with other users called "rel" in the paper could be used to group together users. The paper gives the following example:

$$u_1 : [(age, 20), (education, \text{PennState}), (hobbies, \text{Swimming}), (rel, 1 : friend : 2)]$$
$$u_2 : [(education, \text{PennState}), (hobbies, \text{Hiking}), (rel, 2 : friend : 1)]$$
$$u_3 : [(education, \text{PennState}), (hobbies, \text{Tennis}), (rel, 3 : friend : 1)]$$
$$u_4 : [(age, 23), (education, \text{Stanford}), (hobbies, \text{Basketball}), (rel, 4 : friend : 1)]$$
$$u_5 : [(age, 20), (education, \text{UCLA}), (hobbies, \text{Movie}), (rel, 5 : friend : 1)]$$
$$u_6 : [(age, 21), (hobbies, \text{Movie}), (rel, 6 : friend : 1)]$$
$$u_7 : [(age, 22), (hobbies, \text{Movie}), (rel, 7 : friend : 1)]$$

They quantise various values employing a *utility function* to describe how similar two values are, for example how many miles are between locations, or whether two ages are in the same discrete range. The Aprioiri algorithm is able to find frequently occurring sets of profile characteristics,

given the optimal subproblem that frequently occurring sets of characteristics in the overall dataset will also occur frequently in a randomly distributed sample.

Whilst this sort of analysis would have worked previous to Facebook API changes [17], these changes now prevent third-party applications gaining access to friends profile information unless those friends have also installed and approved the application.

However, Their results do show that similarities in these characteristics can be used to group friends in a way that is useful from the perspective of a user's privacy. The findings are roughly in line with what we would expect given the principle of homophily outlined in Section 3.5.1.

2.2 Predicting Sharing Policies for Text-based Content on Facebook

Sinha et al. [26] use a MaxEnt machine learning approach utilising *n-grams* (see Section 3.2.1) and the previously defined privacy policies of Facebook posts. Their evidence suggests that there is disparity between a user's intended privacy settings for content, and the actual privacy policy used on Facebook, so Sinha et al. asked users to manually set the privacy policies for 20 pieces of content and utilise this in the learning process of the algorithm.

They also compare this to Facebook's default privacy settings, which is to simply utilise the previously used privacy settings from the last status the user posted and to use that for the new status. Their algorithm achieves an average accuracy of 81% compared to Facebook's method achieving 67% accuracy. This motivated our choice in choosing a learning dataset that we know has ground truth in whom that user wanted to share that content with rather than relying on the policies already being utilised, as often a user fails to accurately define their intended privacy policy.

2.3 Learning to Rank Audience for Behavioral Targeting in Display Ads

Tang et al. [27] propose the most similar methodology, conceptually, to the one we have proposed. They build upon Latent Dirichlet Allocation *LDA* proposing *RankLDA*. Tang et al. are concerned specifically with users of search engines. They reason that when two users click on the same advert it signifies that their interests are similar and this similarity can be used in future when delivering search results and future advertising.

This contrasts slightly with our methodology where we reason that when two users explicitly share similar words with one another, it signifies that their interests are similar. Our model is an *information pull*, modelling users explicit participation in topics, rather than an *information push* of distributing content based on assumed demand. This is because of the nature of privacy: that we should be restricting information dissemination.

They produced an *expectation-maximisation* algorithm which is the same foundation that the more widely used *LDA* utilises. Tang et al. use both advert impressions and search terms to model a global topic distribution for both the topics associated with the search terms of a user, and the

topics of advertisements. Their algorithm considers them jointly, and in this sense the work is similar to our own work, except where we have implemented a graph based approach.

They found through combination of these two data sources, they were able to boost the predictive capabilities in both domains, boosting the quality of the topics modelled, and the click through rate of adverts on a commercial search engine.

2.4 Enhancing Text Clustering by Leveraging Wikipedia Semantics

One of the issues that we need to overcome is modelling topics within our corpus. Hu et al. [28] propose a multistep method utilising Wikipedia for text clustering. They state that each Wikipedia page title is a succinct phrase or word that is similar in structure to an entity in a conventional thesaurus and that each entity represents a topic.

The proposed clustering method is as follows:

1. Firstly, they suggest utilising the redirects and internal links on Wikipedia to resolve synonymy in a corpus. These redirections map variations in capitalisation and spelling, abbreviations, synonyms, and colloquialisms to the relevant Wikipedia article. Wikipedia have tools for looking up the redirects of a page, and we find the Gospel of Matthew has the most redirects standing at a grand total of 923. These range from alternate spellings such as "Gospel of Mathew" to abbreviations such at "Mat."[3].

2. Secondly they suggest using Wikipedia to resolve polysemy within a corpus by utilising disambiguation pages to map entities to the correct concept using Cucerzan's methodology [31], which we discuss in Section 3.2.2.

3. Finally they suggest Wikipedia categories can be utilised to assist with clustering. The hierarchy of categories present on Wikipedia can be used to create a representation of a corpus using a documents position in a hierarchy.

Hu et al. show an improvement in text clustering using this category information in a K-means clustering algorithm. We utilise some of these techniques later in Section 5.1.

2.5 A Graph Analytical Approach for Topic Detection

A slightly different approach to topic detection, and perhaps a complimentary one, is presented by Sayyadi and Raschid [29]. They propose building a co-occurrence graph between *keywords*, where nodes represent these *keywords* and edges represent co-occurrence within a document. They define *keywords* as being words, noun phrases, and named entities, which is why this method can be seen as complimentary to the previous method from Hu et al. who were using named entities but with K-means clustering.

[3] A full list of the redirects can be viewed at http://en.wikipedia.org/w/index.php?title=Special:WhatLinksHere/Gospel+of+Matthew&hidetrans=1&hidelinks=1&limit=5000

Here Sayyadi and Raschid use the Girvan-Newman algorithm for community detection, which involves calculating the betweenness centrality of edges. The betweenness centrality captures the likelihood that an edge connects two highly connected communities. The edge betweenness centrality, informally, is the proportion of shortest paths between all pairs of nodes that include this edge. Edges with a high betweenness rank will be edges between communities, and thus by removing these, eventually a graph will become disconnected, and each community will be a separate component.

Removing an edge requires the recalculation of some of the shortest paths, and thus the computation time is $\mathrm{O}\left(n^3\right)$ for sparse graphs [32]. Their results were comparable to LDA with Gibbs Sampling for the TDT4 dataset. Whilst the topic modelling literature is dominated by LDA, it is refreshing to see support for graph based topic detection. We propose a similar graph based approach in this body of work.

We focus on algorithms such as the Louvain [33] algorithm and CJ0_ZS [34] in our method instead of the Girvan-Newman algorithm given that these algorithms have a lower complexity. However, essentially all three algorithms are all similar in that they try to find communities within a graph.

3 Background

In the related work we have seen there are solutions for overcoming the problems we outline in our motivation in Section 1.1. In this section we provide detailed background knowledge on the techniques that are of particular relevance to our solution.

3.1 Survey of User Behaviour on Online Social Networks

We first survey the literature around user behaviour on three major online social networks: Google+, Twitter and Facebook. We use this survey to ground our later work, which relies on some familiarity with their usage. We also summarise our access to various APIs and how these fit into the overall objective to build a plugin to automate audience selection. Of significant importance is research by Naaman et al. [35] that suggests people who share more intimate details on Twitter than sign posting content tend to have less followers, which we feel is indicative of the findings from Pew Research [4] that 36% of people they surveyed strongly disliked other people sharing too much about themselves.

3.1.1 Behaviour on Google+

Google+ is an online social network operated by Google. It allows people to follow others, which is to say that friendship on the site does not have to be reciprocated. There is a focus on segmenting audiences with users forced to assign other users to *circles* as they connect with them. A circle is simply a common label applied to multiple users, thus a way of grouping them; users can have multiple labels and thus exist in multiple circles. When authoring a piece of content on the site, the user has to make an explicit decision as to which circles to share that content.

A profile page contains content such as videos, photos, links, events, and natural language-based *status updates*. Each item has an associated privacy policy which can include individuals, or a post can be marked as public or viewable by extended circles. There are also micro-interactions which appear on a profile, for example comments, tagging people, reposting other's content, and whether a user is attending an event or not. These micro-interactions do not have the same privacy settings attached to them; they, by default, abide by the privacy options of the *parent* content.

A profile can consist of static content, for example birthplace, gender, and relationship information. This information can also be restricted to only certain circles. Users by default have four circles: "Friends", "Family", "Acquaintances", and "Following". These four groups are based entirely on tie strength. Users have the option to delete these circles or to create new ones.

Analysis by Kairam et al. [7] shows that users create new groups for either: life facets such as school, work, or communities based around shared interests; or create groups based on tie-strength such as close family or best friends. The research also shows that content usually falls into several categories: sign posting existing content with inherent value (59% of users surveyed); sharing personal information (25.9% of users surveyed); to participate in discourse around a topic (16.9% of

users surveyed); or to evangelise work done by others (6.6% of users surveyed). Results also conclude that 79% of users share links often or always, and 82% of people share photos occasionally. People rarely share location or videos. Only 25% of users reported that they post personal information to Google+.

Google highlights that content can be distributed outside of a user's chosen privacy settings if the post is *reshared* or someone tags someone else within the content. This requires the user to take additional steps to prevent this from happening [36].

3.1.2 Behaviour on Twitter

Twitter is another online social network. Similar to Google+, it uses a directional relationship between users. Twitter limits content on the site to 140 characters, called a *tweet*, and unusually almost all interactions on the site take place within these tweets. Users are not asked who they wish to share each tweet with instead they are able to choose the visibility of all their tweets as either public or shared only with those followers who have been accepted. There is no concept of grouping friends on Twitter. However users are able to tag or *mention* someone in a tweet to explicitly state that the tweet is for their attention, although this does not limit the audience.

Twitter has very coarse privacy controls: either every piece of content a user posts can be visible to everyone (including those who do not have profiles on the network) or users can select the "protect your tweets" option, which will mean that other users need to be approved before they can see that user's content. Changing from private to public will share all of a user's content publicly, including content that user previously shared whilst the profile was private [37]. Because most content on Twitter is publicly visible, it provides lots of data to use to test our methodology with.

Content posted to Twitter usually falls into two different categories: those who share information about their own personal life and those who signpost existing content, as discovered by Naaman et al. [35]. These categories are not mutually exclusive and those users who primarily fall into one category does not prevent them from posting content fitting into the other category. This loosely fits with the conclusions of Kairam et al. [7] regarding content distributed on Google+. Whilst we do not have access to the full dataset used by Naaman et al. for this research, the examples used in the paper suggest that content coded as being about the users themselves were strongly *affective*[4]. We see the mean number of followers stands at 208 and [38] is lower than the mean number of friends a user has on Facebook at 338 [6].

The research from Naaman et al. suggests that people share more intimate details on Twitter than sign posting content, but people who signpost content tend to have larger networks. This indicates the possible existence a social cost from over-sharing private information.

Twitter diverges from Google+ in that all content is shared in the form of tweets, and each tweet (could theoretically) have a different privacy setting if Twitter implemented these settings on a per tweet level, instead of a per account level. These interactions would be considered *micro-interactions* on other networks.

[4]Affective meaning to be of, caused by, or concerning emotions

A tweet can contain links, pictures, videos or location information and tie-ins with third party services, such as SoundCloud, to enable sharing of more specialised content [39].

For many years Twitter offered almost unlimited to its API end points, letting research thrive on the platform. In recent years stringent rate limits imposed means that it is much more difficult for researchers to gain access to the vast quantities of data that they previously could [40, 41, 42]. However, our plugin only predicts within a user's close social network, so rate limiting does not significantly impact the viability the plugin, but impacts how much data we can acquire to test our methodology.

3.1.3 Behaviour on Facebook

Facebook is the online social network much of our research is based around due not only to the API being matured and stable, but because it also offers third-party applications to post content on a users behalf, whilst also specifying the privacy settings of that post.

Facebook shares many similarities with Google+, such as the ability to segment friends into friendship lists that act in a similar way to Google+'s circles. However, it is optional for users to select an audience for each status update they post. It also differs from Twitter and Google+ in that friendship is reciprocal. This concept of friendship requires one party to request friendship and the other to confirm the friendship before the connection is made.

Facebook also introduced smart lists that group together friends based on profile information such as friends who are in the same city as the user, work at the same company, or belong to the same educational institution as the user [11]. When content is posted users can choose to share content with one or more of these lists, or just "Friends", "Friends of Friends", or "Public". The activity log can update the privacy of posts retrospectively [43], as can updating a friendship list.

Jones and O'Neil questioned the usefulness of these smart groups [12] stating that of those surveyed, on average only 20% of those user's friends were segmented automatically into lists by Facebook and typically, users felt that these groups were then too incomplete to be useful for the task of implementing privacy policies for content.

Facebook does not mandate selecting an audience in the same way Google+ does. Facebook will display the audience of a post to others at a granular level or "Friends", "Friends of Friends", "Publicly" or "Custom" if the user has selected specific friendship groups amongst whom to share the content.

We were particularly interested in the API access to Facebook, which allows us to download entire chat conversation history (when permission is explicitly granted by users), and gives us access to post content on the network on behalf of users with the same level of granularity of defining a privacy policy for each status update.

3.2 Working with Natural Language

As outlined above there are different types of content that are shared on online social networks. However, as we are focusing on predicting the audiences of realtime status updates posted to a user's profile we focus on a set of techniques for dealing with natural language.

3.2.1 Preprocessing Natural Language

We cover a few different concepts for natural language processing in this section; the first concept is *stemming*. The older, more established stemming technique is Porter Stemming [44]. This process involves removing morphological and inflexional suffixes from English words using a rule based approach. To motivate the process we give an example for the word *conflated*.

The word *conflated* will match the rule "*(*V*)ED -> *". This rule is read as not a vowel followed by "ed" maps to empty string. The new stemmed word *conflat* will match the rule "*AT -> ATE*" mapping it to *conflate*. *Conflate* is the word without the inflexion *ed*.

The algorithm follows a similar procedure to this for many common suffixes, but not flawlessly. We use an off-the-shelf implementation from Umbel et al. [45]. We note the following instances of *over-truncation* and *under-truncation* with this particular implementation of the Porter Stemming algorithm:

1. *Medical* over-truncates to *Mical*;
2. *Media* over-truncates to *Mia*;
3. *Oliver* over-truncates to *Oliv*;
4. *Bibliography* under-truncates to *Bibliographi*;
5. whereas *Bibliographically* stems correctly to *Bibliograph*.

Its obvious flaws aside this technique has been applied successfully in the information retrieval field to help match terms that have different surface forms.

Another useful technique is *tokenising* [44]. Tokenising to paraphrase Martin et al. is the task of splitting input into logical chunks, usually of words. For example a simplistic tokeniser would convert *I'm visiting New York* to the words:

1. I'm
2. visiting
3. New
4. York

A more sophisticated tokeniser might even tokenise it as:

1. I
2. am
3. visiting
4. New York

We use an off-the-shelf tokeniser in the form of the aggressive tokeniser from the Natural JavaScript package [45].

Our third natural language processing technique is the bag of words *BOW* representation. This is simply a set of tokens as discovered from the tokenisation step. It no longer represents any semantic relationships between the words. An alternative representation of text that captures some semantic relationships between words are *n-grams* these are collections of adjacent words of length "n". The BOW representation of "this is a sentence" is below:

$$\{\text{is}, \text{sentence}, \text{a}, \text{this}\}$$

Which contrasts with the bi-gram representation of the same phrase below:

$$\{\text{is a}, \text{a sentence}, \text{this is}\}$$

Whilst we have ordered the sets to make the difference apparent, hopefully the reader can intuitively see that the latter representation captures more of the semantic relationships between words based on how similar it is to the original text.

LDA Topic Modelling utilises stemming, tokenising and BOW as a preprocessing step, which can improve results, as we will see in 3.2.4

3.2.2 Entity Extraction

One of the tasks within natural language processing (or *NLP* for short) is entity extraction, which is concerned with discovering words and phrases within a corpus that can be mapped to a collection of labels.

One such methodology is to utilise Wikipedia to help to extract terms from a document as presented by Cucerzan [31]. Whilst the research goes above and beyond our requirements with a sophisticated entity disambiguation technique, the research outlines a hybrid of methods for detecting entities in a given piece of text. We use the following example to motivate this method:

> Hey NYC! We'll be at Barnes and Noble on 86th & Lex this Thursday @ 7pm

Cucerzan [31] outlined the following steps for entity extraction:

1. Capitalisation Rules
   By utilising syntactic rules of a language, entities in formal documents can be extracted based on capitalisation. In the example above we are left with capitalised particles: “Hey”, “NYC”, “Barnes”, “Noble”, “Lex” and “Thursday”.

2. Exploiting Wikipedia Entities
   Wikipedia is used to resolve where two capitalised words close together refer to the one entity, so in our example, because “Barnes and Noble” has a Wikipedia entry they will be a single entity in the output.

3. Web Search Queries
   The remaining capitalised particles can be queried through an online search engine: when both particles co-occurred in a statistically significant number of the returned documents then it is assumed the words are referring to the same concept.

Within our specific domain of conversational social media messages overreliance on capitalisation would be detrimental, and submitting private conversation history to a search engine would compromise the privacy we are try to preserve. However, using Wikipedia seems like a highly viable solution.

Cucerzan evaluated the results of this method against solutions submitted to CoNLL-2003 [46]. We note other entries used approaches anywhere from utilising dictionaries like Gazetteer to utilising results from IBM’s question answering system Watson [47].

We also note that Wikipedia has also been utilised for improving the quality of clusters and we cover this research in more depth in Section 2.4.

3.2.3 Sentiment Analysis

As outlined from user behaviour in Section 3.1.2 above, there is a social cost to sharing overly personal information online with an audience that is too wide. One possible way of identifying whether content is personal is to utilise a dictionary such as LIWC [48] that contains whether a word signifies a positive emotion, or a negative emotion, amongst other classifications, and to then take an average of the sentiment expressed in a piece of text.

However, we note that according to the LIWC website, that a similar number of positive emotions are expressed in personal texts as in formal texts, scoring 2.7 and 2.6 respectively, with 2.6 and 1.6 for negative emotions in personal texts respectively. This suggests that an assumption that personal texts are more *affective* may not hold up when analysing real world data. We also note the existence of crowd-sourced affectivity ratings published by Warriner et al [49] that specifically list differences in the dimensions of affectivity for different age groups, socioeconomic status, and education levels. This supports the idea that determining whether a piece of text is personal depends on the audience and not just the words in the content.

Ideally a plugin could determine the social cost of a status being seen by the wrong audience by looking at how personal status is, but given the complexity we chose to focus on defining our probabilistic model and use a basic utility function.

3.2.4 Topic Modelling Using LDA

Every document can be thought of as being composed of words associated with several different topics. Topic analysis is a way of surfacing these latent topics [44].

One of the most well known techniques is *Latent Dirichlet Allocation* or LDA as outlined by Blei et al. [50]. This technique models the concept that documents consist of words and that these words are allocated from a small number of topics. The Dirichlet distributions are a family of continuous multivariate probability distributions, which model the probability distribution of an event occurring within a discrete set of classifications; we cover them in more detail in Section 3.4. The variables of these distributions are calculated using unsupervised machine learning, using *Expectation Maximisation* to improve the distributions of the topics over many iterations. Many software implementations of this process already exist, such as the python library "lda" [51] or the Node.js library of the same name [52]. We are using "pLDA+" by Liu et al [53], because of the speed at which it is able to process our corpus.

It has been noted by Tang et al. [54] that social network content is particularly sparse, given that the document length is often very short, a maximum of 140 characters in the case of Twitter. This has a detrimental effect on the performance of LDA which relies on matching words across multiple documents.

3.3 Community Detection

We explore two algorithms that exploit modularity to progressively refine discovered communities in a graph. Modularity is a measure of how well a graph has been partitioned into communities. Modularity is the fraction of edges between nodes of a given community, minus the expected fraction if edges were distributed at random. Communities detection algorithms based on modularity increase refine communities to exploit increases in modularity.

3.3.1 The Louvain Method for Modularity Increase

Blondel et al. discovered a fast greedy algorithm based on modularity increase to find communities in graphs, in their paper "Fast unfolding of communities in large networks" [33]. The algorithm moves nodes between communities based on the increase in modularity gained from doing so. Modularity is defined as:

$$Q = \frac{1}{2m} \sum_{\text{ij}} \left[A_{\text{ij}} - \frac{k_i k_j}{2m} \right] \delta(C_i,\ C_j) \tag{1}$$

Where i and j are nodes in a network, with the weight of the link between them as A_{ij}, m is the number of links in the network, C_k defines the community of a node k, and $\delta(C_i, C_j)$ is defined as follows:

$$\delta\left(C_i, C_j\right) = \begin{cases} 1 & i \text{ and j belong to the same community} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The purpose of the algorithm is to optimise the C mapping to partition the graph in a way that gives the highest modularity.

Blondel et al. spotted that the increase in modularity that can be achieved moving a node into a new community can be calculated. With the following definitions of the variables:

- m is the total weight of every edge in the network
- Σ_{in} is the sum of the weights of the edges between two nodes inside the destination community
- Σ_{tot} is the sum of all the edges between two nodes, where at least one nodes is inside the destination community
- K_i is the sum of the edges coincident on the node to be moved
- $K_{i,in}$ is the sum of the edges between the node and a node inside the destination community.

We can give the modularity increase as:

$$\Delta Q = \left[\frac{\Sigma_{\text{in}} + k_{i,in}}{2m} - \left(\frac{\Sigma_{\text{tot}} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{\text{in}}}{2m} - \left(\frac{\Sigma_{\text{tot}}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{3}$$

The first step is to set up the network so every node belongs to a community of its own. Each node is then taken in turn and tested against all other communities using ΔQ and then moved into the community for which it has the highest modularity increase. The algorithm continues to iterate over this step until it can no longer move any nodes for a modularity increase.

To overcome the resolution limit in community detection as discovered by Fortunato and Barthelemy [55] the Louvain algorithm upon completion of the above steps will create a new graph where each community is represented by a node, and the edges between these nodes are modelled as the summed weights of all edges spanning between nodes of the seperate communities. This process is repeated until some limit is reached such as the number of communities wanted, or there are no more steps that yield an increase in modularity (for example when there are N disconnected components, a minimum of N communities will be discovered)

The algorithm is very fast, with Blondel at al. suggesting for many sparse graphs it will run in $\mathrm{O}\left(n\log n\right)$ time.

3.3.2 CJ0_ZS Method for Community Detection

CJ0_ZS is similar to the Louvain algorithm outlined above but Rotta and Noack have made several adaptations [34]. It is similarly an algorithm based on incrementally increasing modularity in a network, but the main difference is that it is not *as greedy* as the Louvain algorithm and can produce communities that are more stable over multiple runs of the algorithm. Other changes are:

1. CJ0_ZS joins communities (CJ, for *cluster joining*) instead of moving nodes, (which is analogous to taking the set union of all nodes in two communities).

2. ΔQ is calculated from the original graph, as this merging process does not alter the underlying graph in anyway, thus the algorithm is bound by the resolution limit, as opposed to Louvain, which will keep aggressively clustering.

3. Instead of using ΔQ Rotta and Noack found the quality of their results increased when using a weighted adaptation called the Z-Score measure, which we outline below.

4. CJ0_ZS is not greedy and so will choose the maximum increase from between all communities at each step (although this is still not a deterministic process, because sometimes two pairs of clusters can be joined for the same modularity increase).

5. We do not specify a number of communities we would like to generate, we let the algorithm naturally terminate when the Z-Scores drop to 0.

Rotta and Noack define their Z-Score measure as an extension of ΔQ, where $\deg(c)$ is the degree of that node:

$$\mathrm{ZS}(C_i, C_j) = \frac{\Delta\mathrm{Q}(C_i, C_j)}{\sqrt{\deg(C_i)\deg(C_j)}} \tag{4}$$

We give their definition of ΔQ for joining two communities (as opposed from moving a node into a community):

$$\Delta Q_{C,D} \triangleq \frac{2f(C,D)}{f(V,V)} - \frac{2\deg(C)\deg(D)}{\deg(V)^2} \tag{5}$$

Where f is defined in terms of the sum of edge weights between the elements of both communities:

$$f(C,D) \triangleq \sum_{i \in C} \sum_{j \in D} A_{i,j} \tag{6}$$

Rotta and Noack indicate that this helps to balance large clusters being merged, with smaller clusters that may be equally likely to improve the modularity of the network as part of further merges. However, under ΔQ these smaller clusters will have a lower chance of being merged simply because they have a lower number of edges coincident. Dividing through by the number of degrees helps to balance the merges for both large and small clusters.

3.4 Bayesian Probability

Sometime during the late 1740s Reverend Bayes discovered a probability calculus [56] that provides a basis for changing observations into beliefs. It remained unpublished until Price discovered one of its potential applications.

We frequently use an application of Bayes' rule to a conditional probability over three variables. We give this as:

$$P\left(A \mid B, C\right) = \frac{P\left(B \mid A, C\right)\ \mathrm{P}\left(A \mid C\right)}{P\left(B \mid C\right)} \tag{7}$$

However, Bayesian probability stretches beyond this formula, as best described by *Probabilistic Programming and Bayesian Methods for Hackers* [57] as being:

> "Bayesian inference differs from more traditional statistical inference by preserving uncertainty. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving certainty from randomness? To reconcile this, we need to start thinking like Bayesians."

This is best explained with the use of a coin flipping analogy. Taking a frequentist probability we would have:

$$\begin{aligned} P(heads) =& \frac{\text{observations of heads}}{\text{total observations}} \\ P(tails) =& \frac{\text{observations of tails}}{\text{total observations}} \end{aligned} \tag{8}$$

We can see after just one coin toss, if it landed on heads, we would have the probability that the coin will land on heads being 1 (it will always happen) and the probability that a coin will land on tails is 0 (it will never happen).

However, we believe that if we were to flip the coin again, it is not impossible for it to land on tails. A better model might be to define the following probabilities:

$$\begin{aligned} P(heads) =& \frac{2}{3} \\ P(tails) =& \frac{1}{3} \end{aligned} \tag{9}$$

This is to say that, from our current observation of one coin toss, we know it is unlikely to be so biased that it will always land on heads, but we do not yet believe that the coin is unbiased. As we flip the coin more times our beliefs can and will change to reflect our knowledge.

We utilise a couple of distributions in our work that *soften* our frequencies to model that we are not always certain.

Firstly is the *Beta* distribution. We are only really concerned with $Beta\,(1,1)$, which is often referred to as expressing ignorance about any prior beliefs about a two state variable, such as a flipping a coin. We give this as:

$$X \sim Beta\,(\alpha = 1, \beta = 1) \tag{10}$$

This is sometimes referred to as the Bayes-Laplace rule, and has an expected value of:

$$E[X] = \frac{\alpha}{\alpha + \beta} \tag{11}$$

We also use a uniform Dirichlet distribution, this models ignorance as to what the correct prior beliefs are for a categorical variable. In much the same way that the Beta distribution can model no prior knowledge for a two-state variable, the Dirichlet distribution can model that we have no prior knowledge for a K-state variable, where α is a vector of size K:

$$Y \sim Dir\left(\alpha = \begin{pmatrix} 1 \\ \dots \\ 1 \end{pmatrix}\right) \tag{12}$$

The expected value of this is:

$$E[Y_i] = \frac{\alpha_i}{\sum_k \alpha_k} \tag{13}$$

This is a uniform probability of $\frac{1}{K}$ for all states.

3.5 Concepts from Neighbouring Fields

We look to fields outside of computing to look for support for our hypothesis that *register graphs* can model both friendship groups and shared language used within them. This work provides grounding for our concept.

3.5.1 Homophily

Homophily is the idea that people associate and bond with people who are similar to them.

Jones and O'Neil discovered that when selecting an audience for a post a user weighs up several factors [12] (ranked by number of surveyed users who thought they were a factor):

1. social circles,
2. tie strength,
3. temporal episodes,

4. geographical locations,
5. functional roles,
6. organisational boundaries.

These criteria encompass the default smart lists within Facebook, which means that how users reason about groups is similar to Facebook's default smart lists. Users do not use this functionality on Facebook because the smart lists seem incomplete to users not because the concept of groups is inherently incorrect [12].

Within the literature, McPherson lists four main structural causes of homophily [58], which are:

1. geography (it is more likely for a user to be close to another user who is close geographically);
2. family ties (tie strength is very strong between users who are kin);
3. organisational foci (school, work, and voluntary organisational foci provide the great majority of ties that are not kin);
4. isomorphic sources (ties based on sharing a mutual occupational, family, or informal role).

We can see that these factors are a superset of the factors Jones and O'Neil outlined in their research.

Aside from structural sources of homophily McPherson also discusses cognitive processes resulting in homophily, specifically the concept that people who share similar knowledge are more likely to interact and therefore to have stronger social ties.

Han et al. conducted a study of homophily across online social network users [59] and discovered that there is a correlation between structural homophily between two users and their shared interests. The research also finds a correlation between the similarity of two user's friends and their shared interests. We can no longer explore friend's interests expressed through likes through the Facebook API. However, the correlation supports our hypothesis that our application of clustering on our register graph will work as effectively in place of current solutions using the social graph.

3.5.2 Ingroups and Registers

Building on homophily is the concept of an *ingroup*. This is defined as a social group that a person feels they belong to. In previous work identifying friendship groups from the *social graph* of a user; an analogy would be that homophily is the edge between two users that represents friendship and an *ingroup* represents a highly connected subgraph of these users.

Ingroup affiliation comes with its own practices, knowledge and language. It is the latter that is of particular relevance to our approach. A closed register [60] is a set of vocabulary, dialects, jargon or slang that is not commonly used or understood to those who are unaffiliated with the ingroup that

uses it. These registers can be used to affirm ingroup affiliation, and to create barriers between the ingroup and non-members. We have four sources of homophily: geographical, family based, organisational, and isomorphic. We similarly see variation in language based on geography, with MIT conducting research into mapping the variations of English with their online game Which English[5]. ISO 12620 defines a category of registers specifically for in-company language. And there is anecdotal evidence that language varies between families and informal roles. Homophily and the language used within the groups it forms are causally linked.

We focus closely on one aspect of registers: jargon, which is vocabulary shared within a group, but not external to it. It is designed to aid communication between participants in a conversation, but has an unintended side affect that people who are not part of that ingroup may not understand the terminology. Therefore when this language is explicitly sent from one person to another it signifies that the originator knows the recipient is part of the same ingroup and so can understand the language being used.

If Latent Dirichlet Allocation is built on the basis that similarities within documents occur because documents consist of words from topics and each document is made up of a mixture of topics. Then our work is built on the basis that documents consist of words from multiple registers, and that each document consists of a small subset of registers. Our hypothesis is that these documents are a product of both their author's membership of specific ingroups and the registers used amongst them.

[5]Which English is available online http://www.gameswithwords.org/WhichEnglish/

4 Gatekeeper: an Overview of the Approach

We identified three tasks and solved them in order to predict audiences for online social network status updates, these are:

1. Entity extraction, to reduce the computation required to generate the topic models;
2. Topic modelling, in order to build an understanding of the differences in language used in different social groups;
3. Audience prediction, based on our probabilistic model enabling us to make predictions from the topic models we have built.

Throughout this body of work we focus on these three main tasks for our algorithm, giving methodology, implementation details and evaluating each of the tasks as well as the overall algorithm.

Our topic modelling approach considers both the vocabulary and the audience of each document jointly and enables us to boost the quality of the topic model, and thus to increase the accuracy with which we can predict audiences of online social network content.

We build a plugin that runs on end user's commodity hardware and so we focus on building a robust and fast implementation of our algorithm, so as to enable end user's to make privacy policy decisions per status update without requiring them to manually select each friend they wish to share a status update with.

5 Methodology

We have broken the broad task of predicting audiences for natural language status updates into three subtasks. The first task we present in our methodology is entity extraction, in Section 5.1. In the two other tasks, entity extraction allows us to focus on the words that are most likely to capture the topic of a status update. Our hypothesis is that certain topics are only discussed amongst specific *ingroups*, so reducing noise in topic models is critical for improving the accuracy of predicting audiences for status updates.

The second task is to model topics. We have several approaches that we have experimented with and we present the methodology for each in Section 5.2.

Finally in Section 5.3 we present the probabilistic model we use for making audience predictions from these topic models.

5.1 Entity Extraction

Entity extraction allows us to focus exclusively on the words in a document that are most likely to communicate the topic of that document. Using entity extraction makes it more tractable to predict audiences for status updates on the commodity hardware of end users of our plugin.

We were particularly inspired by the work from Hu et al. [28] and Cucerzan [31] on entity extraction utilising Wikipedia as covered in Sections 2.4 and 3.2.2 respectively. Given that Sayyadi and Raschid's [29] (Section 2.5) methodology exploited entity extraction as part of their graph based topic approach, we experimented to see what optimisations we could make around storage space and speed when extracting entities from a corpus, with the specific aim of creating an approach that will work on the commodity hardware that our plugin will need to run on.

In order to extract entities utilising Wikipedia, we need to process a data dump of the Wikipedia database, which is made available weekly. Figure 4 outlines a subset of redirects on Wikipedia, from redirection page titles to the page titles of the entities they represent. A redirect such as http://en.wikipedia.org/wiki/Zinc_supplementation will redirect the browser to the main article, which in this case is a page on Zinc. We use these as the basis for our entity extraction.

Given an input of *Australia* we ideally want to map this back to the page title *Australia*. Given an input of the misspelling *Austrlia*, we would also want to map this to the page *Australia*.

Given that a page title on Wikipedia represents the name of an entity, we can utilise these to discover entities on Wikipedia. We can say that redirects are synonymous with the page they point to. Thus the redirects, which represent either other names for the same entity, misspellings or colloquialisms, can be used to resolve synonymy within our corpus.

Wikipedia page titles sometimes contain symbols or capitalisation, given the formal writing style of Wikipedia, but we do not expect this formality in an informal corpus [23]. We need to map input text to a canonical form that normalises the character set used and the capitalisation. We also wish to normalise the spacing between words. Because we are working specifically in the domain of

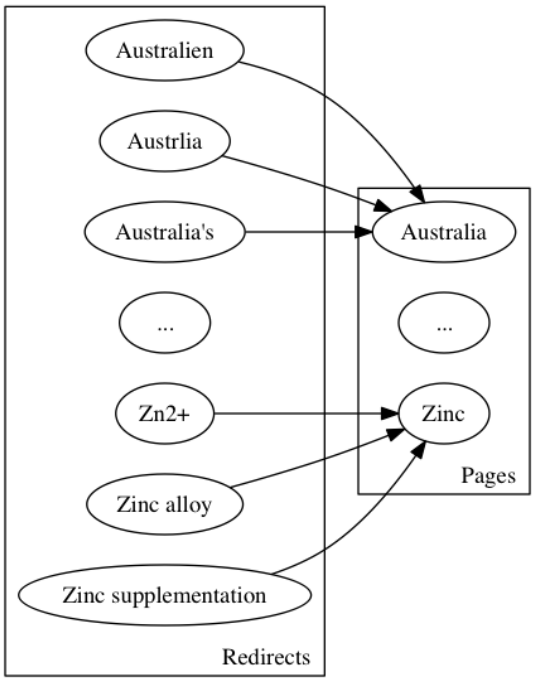

Figure 4: Subset of redirect structure in Wikipedia between redirect titles and page titles

the English language we can summarise that word boundaries fall on *non-word symbols* such as a space, a tab or a newline [44].

Given the informal nature of our corpus, we take this to the extreme that any non-word symbol is a word boundary. This means we can normalise space between word boundaries, we also lowercase the input during normalisation. This means given any text we can represent it in normal form using the characters a-z, the numbers 0-9, and the space character. A visualisation of the pipeline is given in Figure 5, motivating the process for a couple of entities on Wikipedia. Our input is first tokenised removing all non-word symbols and the spacing between words is normalised, the next step is to map these to a entity that is all lowercase.

We would like to map both normalised redirect titles, and normalised page titles to the Wikipedia entity, represented by the original full page title. This would allow us to utilise information from the Wikipedia entities to resolve polysemy, if needed, or as a source of additional information about an uttered entity.

We visualise the process of canonicalising Wikipedia page titles and redirect titles in Figure 6. We can see that this is a surjective function, and so there is no precise inverse for mapping from normalised terms back to the Wikipedia entities. We create a function, such that:

$$\begin{aligned} &lookup : \text{normalised text} \mapsto \mathcal{P}(\text{Wikipedia entity}) \\ &\forall wiki_{entity}.(wiki_{entity} \in lookup(normalise(wiki_{entity}))) \end{aligned} \tag{14}$$

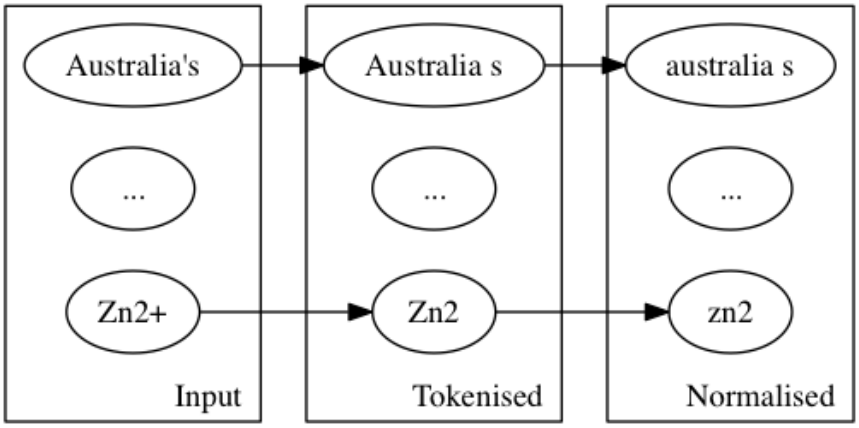

Figure 5: Normalisation process for the English language

The simplest implementation of this function would be to map everything on to a set containing all Wikipedia entities. Thus we introduce an ordering based on set inclusion, and say that *lookup* should return the smallest sized set that still satisfies Equation 14.

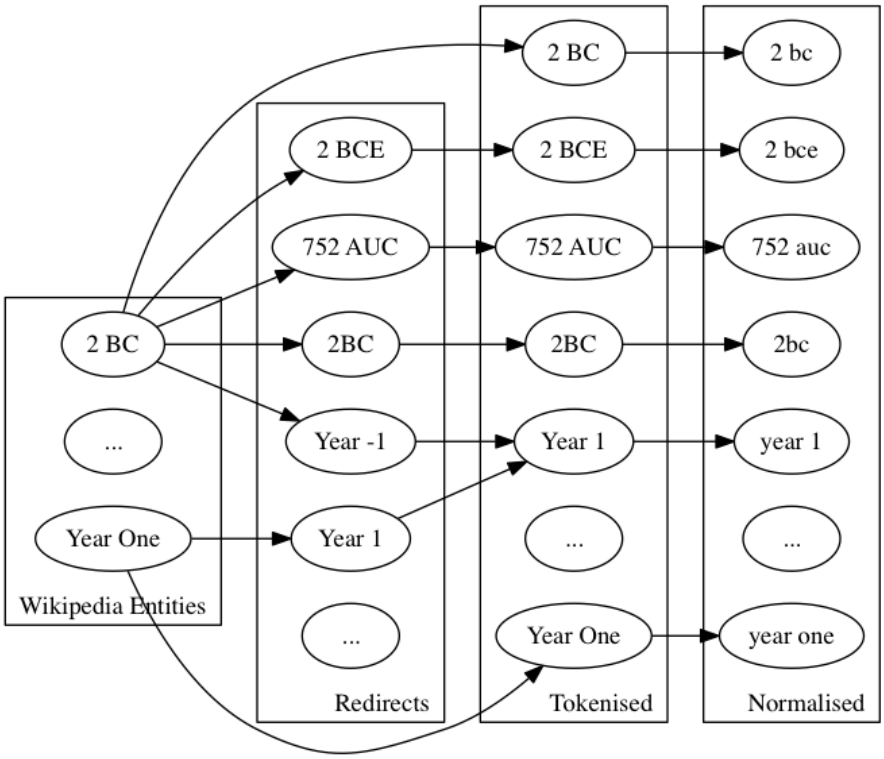

Figure 6: Example of Wikipedia entities being normalised

Now we have a dictionary in the form of *lookup*, we need to utilise it to extract entities from a piece of text. We use a sentence from our sample to motivate this process:

> Ooh, this is a particularly nice interest rate. When interest rates get you excited you know you're a real adult :P

We utilise exactly the same normalisation and tokenising scheme as laid out above. We refer to input that has been processed in this way as *text* in the later definitions. For our example we have:

> ooh this is a particularly nice interest rate when interest rates get you excited you know you re a real adult p

We can immediately see a couple of entities such as *adult* and *interest rate*. To be able to extract these entities we need divide our input text into various length *n-grams* and pass them through *lookup*. When *lookup* returns a set whose size is greater than 0, we know that the *n-gram* represents an entity.

We define $take(n, text)$ such that it will return the first n words from an input $text$. We define $drop(n, text)$ such that it will return the last $|text| - n$ words, where $|text|$ means the number of words in $text$. We define $extract(n, text)$, which will extract all entities of length n in $text$, given in terms of $take$ and $drop$:

$$extract(n, text) = \begin{cases} take(n, text) \cup extract(n, drop(1, text)) & |lookup(take(n, text))| > 0 \\ \emptyset & |text| < n \\ extract(n, drop(1, text)) & \text{otherwise} \end{cases} \tag{15}$$

In order to extract entities of any length we start with $extract(|text|, text)$ and next we remove any *entities* we discovered from the text before running the process again; taking the set union of the results:

$$extract(|text|, text) \cup extract(|text| - 1, text/extract(|text|, text)) \tag{16}$$

This is best explained diagrammatically in Figure 7 for a small subset of our example text *... particularly nice interest rate when ...*. Where we give the *interest rate* node in grey to signify that it has a non-empty result for *lookup*(interest rate). We take the set union of all *grey* results as our extracted entities, and we also note that when an entity is found, no subset of the words in that noun phrase are processed for any extract with a lower n value.

Theoretically, extracting entities from a document runs in $O(n^2)$ time, where n is the number of words in the input. If our dictionary is sorted we can use a binary search for looking up entities within the dictionary, thus if a binary search lookup is required for each fragment *extract*ed from our input, the complexity is $O(n^2 \log d)$ where d is the size of our dictionary.

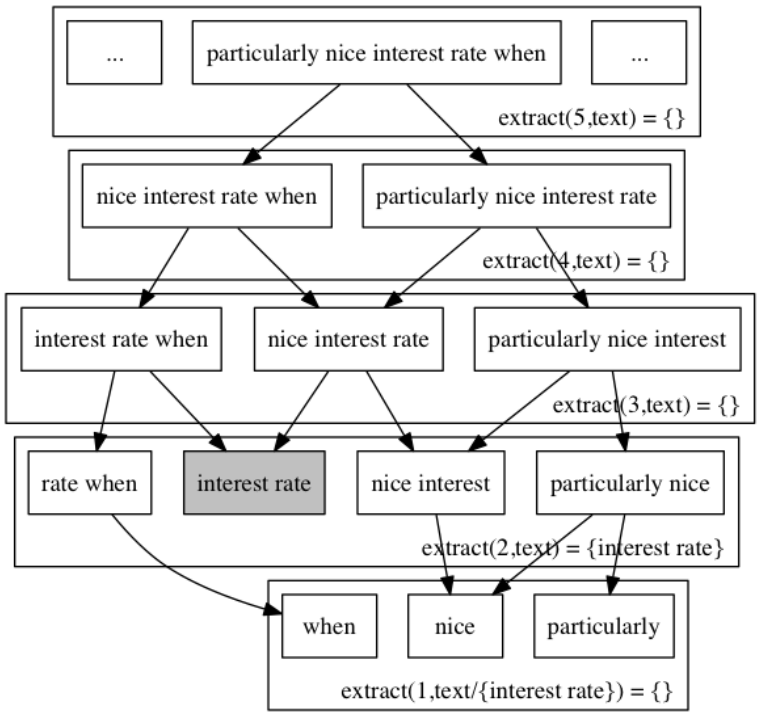

Figure 7: A diagram motivating *extract*

5.1.1 Testing Methodology

We evaluate our implementation against that of T-Seg(None) from Ritter et al. [61] (who in turn evaluated their results against Stanford NER [62]). We use an annotated Twitter dataset [63]. We expect a drop in precision when compared against Stanford NER and T-Seg(None) given that we are utilising every entity on Wikipedia. We could use a curated dictionary, or utilise a Part-of-Speech tagger, as Ritter et al. did, which can help to remove false positive results.

Our results and those of T-Seg and Stanford NER are given in 7.1.

5.2 Topic Modelling

We have outlined three different tasks that need to be completed in order to make accurate audience predictions for the status updates from the users of our plugin. Our hypothesis is that certain language is specialised and will only be shared amongst groups of friends that belong to the same *ingroup*. We can utilise topic modelling in order to discover these groups of friends and the restricted language used within them. We take advantage of our entity extraction methodology outlined in the previous section, Section 5.1, and in turn our topic models are utilised by our audience prediction method, which we give in the next section, Section 5.3.

These three tasks form the core of our method for predicting audiences for pieces of content shared on online social networks. Our plugin will show users these predictions enabling them to make

swift decisions as to who exactly they want to share a status update with.

Existing solutions such as LDA (Section 3.2.4), which is often given as the baseline state-of-the-art [22, 23, 27], utilises a generative probabilistic model that explains how a document is composed from a mixture of topics consisting of different words. By utilising an Expectation-Maximisation algorithm these probability distributions can be refined until the probabilities have converged to a distribution that best fits the corpus of documents. Sayyadi and Raschid [29] utilised a graph based approach for topic modelling, where words are nodes in a graph and the edges between them represent cooccurrence in documents in the corpus.

We know from Tang et al. [27] that adding in additional, complementary data will boost the performance of both predictions and topic modelling. In the instance of Tang et al. utilising logs from advert clicks to boost the performance of topic modelling within a commercial search engine. In this section we present a novel approach to building topic models utilising our hypothesis that topics can be modelled by the groups of users and the specialised language they share within them.

Section 3.5.1 outlined supporting material for our hypothesis from the social sciences. Our hypothesis being that topics can be modelled by analysing the vocabulary (*registers*; Section 3.5.2) used within subsets of users (*ingroups*; Section 3.5.2). The underlying assumption is that similarities within documents in a corpus occur because documents consist of words from multiple *registers*; and that each document consists of a mixture of *registers*. This is a stronger assertion because *registers* are formed from jargon that is shared within specific *ingroups*. Topics contain domain specific language and that the use of that language implies that both the author and recipient have some affiliation or authority within that domain. Our hypothesis is that by utilising this additional information we can boost both the quality of the topics discovered, and the quality of audience predictions we can make using these topic models.

We capture that when two different users send of receive the same entity in a document; that they must share a social tie; and that the more shared usage of entities in turn connotes a stronger tie by utilising a bipartite graph. A bipartite graph can be partitioned into two sets of nodes where nodes in each set have no edges to other nodes in the same set. An example bipartite *register* graph is given in Figure 8, we represent entities as boxed nodes, and publishers as circular nodes. The edges between these nodes represent that a document exists, published by that user or mentioning that user, and mentioning that entity.

We can use community detection algorithms over this graph to find clusters where a group of authors share a large amount of language in common, thus suggesting that those words form a *register*, and that the shared usage of that *register* implies that these authors belong to the same *ingroup*. Whilst Sayyadi and Raschid's methodology utilised removing edges with a high betweenness centrality (Section 2.5 covers this in more detail) there are many algorithms for community detection.

We opted to use community detection algorithms that progressively increase modularity across the graph. The modularity is a measure of how well a network has been divided into groups/clusters/-communities. These algorithms work by measuring the fraction of edges between nodes of a given

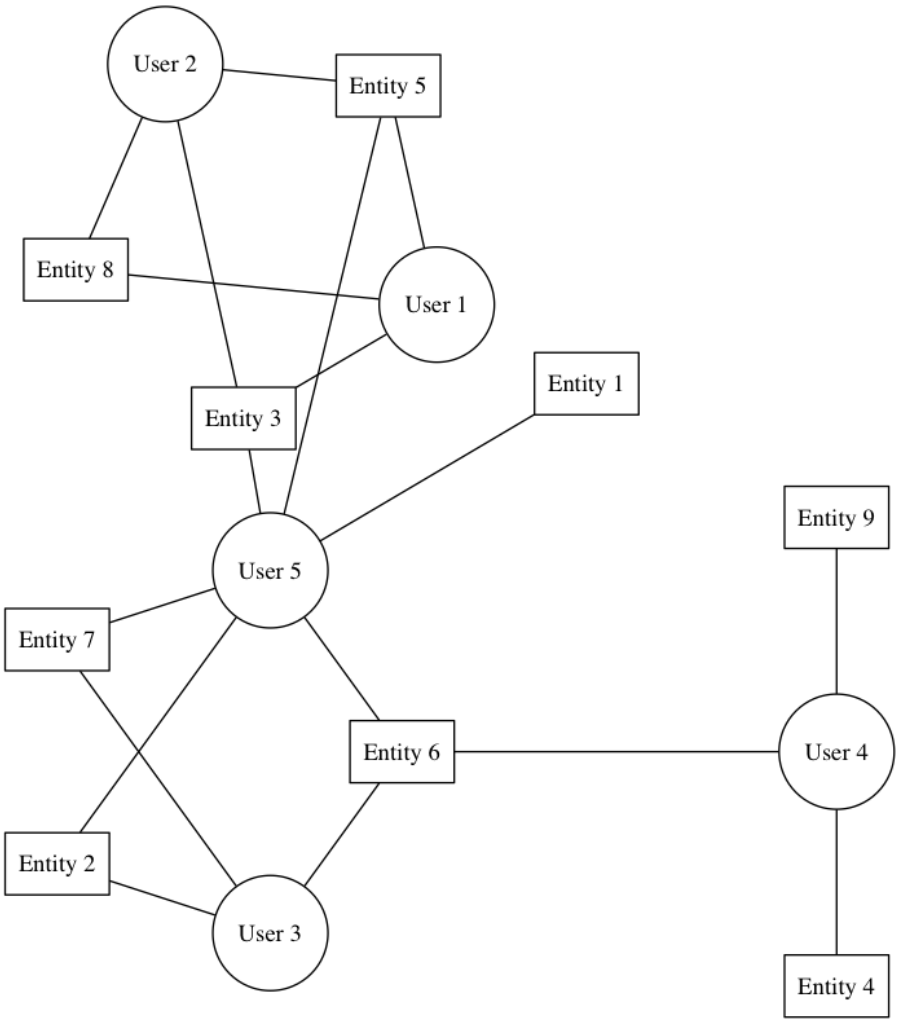

Figure 8: An example bipartite *register* graph

community, minus the expected fraction if edges were distributed at random.

By using a global null model as a basis for comparison, modularity increase algorithms are prone to "over-cluster" large networks, failing to discover smaller communities with high connectivity. We note that these algorithms are thus only appropriate for small networks. We are unlikely to generate large networks given our domain of social network messages, which tend to be very small in terms of the number of entities per document, and the number of documents available. It is especially limited in our model because we are only building a topic model for the current user. However, modularity increase based community detection tend to have lower complexity [32], and so are perfect for running on end-user's commodity hardware.

Using a bipartite graph allows us to include users in the communities that are detected, whilst we do not utilise these communities of users in this work, these friendship groups enable previous privacy solutions based on these friendship groups to continue to work [14, 15, 16]. Given that these previously relied on mining the friendship lists of many users using the Facebook API endpoint

"/user/friends", which is now restricted due to the privacy issues it creates.

We explore two modularity increase based community detection algorithms, with differing levels of complexity and different tradeoffs:

- Louvain
- CJ0_ZS

Then we evaluate the quality and stability of the topics generated in Section 5.2.6. Where we measure quality *by proxy* of whether the algorithms classify users into the same communities as a known ground truth. Stability is an important quality for this method: we do not wish to create radically different topic models when the algorithm is rerun as this would confuse a user, thinking that their social groups are unnaturally unstable.

5.2.1 Our Model of a Social Networking Corpus

We define a corpus of documents as $\mathcal{D}$, which consists of some entities (utilising our *lookup* function from Section 5.1) from the global set of all entities from all documents, which we call $\mathcal{N}$.

In this usage of a document we are restricting the contents to the entities as discovered by using the *lookup* method on the contents of documents. However, our model is flexible enough to work for a vector of words or tokens. By only working with entities we reduce the number of nodes in our bipartite *register* graph and thus speed up our community detection algorithms at a cost of removing *slang*, *colloquialisms* and *dialects* from our corpus that could also indicate *ingroup* membership.

We define a document as d_i ($d_i \in \mathcal{D}$) as containing either the presence of an entity, or lack of it as:

$$\mathbf{n}_i : \{\top, \bot\}^{|\mathcal{N}|} \tag{17}$$

Where $\mathbf{n}_{ij} = \top$ represents the presence of entity j a in document d_i, and $\bot$ represents the lack of the entity in the document.

We define the audience of a document as $\mathbf{m}_i$. Similarly it is a vector indicating the presence, or lack thereof, of each user in a global set of users which we have denoted $\mathcal{U}$.

$$\mathbf{m}_i = \{\top, \bot\}^{|\mathcal{U}|} \tag{18}$$

Where $\mathbf{m}_{ij} = \top$ represents that j is part of the known audience of document d_i, and $\bot$ represents that it is not known if j was audience to that document.

h_i denotes the publisher of the document d_i where $h_i \in \mathcal{U}$.

Thus our documents have the form:

$$d_i : h_i \times \mathbf{m}_i \times \mathbf{n}_i \tag{19}$$

Our corpus, $\mathcal{D}$, is the set of all these documents.

There is an assumption for our *register* graph model that publishers and audiences are part of groups. Our method works by discovering these latent *ingroups*.

We can build a bipartite graph with vertices as follows:

$$V = \{\mathcal{U}_1, ..., \mathcal{U}_{|\mathcal{U}|}, \mathcal{N}_1, ..., \mathcal{N}_{|\mathcal{N}|}\} \tag{20}$$

We define that a user u and entity n are adjacent if and only if a document exists such that: a document includes that user either as a publisher or in its audience and includes the entity:

$$u \ \sim \ n \Leftrightarrow \exists i(h_i = u \wedge \mathbf{n}_{ij} = \top \wedge \mathcal{N}_j = n) \tag{21}$$

Thus, the graph used later for community detection is the undirected graph:

$$G = (V, \{(\mathcal{U}_i, \mathcal{N}_j) \mid i \in \{1, ..., |\mathcal{U}|\}, j \in \{1, ..., |\mathcal{N}|\}, u \sim n\}) \tag{22}$$

An example of such a graph was given previously in Figure 8.

We let $\mathcal{T}$ be the set of all topics a community detection algorithm has discovered, and thus let C represent the raw results from our community detection algorithm for all nodes $U \cup N$:

$$C \ : V \ \times \ \mathcal{T} \ \mapsto [0, 1] \tag{23}$$

We use the convention of using lowercase t to represent a topic in $\mathcal{T}$.

This captures the notion that C maps elements and respective topics to a natural number representing how much users and entities belong to that topic. This definition is flexible enough to represent our results of both community detection and LDA. We see later for community detection algorithms, we do not have the concept of overlapping communities.

Motivating this using Figure 8. We can observe roughly 3 communities in this figure by eye:

1. User 2, User 1, Entity 3, Entity 5, Entity 8, potentially Entity 1 and potentially User 5.
2. User 3, Entity 2, Entity 7, potentially Entity 1, potentially Entity 6, and potentially User 5.
3. User 4, Entity 9, Entity 4, and potentially Entity 6.

We give a possible C mapping for Figure 8 in Table 1. We note that for our community detection algorithms each element is hard assigned to a community, therefore we would not expect 0.5 as a result for the community of a node and topic pair.

C(v,t)	t=1	t=2	t=3
v=User 1	1.0	0.0	0.0
v=User 2	1.0	0.0	0.0
...	...	...	...
v=User 5	0.5	0.5	0.0
v=Entity 1	0.5	0.5	0.0
v=Entity 2	0.0	1.0	0.0
...	...	...	...
v=Entity 6	0.0	0.5	0.5
...	...	...	...
v=Entity 9	0.0	0.0	1

Table 1: Motivating C using example data

We define two functions over C: C_U and C_N that help us more intuitively use the results of the community detection. $C_U(t)$ represents a set of users that have been labeled as being part of topic t. $C_N(t)$ represents a set of entities that have been labeled as being part of topic t.

$$C_U \ : \mathcal{T} \mapsto \mathcal{P}(U) \tag{24}$$

$$C_N \ : \mathcal{T} \mapsto \mathcal{P}(N) \tag{25}$$

Such that:

$$C_U(i) = \{u \mid u \in U,\ C(u,i) > 0\} \tag{26}$$

$$C_N(i) = \{n \mid n \in N,\ C(n,i) > 0\} \tag{27}$$

Using our example from 8 we can give C_U as:

1. $C_U(t=1) = \{\text{User 1}, \text{User 2}, \text{User 5}\}$
2. $C_U(t=2) = \{\text{User 3}, \text{User 5}\}$
3. $C_U(t=3) = \{\text{User 4}\}$

We give C_N as:

1. $C_N(t=1) = \{\text{Entity 1}, \text{Entity 3}, \text{Entity 5}, \text{Entity 8}\}$
2. $C_N(t=2) = \{\text{Entity 1}, \text{Entity 2}, \text{Entity 6}, \text{Entity 7}\}$
3. $C_N(t=3) = \{\text{Entity 4}, \text{Entity 6}, \text{Entity 9}\}$

We explore Louvain, CJ0_ZS and LDA as possible algorithms for building topic models from our corpus in the next three sections. These algorithms will define the mappings C, C_U and C_N.

5.2.2 The Louvain Algorithm as a Method for Topic Modelling

The Louvain algorithm (introduced in Section 3.3.1) extracts communities by increasing modularity [33]. The complexity is $O(n \log n)$ where n is the number of nodes in our network.

We diverge from the original variable names Blondel et al. used, by using L instead of C to define the mapping of vertex to community. This is to be explicit that this mapping does not behave in the same way as our C, as defined previously in Section 5.2.1.

Recall in Section 3.3.1 that we gave the following definitions for variables:

- m the summed weight of all edges in the graph
- Σ_{in} is the sum of the weights of the edges between two nodes inside the destination community
- Σ_{tot} is the sum of all the edges between two nodes, where at least one nodes is inside the destination community
- K_i is the sum of the edges from node i
- $K_{i,in}$ is the sum of the edges between the node and a node inside the destination community
- A_{ij} is the weight of the edge between nodes i and j

Recall the equation for the modularity of a graph is:

$$Q = \frac{1}{2m} \sum_{\text{ij}} \left[A_{\text{ij}} - \frac{k_i k_j}{2m} \right] \delta(L_i,\ L_j)$$

Where i and j are nodes in a network, with the weight of the link between them is A_{ij}, m is the number of links in the network, L_k defines the community of a node k, and $\delta(L_i, L_j)$ is defined as follows:

$$\delta\left(L_i, L_j\right) = \begin{cases} 1 & i \text{ and j belong to the same community} \\ 0 & \text{otherwise} \end{cases}$$

Thus the original equation for the delta in modularity when moving a vertex i to a new community, as discovered by Blondel et al. [33] is:

$$\Delta Q = \left[\frac{\Sigma_{\text{in}} + k_{i,in}}{2m} - \left(\frac{\Sigma_{\text{tot}} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{\text{in}}}{2m} - \left(\frac{\Sigma_{\text{tot}}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right] \tag{28}$$

Notice that, given the number of links in our network never changes, this is equivalent to:

$$\Delta \text{Q} \propto m k_{i,in} - \Sigma_{\text{tot}} k_i \tag{29}$$

This shorter equation can be calculated much faster, and given this is in our busiest loop, this can have a big impact on the speed of the algorithm.

We give a step by step break down of the algorithm below for community detection over our bipartite *register* graphs. Firstly given a set of nodes V, and edges E, we define L_i such that every node i is assigned a unique community that contains only that node:

$$L_i = c \text{ where } \forall i' \in \mathrm{V}. \left(L_{i'} = c \Rightarrow i = i'\right) \tag{30}$$

We define explicitly the following functions:

$$k_{i,in}\left(i, L_i\right) \triangleq \sum_{j \in V} \delta(L_i, L_j) A_{i,j} \tag{31}$$

$$k_i(i) \triangleq \sum_{j \in V} A_{i,j} \tag{32}$$

$$\Sigma_{\mathrm{tot}}\left(L_i\right) \triangleq \sum_{j \in V} \Delta\left(L_i, L_j\right) \sum_{k \in V} A_{j,k} \tag{33}$$

$$m \triangleq \sum_{(i,j) \in E} A_{i,j} \tag{34}$$

These allow us to express modularity increase as:

$$\Delta Q(i, L_j) \propto \left(1 - \delta\left(L_i, L_j\right)\right) \left[m k_{i,in}\left(i, L_j\right) - \Sigma_{\mathrm{tot}}\left(L_j\right) k_i\left(i\right)\right] \tag{35}$$

Step 1

For the initial step in the Louvain algorithm, we follow the Algorithm 1, where we specify the number of communities we want to find as *nCommunities*, and where we use the previous definitions for ΔQ, V, L and A. We return L (the mapping of nodes to communities) when improvement is false (there is no vertex that can be moved to increase modularity), or the number of unique communities has dropped below *nCommunities*, otherwise we proceed with the algorithm. We explicitly ensure that walking over the nodes is non-deterministic.

Step 2:

For the second step of the algorithm we introduce the concept of a fake node: it contains a set of nodes; the weight of the edge between two fake nodes is the sum of all edges between the sets of nodes in both fake nodes. Setting L_n of a fake node sets the L_i of all child nodes. We create a new graph that consists of these fake nodes and a new communities mapping L', where we create a fake node for each community discovered so far.

$$F \ : \mathcal{P}(N) \tag{36}$$

F is simply a set of nodes.

$$V' = \{F_j \mid \ F_j = \{i' \mid i' \in V, \ L_{i'} = j\}\} \tag{37}$$

Our $V^{'}$ is a set of these *fake nodes*.

Algorithm 1 Pseudocode for step 1 of the Louvain algorithm

```
improvement ⇐ false
for all i ∈ V do
  if |{L_n' | n' ∈ V}| ≤ nCommunities then
    return L
  end if
  old community ⇐ L_i
  cost ⇐ ΔQ(i, old community)
  max ⇐ 0
  new community = old community
  for all c ∈ {L_n' | n' ∈ V} do
    modularity increase = ΔQ(i, c) − cost
    if modularity increase > max then
      max ⇐ modularity increase
      new community ⇐ c
    end if
  end for
  if max > 0 and new community ≠ old community then
    L_i ⇐ new community
    improvement ⇐ true
  end if
end for
if improvement = false then
  return L
else
  return step2(V, A, L, nCommunities
end if
```

$$A'_{F_i,F_j} \triangleq \sum_{i' \in F_i} \sum_{j' \in F_j} A_{i',j'} \tag{38}$$

The edge weights between *fake nodes* is the sum of edge weights between each pair of nodes in each fake node set.

$$E' \triangleq \{(i,j) \mid i \in V', j \in V', A_{i,j} > 0\} \tag{39}$$

We create an edge between fake nodes, where edges exist between two nodes in the original graph.

$$L'_i = c \text{ where } \forall i' \in V'. (L'_{i'} = c \Rightarrow i = i') \tag{40}$$

We then rerun our step 1 algorithm (Algorithm 1) with our new $E^{'}$, $V^{'}$, $A^{'}$ and $L^{'}$. This algorithm iterates until there are exactly the required number of communities, $nCommunities$, or there is no modularity increase when moving any vertex.

This concept of a fake node is best illustrated diagrammatically in Figure 9. Where we present a possible clustering of 8 and how this corresponds to a graph of fake nodes. We can see at the end of step 1, the following communities have been discovered:

$$f_{t=1} = \{\text{Entity } 8, \text{Entity } 5, \text{Entity } 3, \text{User } 1, \text{User } 2\}$$
$$f_{t=2} = \{\text{Entity } 1, \text{Entity } 2, \text{Entity } 7, \text{User } 5, \text{User } 3\}$$
$$f_{t=3} = \{\text{Entity } 6, \text{Entity } 9, \text{Entity } 4, \text{User } 4\}$$

We create a *fake node* for each community, in this instance represented by the sets $f_{t=1}$, $f_{t=2}$, and $f_{t=3}$. Because there are edges between nodes in $f_{t=1}$ and $f_{t=2}$ we create a link between these two fake nodes. There are edges between nodes in $f_{t=2}$ and $f_{t=3}$ therefore in our new graph there is an edge between these two fake nodes. We sum the weights of links between the nodes in the sets of fake nodes, and this leads us to assign a weight of two to both edges in our new graph, because there were two edges between each with a weight of one.

Humans recall social groups more efficiently when they are triadic in structure [13], and that this forms the basis of how friendship groups are formed. This gives us a rough estimate that the maximum number of communities, $nCommunities$ will be roughly a third of the number of friends a user has.

Our $C(v,t)$ function gives us the probability that an entity or user v belongs to a topic t as outlined in Section 5.2.1.

The results of the Louvain algorithm can be given in the $C(v,t)$ form, by using the following definition:

$$C(v,t) = \begin{cases} 1 & L_v = t \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

By definition, our entities and users are hard associated to a topic.

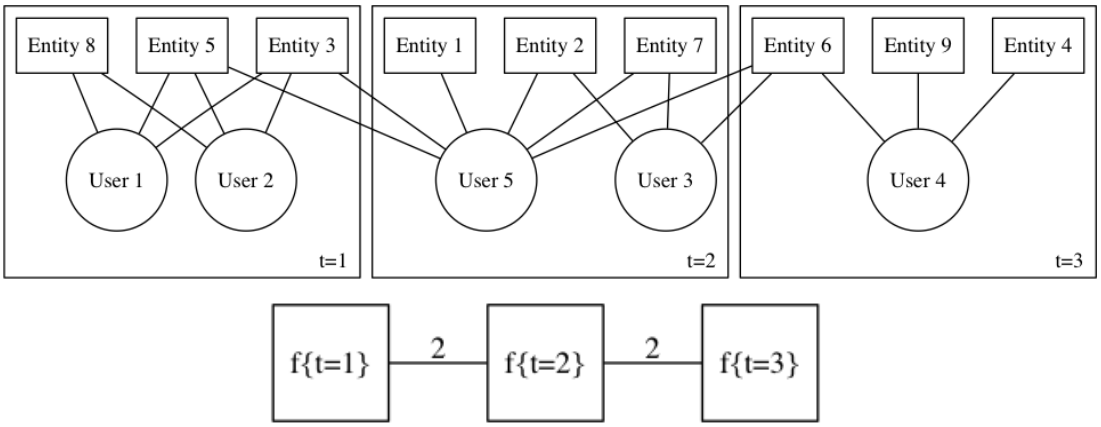

Figure 9: First graph, a possible clustering for Figure 8. Second graph: the graph presented as a graph of fake nodes.

5.2.3 The CJ0_ZS Algorithm as a Method for Topic Modellling

Whilst the Louvain algorithm is quick, the communities it detects are unstable as covered later in the evaluation of topic modelling. We implement CJ0_ZS as described by Rotta and Noack [34] and introduced in Section 3.3.2. This algorithm has a higher complexity than the Louvain algorithm but the communities it generates are more stable over multiple runs.

Recall the differences of this algorithm in comparison to the Louvain algorithm:

1. CJ0_ZS joins communities (CJ, for *cluster joining*) instead of moving nodes, (which is not the same as creating a fake node, but instead just changes the mapping L and is analogous to taking the set union of all nodes in two communities).
2. ΔQ is calculated from the original graph, as this merging process does not alter the underlying graph in anyway, thus the algorithm has a natural bound on the minimum number of communities it will return, as opposed to Louvain, which will keep clustering.
3. Instead of using ΔQ Rotta and Noack [34] found the quality of their results increased when using a weighted adaptation called the Z-Score measure, which we outline below.
4. CJ0_ZS is not greedy and so will choose the maximum increase from between all communities at each step (although this is still not a deterministic process, because sometimes two pairs of clusters can be joined for the same modularity increase).
5. We do not specify a number of communities we would like to generate, we let the algorithm naturally terminate when the Z-Scores drop to 0.

Rotta and Noack [34] define their Z-Score measure as a weighted variant of ΔQ, where $\deg(c)$ is the degree of a node c, as follows:

$$\mathrm{ZS}(C_i, C_j) = \frac{\Delta\mathrm{Q}(C_i, C_j)}{\sqrt{\deg(C_i)\deg(C_j)}} \tag{42}$$

We present Rotta and Noack's [34] ΔQ formula for calculating modularity increase for joining two clusters, as follows:

$$\Delta Q_{C,D} \triangleq \frac{2f\left(C,D\right)}{f\left(V,V\right)} - \frac{2\deg\left(C\right)\deg\left(D\right)}{\deg\left(V\right)^2} \tag{43}$$

Where f is defined in terms of the sum of edge weights between the elements of both communities:

$$f(C,D) \triangleq \sum_{i \in C} \sum_{j \in D} A_{i,j} \tag{44}$$

We note that ΔQ is only > 0, when the communities have at least one edge between them, for this reason we are able to make a time saving and space saving optimisation by only storing the Z-Score values for merging two communities when those communities share an edge.

We also store all Z-Score values in a priority heap ZSs, which allows us to get the maximum element in $\mathrm{O}\left(\log n\right)$ time. Given that we have a maximum of $|V|^2$ elements in our heap for each pair of communities, it will take $\mathrm{O}\left(\log |V|\right)$ time to get the maximum Z-Score once we have populated the initial priority heap with all values. If we were to use the concept of a fake node from the Louvain algorithm, we would have to recalculate every Z-Score on each merge but given that the Z-Score is always calculated from the original graph, we know that all Z-Scores remain valid except from those communities adjacent to a community that has been merged.

Once we have selected two communities to merge, we follow the steps outlined below:

1. We arbitrarily name one community α and the other β;
2. Move all nodes from β into α, which is a constant time operating given a doubly linked list implementation;
3. Then delete all Z-Scores for β, such that $\forall c \in \mathrm{C.ZSs}_{\beta,c} = 0$;
4. Finally, update all Z-Scores for all communities that are connected to α, such that $\forall c \in \mathrm{C.ZSs}_{\alpha,c} = ZS(\alpha, c)$.

Each of these updates in step 4 requires $O(\log n)$ time to re-establish the heap invariants, so this operation takes $\mathrm{O}(n \log n)$ time. Again, given that we initially have $|V|$ communities this operation therefore takes $\mathrm{O}\left(|V| \log |V|\right)$ time. Given that we repeat this merging for every pair of communities, we have an upper bound on our computation of $O(|V|^3 \log |V|)$ for a connected graph, given that most graphs are not fully connected and we only have to compare each pair of communities joined by an edge we have $\mathrm{O}(|E|\,|V| \log |V|)$. This is much higher than the standard Louvain algorithm, which is $\mathrm{O}(|V| \log |V|)$.

On each iteration the number of communities decreases by one, therefore only $\frac{1}{2}|V|\left(|V|+1\right)$ communities have to be merged during the algorithm, which is also subject to the restriction that there must be an edge between communities, so our running time in the average case is not as high as our upper bound.

Algorithm 2 Pseudocode for the CJ0_ZS algorithm

$L \Leftarrow$ new mapping
for all $v \in V$ **do**
  $L_v \Leftarrow$ fresh n such that $\forall v' \in V\big(L_{v'} = n \Rightarrow v' = v\big)$
end for
for all $(\alpha, \beta) \in E$ **do**
  $ZSs_{index(\alpha,\beta)} \Leftarrow ZS(\alpha, \beta)$
end for
initalise heap ZSs # O($VlogV$)
for max $index(\alpha, \beta) = p$ from ZSs **do**
  if $p \leq 0$ **then**
    return L
  end if
  for all $n \in \beta$ **do**
    move vertex n to α
    $L_n \Leftarrow L_\alpha$ # mark all nodes as belonging to α comunity
  end for
  for all $(\gamma, \beta) \in E$ **do**
    $ZSs_{index(\gamma,\alpha)} = ZS(\gamma, \alpha)$
    remove $ZSs_{index(\gamma,\beta}$
  end for
  for all $(\alpha, \gamma) \in E$ **do**
    $ZSs_{index(\alpha,\gamma)} \Leftarrow ZS(\alpha, \gamma)$
  end for
end for

In Algorithm 2 we give our method for using the CJ0_ZS algorithm for topic modelling over a corpus.

Noticing that $\mathrm{ZS}(\alpha, \beta) = \mathrm{ZS}(\beta, \alpha)$ we utilise "index(a,b)" in this pseudocode to signify that $\mathrm{index}(a, b) = index(b, a)$. This means only half of the elements actually need to be stored in the heap.

Our $C(v, t)$ function gives us the probability that an entity or user v belongs to a topic t as outlined in Section 5.2.1.

The results of the CJ0_ZS algorithm can be given in the $C(v, t)$ form, by using the following definition:

$$C(v, t) = \begin{cases} 1 & L_v = t \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

Similarly to the Louvain algorithm, by definition, entities are hard assigned to communities.

5.2.4 Using LDA for Topic Modelling

We want to quantify the quality and stability of our graph based results against those from LDA, which is our baseline.

Online social network corpuses can be very small, especially when only scraping the content from one user and their friends. If we were to run LDA on Twitter data, then a document consists of no more than 140 characters.

For an entity n, and topic t, we define the probability n belongs in topic t, using LDA, as:

$$L(n, t) = [0, 1] \tag{46}$$

The results from LDA only cover entities, so in order to put this in the $C(v, t)$ form required, we first need to prepare a document that represents that user, that we can infer the topic of:

$$\Phi(u) = \text{concatenate}(\{n \mid n \in \mathcal{N}, \exists i \exists j (h_i = u \wedge \mathbf{n}_{ij} = \top)\}) \tag{47}$$

Where $\Phi(u)$ represents the concatenation of every entity uttered by a user u in the corpus.

$$C(v, t) = \begin{cases} L(v, t) & v \in N \\ \frac{\sum_{n \in \Phi(v)} L(n,t)}{\sum_{t' \in T} \sum_{n \in \Phi(v)} L(n,t')} & v \in U \end{cases} \tag{48}$$

5.2.5 Motivating Community Detection with Twitter Data

We can now define the topics $\mathcal{T}$ of our test data set of tweets using the CJ0_ZS algorithm outlined in the previous section, using a collection of tweets we collect in Section 5.3.10 as the corpus documents.

Figure 10: Tag cloud of a community from Twitter

We give a tag cloud of a community in Figure 10 listing the 100 most frequently found words in that community for a sample of our data, to help motivate what a community looks like.

Whilst the community is noisy, we can see several related concepts, for example:

- "bash", "buffer", "microsoft office", "sh", "skype", "software versioning", "version", "wordpress", "network", "konami code", "network"
- "cv", "curriculum", "graduation", "study", "college", "research"
- "doctor who", "dr", "doctor"

Some of this noise might be introduced by the ambiguity in our Wikipedia model being used for entity extraction in Section 5.1, it could also be caused by the community detection algorithm aggressively clustering these entities together, or perhaps that multiple interest groups overlap, for example those interested in Doctor who, or in studying might also be interested in software versioning.

Full community listings from this Twitter dataset are in Appendix A.

5.2.6 Evaluation

We want to quantify two things from the algorithms proposed: the accuracy of the clusters, and the stability of the true positive rate over multiple runs.

We used the Downton Abbey Season 1 screenplay to evaluate these algorithms. The benefit of using a screenplay is that we already have an independently established code for the *ingroups* of the characters, in the form of the groupings available on the Wikipedia page for the characters[6]. It is small enough to reproduce in Table 2 (where we have used the names of characters from the screenplay rather than from the Wikipedia page).

In this dataset we have 4136 individual utterances from cast members, consisting of 11.6 words on average (mean). These map to 5228 unique entities, with 2339 entities being spoken in more than one conversation.

Character	Code
Robert	Crawley Family
Violet	Crawley Family
Cora	Crawley Family
Mary	Crawley Family
Edith	Crawley Family
Sybil	Crawley Family
Matthew	Crawley Family
Isobel	Crawley Family
Rosamund	Crawley Family
Carson	Staff
Mrs Hughes	Staff
Bates	Staff
O'Brien	Staff
Anna	Staff
William	Staff
Thomas	Staff
Daisy	Staff
Mrs Patmore	Staff
Gwen	Staff
Branson	Staff
Molesley	Staff
Mrs Bird	Staff
Evelyn	Crawley Family Acquaintance
Crowborough	Crawley Family Acquaintance
Kemal	Crawley Family Acquaintance
Strallan	Crawley Family Acquaintance
Clarkson	Crawley Family Acquaintance

Table 2: Codes for Downton characters using screen name

The disadvantage of using a screenplay is that it is the sole work of Julian Fellows writing as

[6]http://en.wikipedia.org/wiki/List_of_Downton_Abbey_characters

multiple characters. Thus, it does not perfectly reflect the domain we are working in, which actually has content from multiple authors. We could run our suite of community detection algorithms over real data and ask users to code all their users into communities, but given that both Twitter and Facebook rate limit the downloading of data, it would require a substantial time commitment from these users.

Where possible we specify that each algorithm should find 3 communities. We can utilise $C_U(u)$ to get the classification of each character. Our topics $\mathcal{T}$ are unlabelled and thus $\mathcal{T} = \{1, 2, 3\}$. However, the classifications from Wikipedia are $\{\text{Staff}, \text{Crawley Family}, \text{Crawley Family Acquaintance}\}$. In order to evaluate the accuracy of each method, we need to define a mapping between the topics used by LDA, CJ0_ZS and Louvain and the topics used by Wikipedia editors.

We given the classifications from Table 2, as $\mathbf{g}$:

$$\mathbf{g} = \begin{pmatrix} \{\text{Robert}, \text{Violet}, \ldots, \text{Rosamund}\} \\ \{\text{Carson}, \text{ Mrs Hughes}, \ \ldots, \text{Mrs Bird}\} \\ \{\text{Evelyn}, \text{ Crowborough}, \ldots, \text{Clarkson}\} \end{pmatrix} \tag{49}$$

$\mathbf{g}$ represents the classifications from Wikipedia editors. We then have the classifications from a run of one of the algorithms: $C_U(1), C_U(2), C_U(3)$, which represent the 3 (unlabelled) classifications from our algorithms. We want to assign each set to one of our ground truths to maximise the true positive rate. We can enumerate all possible mappings:

$$\mathbf{M} = \begin{Bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 \\ \mathbf{g}_1 & \mathbf{g}_3 & \mathbf{g}_2 \\ \mathbf{g}_2 & \mathbf{g}_1 & \mathbf{g}_3 \\ \mathbf{g}_2 & \mathbf{g}_3 & \mathbf{g}_1 \\ \mathbf{g}_3 & \mathbf{g}_1 & \mathbf{g}_2 \\ \mathbf{g}_3 & \mathbf{g}_2 & \mathbf{g}_1 \end{Bmatrix} \tag{50}$$

We give the true positive rate for one run of LDA, Louvain or CJ0_ZS as:

$$TP = \max_{i \in \{1,2,3,4,5,6\}} |C_U(1) \cap \mathbf{M}_{i1}| + |C_U(2) \cap \mathbf{M}_{i2}| + |C_U(3) \cap \mathbf{M}_{i3}| \tag{51}$$

We randomise the input and run each algorithm 30 times on the same data but different orderings in order to gauge how stable the results are.

5.2.7 Motivating the Evaluation Methodology

We motivate how our evaluation methodology works by giving the concrete values from C_U for one run of LDA over our Downton dataset.

1. $C_U(1) =$ Crowborough (Acquaintance)

2. $C_U(2)$ =Gwen (Staff), Anna (Staff), Mrs Patmore (Staff), Daisy (Staff), Thomas (Staff), William (Staff), Mrs Hughes (Staff), Carson (Staff), O'Brien (Staff), Robert (Family), Sybil (Family), Bates (Staff), Molesley (Staff), Strallan (Acquaintance), Evelyn (Acquaintance), Kemal (Acquaintance), Branson (Staff), and Mrs Bird (Staff)

3. $C_U(3)$ = Edith (Family), Mary (Family), Cora (Family) Violet (Family), Isobel (Family), Matthew (Family), Clarkson (Acquaintance), and Rosamund (Family)

The majority of elements in topic 1 are acquaintances, in the second set the majority of characters are staff, and in the final classification the majority of characters are family. If we let $i = 6$ we have:

$$TP = |C_U(1) \cap \mathbf{g}_3| + |C_U(2) \cap \mathbf{g}_2| + |C_U(3) \cap \mathbf{g}_1| \tag{52}$$

This happens to be the mapping that gives our maximum true positive rate for this run:

$$TP = 1 + 13 + 7 = 21 \tag{53}$$

5.3 Audience Prediction

In the previous two sections we have outlined entity extraction and topic modelling, both of which we make use of when generating audience predictions. This section outlines our probabilistic model on which we make our predictions. To help guide users when making privacy decisions for their content, we show them the predictions calculated using the method below. This will enable users of our plugin to make quick and informed decisions as to which friends to share a status update with.

We model how a user decides the audience to share a document with. Utilising CJ0_JS to discover topics in a corpus of documents $\mathcal{D}$.

Recall from Section 5.2.1 the definition of a corpus of documents, we we call D, and contains some entities from the global vector of all entities from all documents, which we define as N.

In this usage of a document we are restricting the contents to the entities as discovered using *lookup* as described in Section 5.1 on the contents of tweets but our model is flexible enough to work for a vector of words or tokens.

Recall that we define a document as d_i as either containing an entity, or not:

$$\mathbf{n}_i : \{\top, \bot\}^{|\mathcal{N}|}$$

Where $\mathbf{n}_{ij} = \top$ represents the presence of entity j in the document d_i, and $\bot$ represent the lack of the entity in the document.

We also define the audience of a document as $\mathbf{m}_i$. Similarly it is a vector indicating the presence, or lack thereof, of each user in a global vector of users, which we call $\mathcal{U}$, for document d_i.

$$\mathbf{m}_i : \{\top, \bot\}^{|\mathcal{U}|}$$

Where $\mathbf{m}_{ij} = \top$, represents that j is part of the known audience of document d_i, and $\bot$ represent that j is not known as being an audience member to that document.

We call the publisher of a document h_i, where $h_i \in \mathcal{U}$

Thus our documents have the form:

$$d_i : h_i \times \mathbf{m}_i \times \mathbf{n}_i$$

When given a new document (which we call d_k), we want to be able to predict the audience vector $\mathbf{m}_k$. This is the basis for the audience we recommend the user selects when sharing a piece of content in real time. Therefore it is imperative to test the accuracy of these methods on real-world data. We do so in Section 7.3 utilising both Twitter and Facebook data.

We build a probabilistic model for audience prediction, where we work under the assumption that users will have a specific latent topic that they want to talk about, motivating both their choice about the audience they want to see that document, and also the entities they select to convey their chosen topic.

We have defined our domain in plate notation in Figure 11. In this figure we are simultaneously presenting the dependancies of our model, as well as the variables that are observed and those that are latent. We can see that our publisher h is fixed, but topic t, audience m and entities n are all dependant on the publisher. We can also see we have removed the cyclic dependency that each audience member m depends on other audience members, and each entity n depends on other entities in a document. These dependencies are captured within our topic distribution. Both audience and entities are dependant on our publisher and topic. We also introduce our uniform Beta priors as α and β, and our uniform Dirichlet prior as ω. These priors help to ensure a more realistic distribution of audience members, entities and topics in practice, we will revisit these in a later section.

5.3.1 Probability of an Audience Given a Document and Publisher

Recall that $\mathcal{N}$ is a vector of all entities that exist within our corpus, and $\mathcal{U}$ is the vector of all users within our model. We define a previously unseen document as d_k:

$$d_k : h \times \mathbf{m}_k \times \mathbf{n}_k \tag{54}$$

We are fixing h to be the same for all new unseen documents. At this point, after topic modelling, we only build an audience prediction model for the user who is using the plugin and thus h is the same for all new documents.

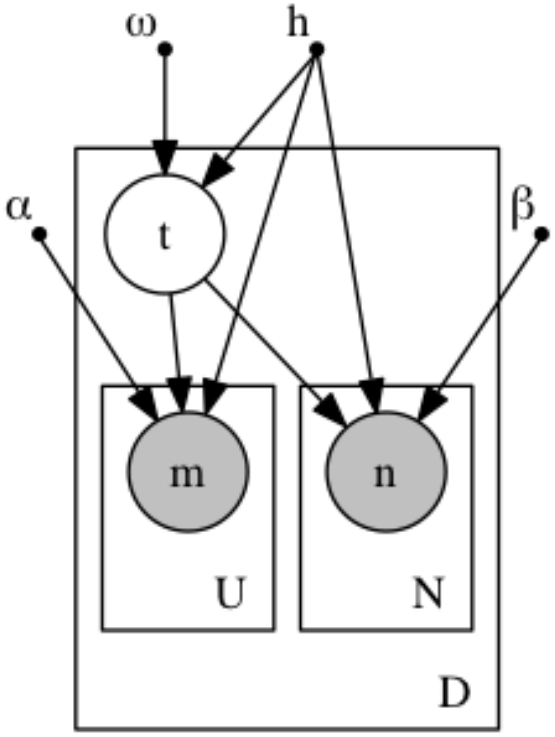

Figure 11: Plate notation for the topic of a document.

Recalling that $h \in \mathcal{U}$, $\mathbf{m}_{kj} = \top$ means j is in the audience of d_i, $\mathbf{n}_{kj} = \top$ means j is in the document d_k. This follows the same format as d_i but we note that $d_k \notin \mathcal{D}$, and instead $d_k \in \mathcal{D}_{\text{new}}$.

Note that the publisher of a document will not be included in the predictions of the audience, as the publisher will always be part of the intended audience.

We give the joint probability distribution as follows:

$$P(t_i, \mathbf{n}_i, \mathbf{m}_i \mid h, \alpha, \beta, \omega) = P(\mathbf{n}_i \mid t_i, h, \beta) P(\mathbf{m}_i \mid t_i, h, \alpha) P(t_i \mid h, \omega) \tag{55}$$

This captures our plate notation, that $\mathbf{n}_{ij}$ depends on h and t_i, and $\mathbf{m}_{ij}$ depends on h and t_i, and t_i depends on h. $\mathbf{n}_i$ does not depend on $\mathbf{m}_i$, and vice versa:

$$\begin{aligned} P(\mathbf{n}_i \mid t, \mathbf{m}_i, h, \beta, \alpha) &= P(\mathbf{n}_i \mid t, h, \beta) \\ P(\mathbf{m}_i \mid t, \mathbf{n}_i, h, \beta, \alpha) &= P(\mathbf{m}_i \mid t, h, \alpha) \end{aligned} \tag{56}$$

We say that an audience member's dependance on a latent topic captures the non-mutual-exclusivity between audience members, so that conditioned on the topic of the tweet, the probability of audience member are independent from one another:

$$P(\mathbf{m}_i \mid t, h, \alpha) = \prod_{u \in \mathcal{U}} P(\mathbf{m}_{iu} \mid t, h, \alpha) \tag{57}$$

Similarly, we say that an entities dependance on a latent topic will also capture the non-mutual-exclusivity of each entity; that conditioned on the topic of the tweet, the probability of an entity being in a tweet is independent from another:

$$P(\mathbf{n}_i \mid t, h, \beta) = \prod_{v \in \mathcal{N}} P(\mathbf{n}_{iv} \mid t, h, \beta) \tag{58}$$

In order to infer the probability of our document called d_k to be shared with a specific audience member $\mathbf{m}_{kj}$, we calculate the following probability, which is the probability of that audience member, given the entities, the publisher and our uniform priors α, β and ω:

$$P(\mathbf{m}_{kj} \mid \mathbf{n}_k, h, \alpha, \beta, \omega) \tag{59}$$

We give an application of Bayes' Rule applied to a conditional probability involving three variables as:

$$P\left(A \mid B, C\right) = \frac{P\left(B \mid A, C\right)\ \mathrm{P}\left(A \mid C\right)}{P\left(B \mid C\right)}$$

Therefore, having applied Bayes' rule, we can express our probability as:

$$P(\mathbf{m}_{ij} \mid \mathbf{n}_i, h, \alpha, \beta, \omega) = \frac{P(\mathbf{n}_i \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega) P(\mathbf{m}_{ij} \mid h, \alpha, \beta, \omega)}{P(\mathbf{n}_i \mid h, \alpha, \beta, \omega)} \tag{60}$$

This can be simplified using our results of Equations 56:

$$P(\mathbf{m}_{ij} \mid \mathbf{n}_i, h, \alpha, \beta, \omega) = \frac{P(\mathbf{n}_i \mid \mathbf{m}_{ij}, h, \beta) P(\mathbf{m}_{ij} \mid h, \alpha, \omega)}{P(\mathbf{n}_i \mid h, \beta, \omega)} \tag{61}$$

We can marginalise over topic in order to express the probability of entities given a publisher (which is given as $P(\mathbf{n}_i \mid h)$), where we simplify $P(t \mid h, \beta, \omega)$ to $P(t \mid h, \omega)$, we have:

$$P(\mathbf{n}_i \mid h, \beta, \omega) = \sum_{t \in \mathcal{T}} P(\mathbf{n}_i \mid t, h, \beta) P(t \mid h, \omega) \tag{62}$$

Similarly marginalising over topic in order to express the probability of an audience member given a publisher (which we define as $P(\mathbf{m}_{ij} \mid h)$), where we simplify $P(t \mid h, \alpha, \omega)$ to $P(t \mid h, \omega)$, we have:

$$P(\mathbf{m}_{ij} \mid h, \alpha, \omega) = \sum_{t \in \mathcal{T}} P(\mathbf{m}_{ij} \mid t, h, \alpha) P(t \mid h, \omega) \tag{63}$$

We can also marginalise over topic whilst calculating the probability of entities given the publisher, the audience members, and our uniform priors (we define this as $P(\mathbf{n}_i \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega)$), again simplifying where possible, to give:

$$P(\mathbf{n}_i \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega) = \sum_{t \in \mathcal{T}} P(\mathbf{n}_i \mid t, h, \beta) P(t \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega) \tag{64}$$

Using another application of Bayes' rule for our probability of a topic given the audience members, the publisher and our uniform priors (we define this as $P(t \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega)$), simplifying where possible, gives:

$$P(t \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega) = \frac{P(\mathbf{m}_{ij} \mid t, h, \alpha) P(t \mid h, \omega)}{P(\mathbf{m}_{ij} \mid h, \alpha, \omega)} \tag{65}$$

Finally marginalising $P(\mathbf{m}_{ij} \mid h, \alpha, \omega)$ over topic as we did in Equation 63 lets us express $P(t \mid \mathbf{m}_i, h, \alpha, \beta, \omega)$ as:

$$P(t \mid \mathbf{m}_{ij}, h, \alpha, \beta, \omega) = \frac{P(\mathbf{m}_i \mid t, h, \alpha)P(t \mid h, \omega)}{\sum_{t' \in \mathcal{T}} P(\mathbf{m}_{ij} \mid t', h, \alpha)P(t' \mid h, \omega)} \tag{66}$$

These equations allow us to express in terms of the components from our joint distribution, our inference rule for calculating the probability of an audience given a set of entities, the publisher and our priors, as so:

$$P(\mathbf{m}_{ij} \mid \mathbf{n}_i, h, \alpha, \beta, \omega) = \frac{\left[\sum_{t_1 \in \mathcal{T}} P(\mathbf{n}_i \mid t_1, h, \beta) \frac{P(\mathbf{m}_{ij} \mid h, t_1, \alpha)P(t_1 \mid h)}{\sum_{t_2 \in \mathcal{T}} P(\mathbf{m}_{ij} \mid t_2, h, \alpha)P(t_2 \mid h, \omega)}\right] \left[\sum_{t_3 \in \mathcal{T}} P(\mathbf{m}_{ij} \mid t_3, h, \alpha)P(t_3 \mid h, \omega)\right]}{\sum_{t_4 \in \mathcal{T}} P(\mathbf{n}_i \mid t_4, h, \beta)P(t_4 \mid h, \omega)} \tag{67}$$

Noticing that the inner summation over t_2 does not rely on the outer summation of t_1 we can pull this factor out of the summation, which gives as:

$$\begin{aligned} P(\mathbf{m}_{ij} \mid \mathbf{n}_i, h, \alpha, \beta, \omega) = & \\ & \left[\frac{1}{\sum_{t_2 \in \mathcal{T}} P(\mathbf{m}_{ij} \mid t_2, h, \alpha)P(t_2 \mid h, \omega)}\right] \\ & \times \left[\sum_{t_1 \in \mathcal{T}} P(\mathbf{n}_i \mid t_1, h, \beta)P(\mathbf{m}_{ij} \mid t_1, h, \alpha)P(t_1 \mid h, \omega)\right] \\ & \times \left[\sum_{t_3 \in \mathcal{T}} P(\mathbf{m}_{ij} \mid t_3, h, \alpha)P(t_3 \mid h, \omega)\right] \\ & \times \left[\frac{1}{\sum_{t_4 \in \mathcal{T}} P(\mathbf{n}_i \mid t_4, h, \beta)P(t_4 \mid h, \omega)}\right] \end{aligned} \tag{68}$$

This simplifies to:

$$P(\mathbf{m}_{ij} \mid \mathbf{n}_i, h, \alpha, \beta, \omega) = z^{-1} \sum_{t \in \mathcal{T}} P(\mathbf{n}_i \mid t, h, \beta)P(\mathbf{m}_{ij} \mid t, h, \alpha)P(t \mid h, \omega) \tag{69}$$

Where z is the same evidence when calculating the probabilities of different audience members for the same document:

$$z = \sum_{t \in \mathcal{T}} P(\mathbf{n}_i \mid t, h, \beta)P(t \mid h, \omega) \tag{70}$$

This is our equation for predicting the audience of a document given the entities, the publisher, and our uniform priors which we define in the next section.

We can calculate this equation by utilising the following three distributions from our joint probability distribution:

$$\begin{aligned} &P(\mathbf{m}_{ij} \mid t, h, \alpha) \\ &P(\mathbf{n}_{ij} \mid t, h, \beta) \\ &P(t \mid h, t_i) \end{aligned} \tag{71}$$

Where we utilise the result from Equation 58, in order to use the probability distribution of individual entities ($\mathbf{n}_{ij}$).

5.3.2 Uniform Priors

Because our dataset of documents from social networks is small, we do not want to express absolute certainty in our distributions over the entities and audience members in a document. For this reason we have introduced two uniform Beta priors: α and β, which we use to *soften* our certainty about the distribution of entities and audience members.

We define α over all $u \in \mathcal{U}$:

$$\alpha \sim Beta\left(\bar{\alpha} = 1, \bar{\beta} = 1\right) \tag{72}$$

We define β over all $n \in \mathcal{N}$:

$$\beta \sim Beta\left(\bar{\alpha} = 1, \bar{\beta} = 1\right) \tag{73}$$

Use of the beta distributions captures the idea that the occurrence of audience members and entities are not mutually exclusive, and helps to model our uncertainty in the test data given that it is not exhaustive.

Our Dirichlet prior (which we have called ω) is of the distribution:

$$\omega \sim Dir\left(\bar{\omega} = \begin{pmatrix} 1 \\ ... \\ 1 \end{pmatrix}\right) \tag{74}$$

Where $K = |T|$, and $|\bar{\omega}| = K$. Our Dirichlet helps to model that in a small corpus, such as those from online social networks it is possible that a user has never published a status update about a given topic, but that we cannot be certain that this mean that a user will never publish a status about that topic.

5.3.3 Maximum Likelihood of a Topic of a Document

We need to classify the topic of each document in our corpus, we can do this by assigning a single topic to each document that maximises the number of entities that are part of that document and in that topic:

$$t_{i,\mathrm{ML}} = g\left(\arg\max_{t \in \mathcal{T}} \sum_{v \in \mathcal{N}} \delta(\mathbf{n}_{iv}) C(v, t)\right) \tag{75}$$

Where we define g as returning a single element from the set with a uniform probability, and δ as converting our top and bottom values to integers such that we only sum the over entities that are present in the document.

$$\delta(t) = \begin{cases} 1 & t = \top \\ 0 & t = \bot \end{cases} \tag{76}$$

5.3.4 Probability of Audience Members given Topic

We need to calculate $P(\mathbf{m}_{ij} \mid t, h, \alpha)$ from our dataset. We can do this by counting the frequency of documents that have a specific audience member and have $t_{i,\mathrm{ML}} = t$. We combine these frequencies with our prior uniform Beta distribution.

Recall that we can calculate the probability of each audience member separately as a result of Equation 57:

$$P(\mathbf{m}_i \mid t, h, \alpha) = \prod_{j \in \mathcal{U}} P(\mathbf{m}_{ij} \mid t, h, \alpha)$$

We give the probability of an individual audience member being mentioned as the number of documents mentioning user divided by the number of all documents from the publisher h that mention the user, regardless of topic:

$$P(\mathbf{m}_{ik} = \top \mid t, h) = \frac{|\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{m}_{jk} = \top\}|}{|\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}|} \tag{77}$$

We combine this probability with our uniform beta prior α by saying:

$$\begin{aligned} P(\mathbf{m}_{ik} = \top \mid t, h, \alpha) \sim Beta(& \\ & \bar{\alpha}' = \bar{\alpha} + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{m}_{jk} = \top\}|, \\ & \bar{\beta}' = \bar{\beta} + |\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}| \\ ) & \end{aligned} \tag{78}$$

We give the expected probability as:

$$\begin{aligned} & P(\mathbf{m}_{ik} = \top \mid t, h, \alpha) \\ & \quad = \frac{\bar{\alpha} + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{m}_{jk} = \top\}|}{\bar{\alpha} + \bar{\beta} + |\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}|} \\ & \quad = \frac{1 + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{m}_{jk} = \top\}|}{2 + |\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}|} \end{aligned} \tag{79}$$

We give $P(\mathbf{m}_{ik} = \bot \mid t, h, \alpha)$ as equal to $1 - P(\mathbf{m}_{ik} = \top \mid t, h, \alpha)$.

5.3.5 Probability of Entities given Topic

With a very similar method as audience members, we need to calculate $P(\mathbf{n}_{ij} \mid t, h, \beta)$ from our dataset. We can do this by counting the frequency of documents that have a specific entity and have $t_{i,\mathrm{ML}} = t$. We combine these frequencies with our prior uniform Beta distribution.

Recall that we can calculate the probability of each entity separately as a result of Equation 58:

$$P(\mathbf{n}_i \mid t, h, \beta) = \prod_{j \in \mathcal{N}} P(\mathbf{n}_{ij} \mid t, h, \beta)$$

We give the probability of an individual entity being mentioned as the number of documents mentioning entity divided by the number of all documents from the publisher h that mention the entity, regardless of topic:

$$P(\mathbf{n}_{ik} = \top \mid t, h) = \frac{|\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{n}_{jk} = \top\}|}{|\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}|} \tag{80}$$

We combine this probability with our uniform beta prior β by giving:

$$\begin{aligned} P(\mathbf{n}_{ik} = \top \mid t, h, \beta) \sim Beta(& \\ & \bar{\alpha}' = \bar{\beta} + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{n}_{jk} = \top\}|, \\ & \bar{\beta}' = \bar{\beta} + |\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}| \\ ) & \end{aligned} \tag{81}$$

We give the expected probability as:

$$\begin{aligned} & P(\mathbf{n}_{ik} = \top \mid t, h, \beta) \\ & = \frac{\bar{\alpha} + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{n}_{jk} = \top\}|}{\bar{\alpha} + \bar{\beta} + |\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}|} \\ & = \frac{1 + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t, \mathbf{n}_{jk} = \top\}|}{2 + |\{d_g \mid d_g \in \mathcal{D}, t_{g,\mathrm{ML}} = t\}|} \end{aligned} \tag{82}$$

We give $P(\mathbf{n}_{ik} = \bot \mid t, h, \beta)$ as equal to $1 - P(\mathbf{n}_{ik} = \top \mid t, h, \beta)$.

5.3.6 Probability of a Topic

We define our $P(t \mid h)$ distribution for a publisher h_i as the frequency of documents in topic t, divided by the number of documents in all topics, thus:

$$P(t \mid h) = \frac{|\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t\}|}{|\{d_j \mid d_j \in \mathcal{D}\}|} \tag{83}$$

We take into account our uniform Dirichlet prior, which helps to *soften* our certainty about the probability distribution of $P(t \mid h)$. Thus:

$$P(t \mid h, \omega) \sim Dir\left(\bar{\omega}' = \begin{pmatrix} \bar{\omega}_1 + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = 1\}| \\ ... \\ \bar{\omega}_K + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = K\}| \end{pmatrix}\right) \tag{84}$$

Recalling that $K = |\mathcal{T}|$.

We can give the expected probability of this as:

$$\begin{aligned} & P(t \mid h, \omega) \\ & = \frac{\bar{\omega}'_t}{\sum_{t' \in \mathcal{T}} \bar{\omega}'_{t'}} \\ & = \frac{\bar{\omega}_t + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t\}|}{\bar{\omega}_1 + ... + \bar{\omega}_K + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = 1\}| + ... + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = K\}|} \\ & = \frac{1 + |\{d_j \mid d_j \in \mathcal{D}, t_{j,\mathrm{ML}} = t\}|}{K + |\{d_j \mid d_j \in \mathcal{D}\}|} \end{aligned} \tag{85}$$

5.3.7 Computational Complexity

From the equation given in Equation 69, we can see prediction does not depend on the length of a piece of content, but on the number of entities in our model, and the number of topics in our topic model. We need to perform this calculation $|\mathcal{U}|$ times to calculate the probability for each audience member. This gives us a naive computational complexity of $\mathrm{O}(|\mathcal{U}|\,|\mathcal{T}|\,|\mathcal{N}|)$.

We can improve on this. We know that social media documents rarely use every single entity in the model in a single status update, so we can calculate $P(\mathbf{n}_i \mid t, h, \beta)$ for a vector $\perp^{|\mathcal{N}|}$ in advance. When we want to generate probabilities for a given status update, we can calculate the change over this result instead of calculating the whole result from scratch. This gives us a complexity of $\mathrm{O}(|\mathcal{U}|\,|\mathcal{T}|\,n)$ where n is the number of entities present in a document.

5.3.8 Motivating Audience Predictions

We revisit the example of *Bob*, who we introduced in Section 1.1. We created a set of documents to motivate *Bob*'s situation, these documents are as follows:

$$\begin{aligned} d_1 &= (\text{Family Member A}, \{\text{Family Member B, Family Member C, Friend D}\}, \{\text{word 5}\}) \\ d_2 &= (\text{Family Member B}, \emptyset, \{\text{word 3}\}) \\ d_3 &= (\text{Family Member C}, \{\text{Bob}\}, \{\text{word 4}\}) \\ d_4 &= (\text{Bob}, \{\text{Friend B, Friend A}\}, \{\text{word 1, word 2}\}) \\ d_5 &= (\text{Friend D}, \emptyset, \{\text{word 2}\}) \\ d_6 &= (\text{Friend C}, \emptyset, \{\text{word 1}\}) \end{aligned} \tag{86}$$

We give a possible topic model for *Bob* in Section 1.1, for convenience given again in Figure 12.

If we recall that the structure of a document consists of a publisher h_i, a vector indicating the presence of all the audience members $\mathbf{m}_i$, and a vector indicating the presence of entities in the document. We have:

$$d_i : h_i \times \mathbf{m}_i \times \mathbf{n}_i$$

We break convention in order to keep the example readable, instead of presenting vectors $\mathbf{m}_i : \{\top, \perp\}^{|\mathcal{U}|}$ and $\mathbf{n}_i : \{\top, \perp\}^{|\mathcal{N}|}$ we use sets indicating the presence of an audience member or entity.

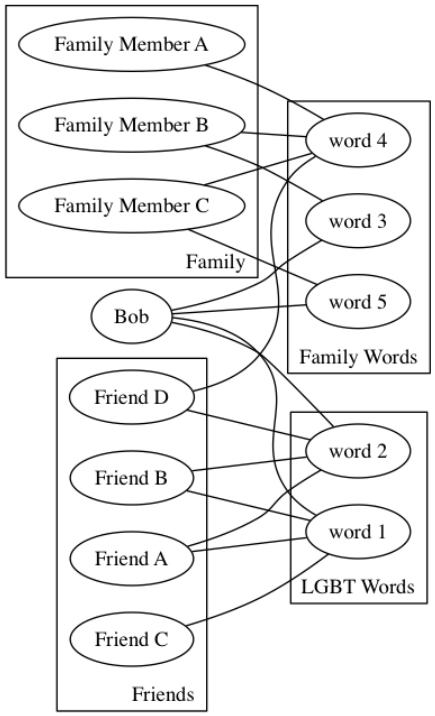

Figure 12: Possible community detection results for *Bob*'s *register* graph

We start with the methodology in Section 5.3.3. We label each document with a topic, which is trivial in this instance given there are so few entities in each document:

$$\begin{aligned} t_1 &= 1 \text{ (Family Words)} \\ t_2 &= 1 \text{ (Family Words)} \\ t_3 &= 1 \text{ (Family Words)} \\ t_4 &= 2 \text{ (LGBT Words)} \\ t_5 &= 2 \text{ (LGBT Words)} \\ t_6 &= 2 \text{ (LGBT Words)} \end{aligned} \tag{87}$$

Giving us $\mathcal{T} = \{1, 2\}$.

From here we can define the probabilities of different words being in different topics using the methodology in Section 5.3.5, we give these in Table 3.

From here we can define the probabilities of different audience members being in different topics using the methodology in Section 5.3.4, we give these in Table 4.

Finally we write the probability of Bob writing a document about different topics using the methodology in Section 5.3.6:

$$\begin{aligned} P(t = 1 \mid h = \text{Bob}, \omega) &= 0.33 \\ P(t = 2 \mid h = \text{Bob}, \omega) &= 0.67 \end{aligned} \tag{88}$$

n	$n = \top, t = 1$	$n = \bot, t = 1$	$n = \top, t = 2$	$n = \bot, t = 2$
word 1	0.5	0.5	0.75	0.25
word 2	0.5	0.5	0.75	0.25
word 3	0.67	0.33	0.5	0.5
word 4	0.67	0.33	0.5	0.5
word 5	0.67	0.33	0.5	0.5

Table 3: The probabilities for $P(\mathbf{n}_k \mid t, h = \text{Bob}, \beta)$.

u	$u = \top, t = 1$	$u = \bot, t = 1$	$u = \top, t = 2$	$u = \bot, t = 2$
Friend A	0.5	0.5	0.67	0.33
Friend B	0.5	0.5	0.67	0.33
Friend C	0.5	0.5	0.5	0.5
Friend D	0.5	0.5	0.5	0.5
Family Member A	0.5	0.5	0.5	0.5
Family Member B	0.5	0.5	0.5	0.5
Family Member B	0.5	0.5	0.5	0.5

Table 4: The probabilities for $P(\mathbf{m}_k \mid t, h = \text{Bob}, \alpha)$.

We want to predict an audience for this new document:

$$h = \text{Bob and } \mathbf{n}_k = \{\text{word 1, word 2, word 4}\} \tag{89}$$

Firstly we calculate z to simplify our later calculations:

$$z = (0.33 \times 0.5 \times 0.5 \times 0.33 \times 0.67 \times 0.33) + (0.67 \times 0.75 \times 0.75 \times 0.5 \times 0.5 \times 0.5) = 0.0531 \tag{90}$$

For Friend A, we calculate $\mathbf{m}_{k\text{Friend A}} = \top$, following the methodology in Section 5.3.1:

$$\begin{aligned} &P(\mathbf{m}_{k\text{Friend A}} = \top \mid \mathbf{n}_k, h, \alpha, \beta, \omega) \\ &\quad = z^{-1}((0.33 \times 0.5 \times 0.5 \times 0.33 \times 0.67 \times 0.33 \times 0.5) \\ &\qquad + (0.67 \times 0.75 \times 0.75 \times 0.5 \times 0.5 \times 0.5 \times 0.67)) \\ &\quad = \frac{0.0346}{0.0531} \\ &\quad = 0.651 \end{aligned} \tag{91}$$

We see that it is more than likely that Friend A should be audience to this status update. We

continue and discover the same result for Friend B. We give the derivation for Friend C next:

$$\begin{aligned} & P(\mathbf{m}_{k\text{Friend C}} = \top \mid \mathbf{n}_k, h, \alpha, \beta, \omega) \\ & \quad = z^{-1}((0.33 \times 0.5 \times 0.5 \times 0.33 \times 0.67 \times 0.33 \times 0.5) \\ & \qquad + (0.67 \times 0.75 \times 0.75 \times 0.5 \times 0.5 \times 0.5 \times 0.5)) \\ & \quad = \frac{0.0266}{0.0531} \\ & \quad = 0.5 \end{aligned} \tag{92}$$

This communicates that Friend C may or may not be in the audience of this status update. We err on the side of caution and by default leave Friend C out of the audience unless a user explicitly directs us to include them.

This happens for the rest of the audience members too because we are working with such a small sample corpus. This gives our most likely audience as $\mathbf{m}_k = (\top, \top, \bot, \bot, \bot, \bot, \bot)^{\mathrm{T}}$.

This follows our intuition that most words in $\mathbf{n}_i$ are part of topic 2, and *Bob* has only messaged two people, Friend A and Friend B about topic 2, so these two audience members should be the only two users with a high probability of being the audience of d_i.

5.3.9 Methodology for Evaluation

It is imperative to test our solution on real-world test data, because these audience predictions are the predictions shown directly to users, to improve their privacy policy making decisions. We outline a methodology utilising cross-validation in Section 5.3.9. In the following two sections we outline how we obtained our real world test data from Twitter and Facebook.

We give ROC curves for our implementation which showcase the tradeoffs between sensitivity and specificity.

We define a function audience' that given a parameter ϑ, returns a predicted audience, when given the entities from a document, and the publisher:

$$\text{audience}'\,(i, \vartheta) = \{u \mid u \in \mathcal{U}, P(\mathbf{m}_{iu} \mid \mathbf{n}_k, h, \alpha, \beta, \omega) \geq \vartheta\} \tag{93}$$

We simply only list the audience members who have a probability greater than or equal to the parameter ϑ.

In order to use cross-validation, documents in which one or more users who were mentioned. We split these documents up into 20 roughly equally sized sets of test data. We built a model for each batch of test documents by calculating the probability of an audience using a model that does not contain the batch from which that test document came from, such that:

$$\begin{aligned} & \text{prediction}\,(i, \vartheta) = \\ & \quad \text{audience}'_{\mathcal{D}/testbatch}\,(i, \vartheta) \;\text{ where } \exists testbatch\,(i \in testbatch) \end{aligned} \tag{94}$$

We calculate the true positive, false positive, true negative and false negative for a single document using:

$$TP(i,\vartheta) = |prediction\,(i,\vartheta) \cap m| \tag{95}$$

$$FN(i,\vartheta) = |m/prediction\,(i,\vartheta)\,| \tag{96}$$

$$FP(i,\vartheta) = |prediction\,(i,\vartheta)\,/m| \tag{97}$$

$$\mathrm{TN}\,(i,\vartheta) = |friends(\text{"H"})| - (TP\,(i,\vartheta) + FN\,(i,\vartheta) + FP\,(i,\vartheta)) \tag{98}$$

We calculate the sensitivity and specificity:

$$\text{sensitivity}_{\vartheta} = \frac{\sum_i TP(i,\vartheta)}{\sum_i TP(i,\vartheta) + FN(i,\vartheta)} \tag{99}$$

$$\text{specificity}_{\vartheta} = \frac{\sum_i TN(t,\vartheta)}{\sum_i TN(i,\vartheta) + FP(i,\vartheta)} \tag{100}$$

Most of our results are presented a ROC curve of $\text{sensitivity}_{\vartheta}$ against $1 - \text{specificity}_{\vartheta}$ for ϑ between [0,1], we have used adaptive sampling (Section 5.3.12) to ensure an even distribution of points across our graph.

A naive implementation where we randomly choose whether an audience member is included or not would yield a ROC curve of:

$$sensitivity = 1 - specificity \tag{101}$$

We can measure success based on whether our ROC curve is:

$$sensitivity > 1 - specificity \tag{102}$$

We let $\mathrm{TPR}\,(\vartheta) = sensitivity_{\vartheta}$ and $\mathrm{FPR}\,(\vartheta) = 1 - specificity_{\vartheta}$.

We sometimes give the AUC or "Area Under Curve". We can calculate this by calculating the summation of the area of trapeziums formed between two points from our sampling, where s is the sampling rate:

$$\begin{aligned} AUC = &\sum_{k=1}^{s^{-1}} \frac{1}{2} \left(\mathrm{TPR}\,(\mathrm{sk}) + TPR\,(sk-s)\right)\ \left(\mathrm{FPR}\,(sk) - FPR\,(sk-s)\right) \\ &\pm \sum_{k=1}^{s^{-1}} \frac{1}{2} \left(\mathrm{TPR}\,(sk-s) - TPR\,(sk)\right)\ \left(\mathrm{FPR}\,(\mathrm{sk}-s) - FPR\,(sk)\right) \end{aligned} \tag{103}$$

Note that due to the sampling, the area under the curve does not necessarily accurately represent the performance of algorithms, so we provide the error for each case (the second part of the formula).

5.3.10 Obtaining Twitter Data

We can model a tweet as a 3-tuple consisting of:

$$(\text{ publisher, users mentioned, } \textit{lookup}(\text{word}_1 \text{ ... word}_n) \text{ }) \tag{104}$$

Utilising *lookup* to extract entities as defined in Section 5.1.

We sometimes refer to a set of these as "tweets", although these form the documents in $\mathcal{D}$:

$$\text{tweets}_i \ : \{(h_i, \ \mathbf{m}_i, \ \mathbf{n}_i)\} \tag{105}$$

The publisher is the user who posted the tweet, the users mentioned specified using @-tags, filtered to only those who are in $\mathcal{U}$, and entities are the entities as discovered using *lookup* in the text of the tweet.

We take a list of a user's friends as those users who both follow, and are followed back by a given user:

$$\text{friends}\,(\text{user}) = \{\text{user}_1, \ldots, \ \text{user}_F\} \tag{106}$$

Given a set of tweets "tweets" we have downloaded from both the user we are building the privacy policy for "user" and that user's friends "friends(user)"; we can build our bipartite *register* graph.

On Facebook and in screenplays a conversation is explicitly initiated between participants and remains between those participants without explicit action to increase the audience through inviting new participants or explicit stage directions for a new cast member to join a conversation. On Twitter, most conversations happen in public and are accessible for anyone to view, and for anyone to join. For this reason it is crucial we create the notion of an "intended audience" which is a subset of users who we explicitly know were supposed to read the tweet by being mentioned. We evaluate our results against this set of users who were mentioned.

Note that we could include users who retweet or favourite a tweet as part of our audience, and whilst we know that they definitely saw the tweet in order to perform these operations, we do not know that they are part of the audience the user intended to see the tweet.

We have access to these tweets through the Twitter API [66]. Our first implementation utilised the "/lists/*" endpoint. This allows us to build a list of users whose tweets we wish to download, we can then repeatedly query the "/lists/statuses" to download all tweets from all users in our tweets. However, we found that we get faster performance querying "/statuses/user_timeline". This could be because multiplexing several user's twitter timelines requires more computation within Twitter's infrastructure than querying a single user.

We downloaded 144,000 of the historical tweets from the timelines of one user "H" and the 124 friends("H"). There were two protected accounts for which we were unable to download any tweets.

We identified 1,690 tweets that met our testing criterion of mentioning at least one friend, from there, we randomly separated our test data out into 20 batches of 84-85 test tweets each.

We are limited to downloading a maximum of 36,000 tweets in any 15 minute window. This represents 19,403,375 bytes of tweet text, or roughly 2,994,271 words.

We set up an application on Twitter infrastructure and utilised the app-level access token to download the Tweets, thus requiring no user confirmation or authorisation to download their unprotected tweets.

5.3.11 Obtaining Facebook Data

The Facebook API allows us to download private conversations that occur on Facebook messenger, once this permission has been confirmed by an end user. For this reason this dataset only contains data from the author's own private Facebook conversation history.

Facebook lists each conversation as a single entity, with each set of participants listed separately. Where a user closes Facebook, or the conversation, and then resumes the conversation, or sends a message to the same set of participants, it will add these messages to the previously created entity.

We download the index of conversations and each conversation in turn, which due to rate limiting can often take upwards of an hour. Unlike Twitter there are no published guidelines on how the rate is calculated or can be monitored with the rate limits appearing to be based on resource contention instead of a fixed number of requests.

We use a streaming JSON parser to quickly read the format and process nodes, which is more scalable than having to load the whole JSON structure into memory.

This dataset contains 185,469 individual messages over an 8 year time frame, with a total of 5,882,902 bytes of conversation text, or roughly 1,172,000 words. This represents less raw data than Twitter. However, because all messages are either from or to a single user (the owner of the account) this actually represents more test data than the Twitter dataset.

5.3.12 Adaptive Sampling

We initially tried a fixed step increase for ϑ but this leaves us with lower resolution graphs as the difference between probabilities of individual audience members decreases. This is especially apparent with the Facebook dataset results, due to there being many conversations where only a few messages were ever exchanged.

We build our graphs by adapting the resolution of our parameter ϑ. Given that we have access to the underlying probabilities of each audience member, we can calculate the value of ϑ where our audience predictions will change:

$$\delta = \{P(\mathbf{m}_{iu} \mid \mathbf{n}_k, h, \alpha, \beta, \omega) \mid u \in \mathcal{U}, i \in \mathcal{D}\} \tag{107}$$

We now have a set of all audience probabilities in this initial test batch. We cannot utilise this set as is, given that for the Facebook data set this would require too much memory to store a point for

each change in sensitivity or specificity. Instead we plot the initial test batch at this exact sample resolution, and select 1,000 evenly spaced points along sensitivity = 1-specificity. We reduce our δ set to these 1,000 points, (including $0 \in \delta$, $1 \in \delta$) and use these for calculating the sensitivity and specificity for the remaining 19 test batches.

Because our test batches are generated randomly, the first test batch is representative of the rest of the test batches and we maintain a high precision sampling rate without affecting the performance of our evaluation program.

5.3.13 Null Hypothesis

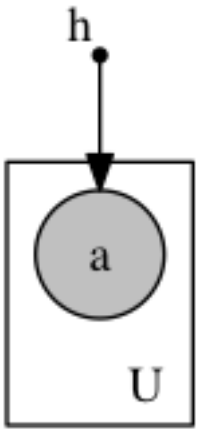

Figure 13: Plate notation for making predictions without topic modelling

For our null hypothesis, we remove the topic information to get a baseline of how much the probability of $P(a \mid h)$ works as a classifier. That is to say, how well a model consisting of only information about the frequencies with which h mentions different audience members performs. We present this model in plate notation in Figure 13.

6 Implementation

In this section we outline how we have utilised the methods outlined in Section 5. Our implementation needs to be quick so as to work effectively on the commodity hardware our plugin will run on. Firstly we explain the overall architecture of Gatekeeper, before presenting our implementation for entity extraction as outlined in Section 5.1, topic modelling as outlined in Section 5.2, and audience predictions as outlined in Section 5.3.

In order to achieve our goal of inline audience predictions to help empower end users to build privacy policies on a per status update granularity, our solution requires three components. Our Gatekeeper server stores the dictionaries we use for entity extraction, and facilitates the Gatekeeper application's initial authorisation with Facebook. Our Gatekeeper application runs on the end-user's computer, building topic models and providing audience predictions to plugins. Our Chrome plugin runs within the Chrome browser and connects to the Gatekeeper application.

This separation of concerns enables plugins to be built for many browsers whilst reusing the same codebase within the Gatekeeper application. Figures 14 and 15 show how our plugin provides inline audience predictions on the Facebook website.

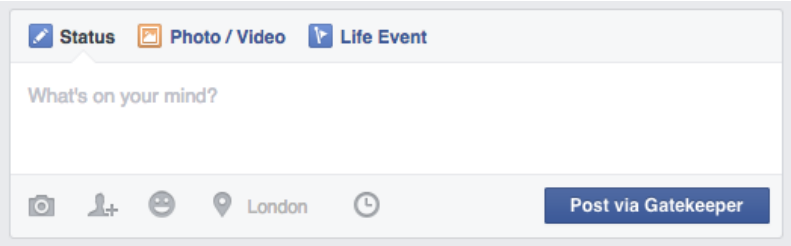

Figure 14: Gatekeeper integrated with the user profile status update component

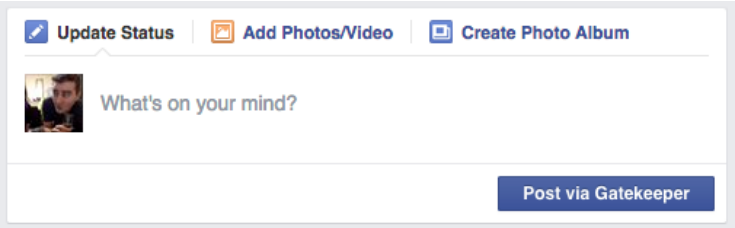

Figure 15: Gatekeeper integrated with the timeline status update component

We have removed the original privacy button from the status update component, as Gatekeeper supersedes this functionality.

Upon writing a status and clicking "Post via Gatekeeper" the user is shown a list of their friends categorised into those friends Gatekeeper is certain a user would want to share that status with, and then varying degrees of uncertainty. Figure 16 shows how a user is able to select which friends they want to share that status amongst by using the checkboxes. We can see that had the user entered

a different status update, then a different audience prediction is calculated as we demonstrate in Figure 17.

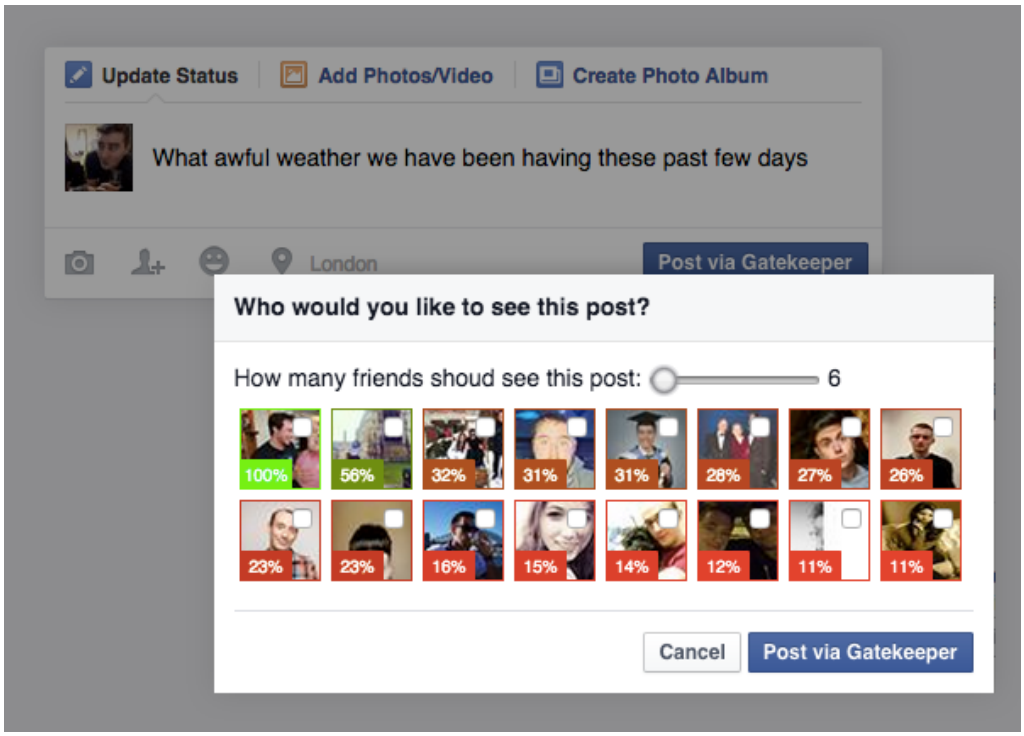

Figure 16: An example audience prediction

When a user confirms, the status update is posted as normal to a user's timeline. We show an example status update both from the point of view of the end user, and the view that their friends will see in Figures 18 and 19.

Note that a user can still alter the privacy settings of a piece of content after it has been published utilising functionality Facebook provides. However, the audience of a status update are not able to view or alter the privacy policy of a piece of content.

There are several goals we achieve with our implementation:

1. Privacy: the plugin must be secure and not expose the personal data of users to anyone but the user;
2. Speed: the plugin must be able to predict audiences quickly, and display them to an end user before they post a status;
3. Accessibility: the plugin must be able to show audience predictions to end users in an appropriately accessible way.

We go through each of these challenges in turn in the following sections, before covering implementation details of the methods listed in our methodology in Section5.

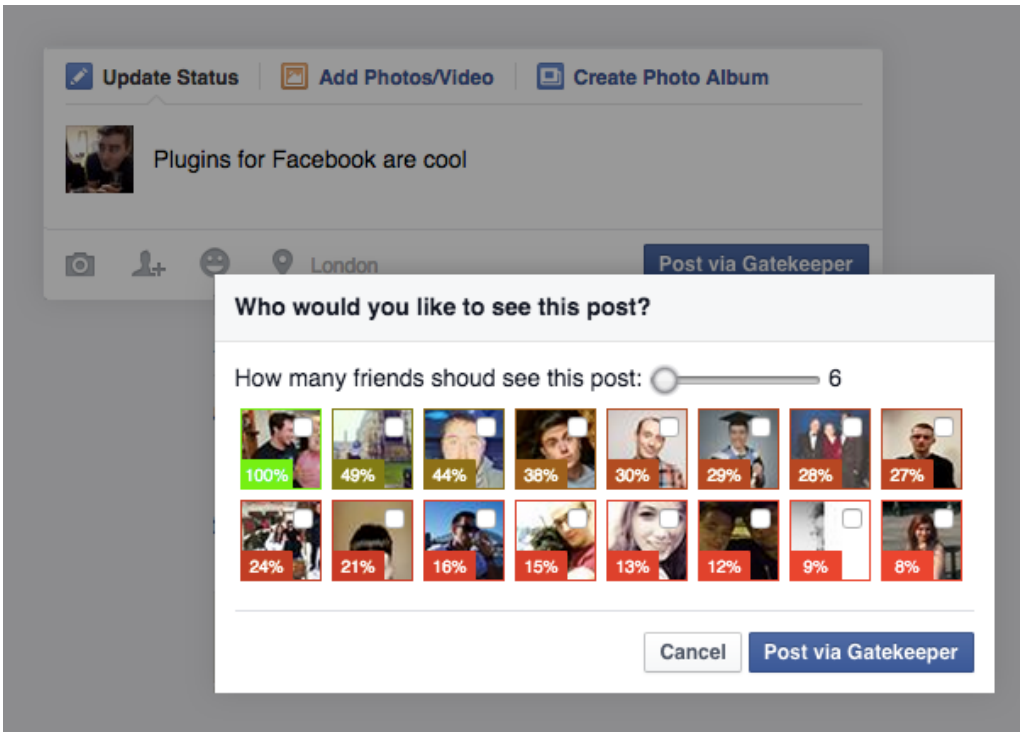

Figure 17: Another example audience prediction

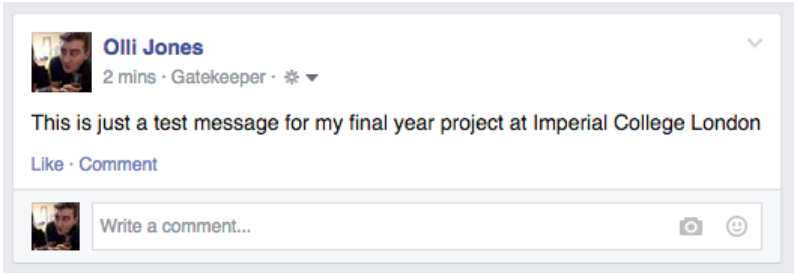

Figure 18: A status posted through Gatekeeper as visible to the end user

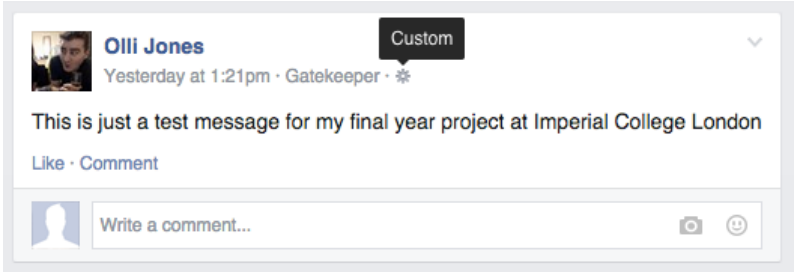

Figure 19: A status posted through Gatekeeper as visible to the users allowed to see it

6.1 Privacy

We want to implement a plugin requiring as little trust of the underlying ecosystem as possible. All communication between the Facebook API and the Gatekeeper server are over HTTPS and the certificates are verified upon connection. We trust the Facebook API given that the users are entrusting Facebook with their status updates in the first place. We do not trust the Gatekeeper server. We trust a user's computer.

However, an end user's computer may not be secure and for that reason we minimise the attack surface of our application by not storing any conversational data on an end user's disk. Once the conversational data has been downloaded and processed into a topic model the plugin instructs the Facebook API to remove its access to conversational user data.

This means for the entirety of downloading the conversational data the user's computer will have to remain on. This requirement is clearly documented during topic modelling.

Given that the Facebook API is rate limited, we want to store the topic models on disk and not regenerate topic models each time the application launches. To make it difficult to extract these topic models we generate a cryptographically secure unique salt and hash each entity to ensure that an attacker can only query the existence of words in the model, and to minimise the effectiveness of rainbow tables over the topic models of multiple targets.

The plugin must store the user IDs of friends in the model in order to communicate to Facebook, which friends should have access to what content. However, Facebook creates per-application user IDs within its API [67] so these values only have meaning to Facebook and the Gatekeeper application.

In order to provide audience predictions the Gatekeeper application requires a copy of the status update in question. This is never transmitted outside of the end user's computer, and if an end users computer is compromised there are easier ways of extracting this information directly such as through key loggers than through attacking Gatekeeper.

Node.js has its own key store [68], and thus can prevent man-in-the-middle attacks on the Gatekeeper applications communications with the Facebook graph API even when an end user's computer has been compromised and additional malicious certificate authorities have been added. However, this does mean that we must trust Node.js's key store, but given that Node.js is open source we can monitor the key store's code listing.

6.2 Plugin Speed

Speed is crucial given that a privacy policy must be defined before a status update is posted. In benchmarks on a Intel core I5 MacBook Air, Gatekeeper was consistently able to predict the audience of content within two seconds. This is representative of our $O(|U|\,|T|\,|N|)$ computational complexity for predicting audiences as covered in Section 5.3.7.

6.3 Making Audience Predictions Accessible

Engineering our plugin in order for it to display audience predictions *as part* of the end user's consumption of the Facebook website was crucial for users being able to utilise these audience predictions as a method for enabling quick privacy policy decisions to be made. If generating audience predictions required a complicated and laborious set of steps then it would not be an improvement over the current experience. However, achieving this was complicated. We outline how our plugin achieves this integration, by motivating the different components of the plugin architecture as in Figure 20.

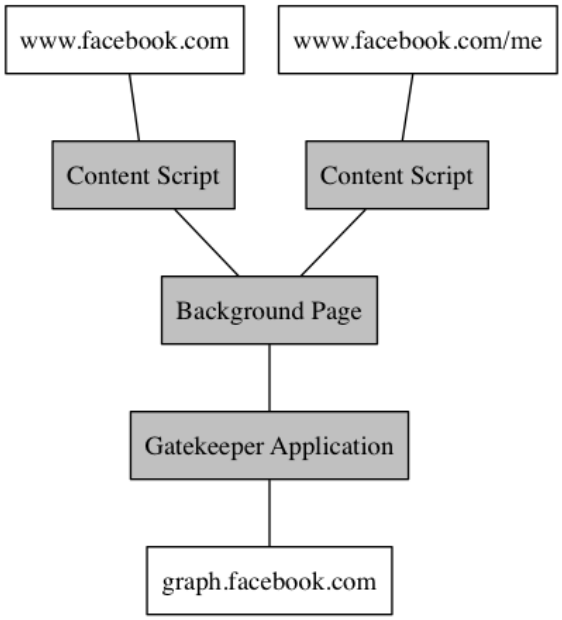

Figure 20: Plugin architecture with components we control in grey

Upon loading "www.facebook.com" or "www.facebook.com/me", Chrome will start a separate JavaScript stack for our extension's content script. This stack has its own environment, and as such its own instantiations of the JavaScript standard library but it does share the Document Object Model (or DOM) with the Facebook page. This allows us to capture the page loaded event and to swap the standard "Post" button with our "Post via Gatekeeper" button.

There are two problems with this process:

1. Firstly Facebook utilises *HTML5* features and loads content dynamically without causing a page reload, and importantly, without triggering the page loaded event that we capture.
2. Secondly the majority of the Facebook markup is generated on-the-fly using React [69], and utilises randomly assigned UUIDs for components, this makes it difficult to discover the "Post" button we want to swap.

We overcome the dynamic load by replacing the default behaviour of "history.pushState", which

is called on dynamic page load to change the URL displayed in the address bar. Whilst this functionality only works on some browsers, we are only concerned with Chrome, given that our plugin is designed specifically for this browser.

Because content scripts run in their own environment and only share the DOM, and not the "history" object, we have to inject this functionality into the Facebook DOM in the form of an inline script, with a channel open to communicate with the content script in order to communicate with the content script when dynamic page loads occur.

We have to use heuristics to locate the status update component on the Facebook page because we cannot locate it by ID, given that these are all dynamically generated. We look for a *textarea* element near the top of the page, which has the caption "What's on your mind?". We then scan outwards[7] from that component until we find the "Post" button. Sometimes the "Post" button is not added to the DOM until after a user clicks on the text area, so we attach a listener to the "focus" event of the text area to check again for the post button if it was not found at first.

Once this button is clicked the content script is able to communicate with the background page, which manages the communication channel to the Gatekeeper Application.

The Gatekeeper Application has access to the file system and can load our topic models (Section 6.6), run entity extraction over the status update (Section 6.5) and then make audience predictions (Section 6.7). These predictions are sent back to the background page, which returns the predictions to the content script that is facilitating the posting of the status update.

The content script is then able to inject UI into the Facebook page in order to display the audience prediction to the end-user. The end-user interacts with this component as though it is a part of the Facebook UI. We built the components utilising Facebook's own open sourced React library [69], with some modifications to allow two instances of the framework to coexist in one DOM.

Utilising grunt [70], browserify [71] and babel [72], when a source file is changed the build system automatically detects this change and transpiles our React components, which consist of HTML inlined in JavaScript, into pure JavaScript for the ECMAScript 5 implementation Chrome runs. Being able to write HTML components directly instead of hand writing JavaScript code to build these DOM elements was very beneficial to the speed at which we could prototype the plugin, and improves the maintainability of the plugin. Because React supports many modern browsers, we have the option to reuse these components in multiple browser implementations; implementing only the browser specific code required to facilitate communication between the plugin and the Gatekeeper application.

Upon clicking "Post" the requested audience is sent back to the Gatekeeper Application via the background script, which posts the status via the Facebook Graph API on behalf of the user.

If the Gatekeeper Application's Facebook access token expires, then it can make a request that the background page open a new tab, in order to allow the end-user to re-authenticate. Often this re-authentication happens seamlessly without user interaction.

[7] Outwards meaning to scan each parent element's content for the "Post" button, this is affectively searching *near* the textarea, with successive loss of precision.

6.4 The Role of the Gatekeeper Server

Because private Facebook conversation history is by its very nature, private, it is important to ensure that the Gatekeeper client can run on a user's laptop requiring as little server side interaction as possible. However, our plugin does not have a graphical user interface of its own, so it needs a server to help facilitate user authentication over HTTPS with Facebook.

We have an application registered on Facebook. They provide us with an application secret, which we embed in the client plugin. This application secret when sent to Facebook confirms that it is our application making the request.

The client application will generate its own secret (called *state* in the Facebook API terminology) and will open a browser session directly for the user to either allow or disallow the application to have access to their profile. This will return a *code* that in combination with the *state* can be exchanged for an access token. Instead of returning the access token directly to the Gatekeeper client, we send it via HTTPS to the Gatekeeper server, the Gatekeeper server is then able to verify it is authenticating the right user from the right computer.

Because only the Gatekeeper client has a copy of the *state*, only the client can redeem an access token using the code, but in order for the code to be sent to the client the user must first confirm to the Gatekeeper server that they want to permit access to their account at this time, the code is then sent back to the client who can redeem an access token to access the user's profile. No further interaction with the Gatekeeper server is required. Once the client has processed a copy of the private Facebook conversation history, it notifies Facebook that it would like its own access to this data revoked.

This communication flow is outlined in Figure 21.

This process does not require the plugin to trust the Gatekeeper server because only the client has the *state* and the *code* required to redeem an access token. The client will revoke the access level granted as soon as processing is complete thus further reducing the attack surface a malicious user has to access private data. Once the client (and only the client) has an access code, the Gatekeeper application can communicate directly with the Facebook API endpoints, and the Gatekeeper server no longer plays a role within the application apart from to provide the dictionaries for entity extraction, as covered in the next section.

The owner of the Gatekeeper server does have control over whether to forward a *code* to an instance of the Gatekeeper plugin, whilst this could be used for malicious purposes such as to stop users from using audience predictions, if a security vulnerability was discovered that allowed the plugin to be subverted, the Gatekeeper server can act as a stopgap measure preventing an attack from gaining traction by preventing the Gatekeeper applications from authenticating with Facebook; prompting users to download an updated version of the Gatekeeper plugin.

6.5 Entity Extraction

In Section 5.1 we gave the methodology for entity extraction.

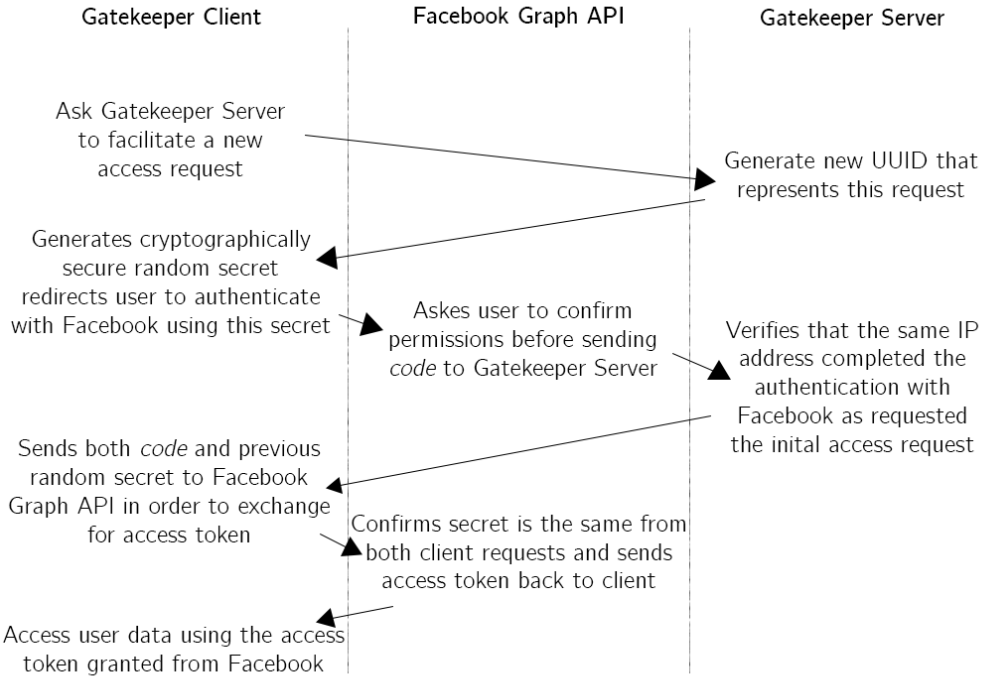

Figure 21: Establishing communication between the client and Facebook

Entity extraction is needed in our plugin in order to optimise the topic modelling and audience prediction engine to run on the commodity hardware of end users. It does this by reducing the amount of information needing to be processed from a corpus by filtering the data in a document to include only the words that are most likely to define the topic of that content.

It is our hypothesis that these topics are shared exclusively within subsets of a user's friends and so modelling them will enable us to make high quality predictions about the audiences of documents. By providing high quality audience predictions to the users of our plugin, we will enable them to enforce privacy policies on a per status update granularity without the burden of manually configuring access rights for each of their friends.

We processed the English language Wikipedia data dump from the 04-Mar-2015 using our Gatekeeper server virtual machine. Speed and memory performance were not necessarily of concern for this subtask. However, we did not want to have to transmit private Facebook conversations from an end-user's computer in order to determine an audience, preferring to keep all processing in the user's control. Whilst this method of entity extraction is not novel, there is complexity in that we need to extract entities quickly (in some situations in near realtime) from a corpus of documents of informal language from online social networks, on commodity end-user machines, both in order to build our topic models, and to predict an audience for a given status update.

Initially given that most plugins for Chrome are written in JavaScript [64], we implemented entity extraction in JavaScript utilising a Promise design pattern to abstract concurrency race conditions.

However, when trying to process a large dataset such as a corpus from Facebook, we ran into memory limits due to creating too many closures.

When thinking about alternative language choices, we considered compilable languages such as C, C++ and the Go programming language[8]. Whilst all three would be fast enough choices, Go's built in *bzip2* library, along with syntax for quickly writing threaded tasks made Go the clear choice for this task.

As mentioned above, we utilise Go's *bzip2* library. This enables us to build our dictionary from the compressed *bzip2* data dump directly from Wikipedia, uncompressing the blocks in memory as we need them, and then freeing the blocks from memory once processed. We were able to use Go's streaming XML parsing library to extract each *page* structure from the data dump, without having to load all 40gb of XML structure into memory at once.

We used Natural's aggressive tokeniser [45], which splits a string of input at every group of non-whitespace characters. We rewrote this functionality in Go, as Natural was written in JavaScript.

From here we were able to normalise every page title and every redirect. We created a mapping between these normalised forms and the exact Wikipedia page title of the original entity. We saved this to disk in a simplistic format:

normalised(*entity*) "|" *entity* "\n"

We followed the same methodology as Hu et al. [28] and removed all Wikipedia pages outside namespace 0 (all pages that are not articles), and all pages where the title is a single stopword.

Disambiguation pages were only included by Hu et al. to resolve polysemy, and because we are only concerned with entity extraction, not entity disambiguation, we kept a single entry for each ambiguous page and removed disambiguating pages from our dataset.

If we were to disambiguate using this methodology, we would need to store more information from Wikipedia, which is not feasible given we are already using a large amount of storage (for a web plugin) for the data we are already extracting and using.

We removed 5,197,443 removed pages in total in our dataset, with 9,059,127 entities remaining.

Once our dictionary has been sorted we can use a binary search for looking up entities within the dictionary. As illustrated previously in Figure 6, our normalising is a surjective function, and once our output is sorted we are able to take the set union of entities with the same normalised text and store this as a single entry in linear time. We revisit this issue towards the end of this subsection in order to quantify how often these collisions occur.

By limiting the maximum fragment size to 10 words, given the maximum length of Wikipedia page title in our dataset, we can bring the computation time down to $O(n \log d)$ with a coefficient that tends towards 10 as input length increases.

[8] https://golang.org

Our dictionary is 360mb in size, so storing this in memory for a web browser plugin is not possible. We have to take into account that the lookup time will inevitably include a cache miss, and require loading multiple pages from disk per lookup.

The solution is three fold, we can speed up this process by using a Bloom filter to check set inclusion before looking up our entry in our binary search tree. We set a false positive rate of $p = 0.01$, and due to the nature of a Bloom filter, our false negative rate is 0. We calculated the ideal number of bits we would need to attain this false positive rate, using this formula [73]:

$$m = \left\lceil -\frac{d \log p}{2 \log (\log 2)} \right\rceil \tag{108}$$

From this we can calculate the number of hashes to use as:

$$k = \left\lceil \frac{m \log 2}{d} \right\rceil \tag{109}$$

This will use 10.8mb for our Bloom filter, with 7 hash functions, which is a large file size for a Bloom filter but means we only end up using a binary search for looking up fragments that are in the binary tree plus 1% of those that are not. We can perform a lookup operation in a Bloom filter in constant time, so apart from the 10.8mb of memory we will require, there is a lot of benefit to using a Bloom filter.

The second part of the solution was keeping most of the binary search structure on disk, but caching part of the structure so we do not have to load $\log d$ pages from disk for each lookup. We store enough levels of the binary search tree so that we can load a single 100kb block from disk per each lookup. This is currently about 200kb of memory usage on top of our 10.8mb Bloom filter.

The choice for 100kb comes from the third detail of our implementation, that 360mb of disk usage for a browser plugin is also very large, we had originally been compressing this whilst moving it between servers, but realised that we could use *bzip2*, which is block based, to keep this file compressed permanently. We decompress the blocks we need to access on demand, and because a block is 100kb, we built the in-memory binary search tree cache around this block size.

This leads to a respectable memory usage of about 11mb, with 100mb stored on disk. We manually enabled parallelism through the "runtime.GOMAXPROCS" function in Go.

We both compress and initialise our Bloom filter on the Gatekeeper server, before downloading this compressed file to end-user's computers.

We wrote our own Bloom filter implementation so as to ensure that there was no shared state and that it would operate correctly during parallel lookups. We modified Go's built in bzip2 implementation to enable us to extract individual blocks and thus we have a fast but lightweight program for entity extraction.

We were able to identify 53,865 entities that are being mapped to the same normalised string. This means 9,059,127 unique normalised entities map to 9,121,210 page titles. We do not believe this represents a large amount of ambiguity in our lookup process.

We would argue that people often use language in a self-disambiguating way, especially given that document length is short on social networks. For two documents to use the same normalised fragment, but mean two different senses is both improbable, and also will affect the results of this method by a similarly small amount.

Our entity extraction tool communicates with the rest of the application through UNIX sockets. This does not pose a security risk to the user data stored within Gatekeeper, because the entity extraction process only accesses the dictionary from Wikipedia.

6.6 Topic Modelling

Whilst topic modelling is used for our audience predictions within the plugin, we also implemented it as a separate component. This allows us to test the accuracy of topic modelling independently from the rest of the plugin. Our results for this are in Section 7.2.

We have three different methods for building topic models for a corpus, and thus we needed a way of slotting the different algorithms in place, allowing us to fix in place the implementation that was most effective for use in our plugin. Each algorithm was written in a different language, with the language being chosen to play on the strengths of the algorithm.

Our topic modelling is needed based on our hypothesis that a topic will be shared more often between a specific set of friends in a user's social network. By capturing this information we are able to provide more accurate audience predictions to the end user and thus enable them to quickly set privacy policies per status update without having to manually select policies for each of an average of 338 friends.

Because the methods are written in different programming languages a common communication format is required for the Louvain and CJ0_ZS algorithms (we consider LDA separately given that its output is heterogeneous to community detection output).

This format consists of passing a list of edges in the format:

> Node ID 1 (int) ":" Node ID 2 (int) "\n"

Followed by an additional blank new line.

The output format is one of two forms, the first of which indicates the progress:

> "[" Number of communities in system (int) "," Modularity increase (double) "]\n"

The second output format signifies the process is finished, and outputs a mapping from nodes to the communities they belong to:

> Node ID (int) " = " Community ID (int) "\n"

Listing 1: Input and Output from Louvain on a simplistic graph

```
$ node Louvain.js 2
< 1:2
< 2:3
< 3:1
< 7:8
< 8:9
< 9:7
<
> [5,1]
> [4,1]
> [3,3]
> [2,3]
> 1 = 0
> 2 = 0
> 3 = 0
> 7 = 1
> 8 = 1
> 9 = 1
```

Listing 1 shows the input and output from running this program on a very simple graph.

We outline our implementation of the topic modelling in the next three sections.

We need to store these models on disk, but in a way that does not compromise the security of end users. The compromise is to cache the model but to hash the data inside using a randomly generated seed. This means that rainbow tables must be constructed manually in order to decipher the contents. However, we are still able to query the model in O(1) time in order to make audience predictions, without regenerating the model each time. We could encrypt the model, but then a user would have to type the decryption key for every status update, which has all the associated issues around implementing effective password management.

6.6.1 The Louvain Algorithm for Community Detection

The Louvain algorithm (introduced in Section 3.3.1) extracts communities by increasing modularity [33]. The complexity is $O(n \log n)$ where n is the number of nodes in our network.

Our priority during implementation was not speed, given that the algorithm has a fairly low complexity anyway. We wanted a language that allowed us to prototype quickly, and given that most plugins for Chrome are written in JavaScript [64], we decided to to use JavaScript.

We implement the algorithm as per the method we outline in Section 5.2.

6.6.2 The CJ0_ZS Algorithm for Community Detection

JavaScript was appropriate when implementing the Louvain algorithm due to it having a lower complexity than CJ0_ZS. However, when we came to implement CJ0_ZS we weighed up the pros and cons of several compilable languages, primarily C++11 and programming language Go, which we previously utilised for entity extraction.

Within our extension we are able to call into native applications [74], so mixing languages is not a problem as long as the speed from a compiled solution negates the overhead of marshalling data between applications.

C++11 has "std::thread" and "std:priority_queue" in the standard library but the programming language Go has concurrency built into the syntax with language features such as channels, Goroutines[9] and the "container/heap" implementation.

We have implemented CJ0_ZS in Go as described in Algorithm 2.

We ensured correct behaviour of the heap by wrapping the standard implementation with a mutex that prevents our heap invariants from becoming invalidated under race conditions. We handle generating the new Z-Scores and reordering our heap in separate Goroutines. This separation of concerns means that the heap should only be cached for one core, thus reducing cache invalidation. Go will run the Goroutines for generating the Z-Scores across the other cores maximising floating point calculation throughput. We utilise channels with message passing to coordinate this operation.

6.6.3 LDA for Topic Modelling

We want to quantify the quality and stability of our graph based results against those from LDA, which is our baseline.

We use an off the shelf LDA implementation pLDA+ [53]. We select a high number of iterations (10,000) to help ensure that our model converges.

The results from LDA only cover entities, so in order for us to test this method we had to implement the wrapper for this external application separately to that of the Louvain and CJ0_ZS algorithms. This wrapper implements the concatenate method outlined in 5.2 in order to get results for user entities. This enabled us to test and benchmark LDA topic models against our community detection algorithms.

We do not utilise (or ship) pLDA+ with our Gatekeeper plugin, instead we use the CJ0_ZS algorith by default.

6.7 Predicting Audiences for Users of the Plugin

Section 5.3 gives our method for predicting audiences. We chose to implement this in JavaScript, although given that the method is essentially a lot of arithmetic there are many appropriate

[9] http://blog.nindalf.com/how-goroutines-work/

languages for this task. By keeping it in JavaScript, we have the option in the future to move the audience prediction component into the plugin itself; only requiring the application for generating topic models. However, given that cross-browser support for accessing the local filesystem is not standardised, keeping this audience prediction separate to the plugin enables us to build plugins for different browsers, without reimplementing audience prediction or the way we store and handle the topic models.

Whilst the methodology give in Section 5.3 is comprehensive, it is worth mentioning that vector $\mathcal{N}$ is very big, as it contains all the entities in a corpus. Taking the product over $\mathcal{N}$ leaves us with a very small $P(\mathbf{n}_i \mid t, h, \beta)$. We normalise using the same probability distribution, which means that our overall probabilities are not particularly small. However, our intermediate values are, and so we must use log probabilities to represent these values otherwise we would lose too much precision and would not get accurate predictions.

7 Results

Entity extraction allows us to focus on the words and phrases that most likely define a topic, speeding up our generation of topic models, and thus making our method of audience prediction more tractable for large datasets. These audience predictions are the foundation of our plugin that allows users to quickly build privacy policies for their social media content, on a per status update basis. In the next three sections we cover our results for entity extraction (Section 7.1), topic modelling (Section 7.2), and audience prediction (Section 7.3).

7.1 Results for Entity Extraction

As we cover in the implementation Section 6.5, we are using 11mb of memory, with 100mb of Wikipedia data stored on disk. We feel that this is a considerable reduction given that our dictionary is formed from a 10gb compressed Wikipedia data dump.

We also have a lookup time of roughly $O(n + \frac{101rn}{100} \log d)$ where r is the percentage of fragments that we expect to be entities in the text, n is the length of the text, and d is the size of the dictionary.

We evaluate our implementation against that of T-Seg(None) from Ritter et al. [61] (who evaluated their results against Stanford NER [62]). We use an annotated Twitter dataset [63]. Table 5 gives the precision, recall and F_1 scores.

Method	Precision	Recall	F_1
Stanford NER	0.62	0.35	0.44
T-Seg(None)	0.71	0.57	0.63
Proposed solution	0.29	0.62	0.40

Table 5: Results for Stanford NER, T-Seg [61], and proposed solution

We expected a drop in precision due to Stanford NER and T-Seg(None) given that we are utilising every entity on Wikipedia. We discover that the loss of precision often occurs for the following 4 reasons:

1. We match a larger text fragment than the annotators selected.
   For example in the test data we have a tweet: *Sorry but if my baby is gonna sound that silly when reading at age 3 then 'll take my chances ___AND___ wait till they 're 6 . #yourbabycanread.* The annotators listed *baby* as an entity, but our solution extracted *my baby.*

2. We match an entity that has been annotated as two separate entities.
   In one instance we match *"D. C."* where as *"D"* and *"C"* have been annotated as separate entities.

3. We match multiple entities where as annotators have grouped them as one entity.
   In another piece of test data, annotators extracted one entity as being *"JORDAN shoe lottery"* but we match *"JORDAN"*, *"shoe"* and *"lottery"* as separate entities.

4. We match a text fragment that is not an entity. (Usually a text fragment that happens to be an album or song title on Wikipedia)
   One frustrated clubber tweets *"Tryna Get Into Something Tonight"* but we extract *"Get Into Something"* because it happens to be the name of an album by The Isley Brothers.

We could curate our Wikipedia data to remove song titles and album names; this would result in an increase in precision. Ritter et al. utilise a Part-of-Speech tagger, which can help to remove these false positive results.

As will become apparent from the rest of this work, extracted entities are largely symbolic, and are just used to extract a subset of words we think are more likely to represent topics. For this reason it is good to see our method reaches a higher recall than both Stanford NER and T-Seg(None). This means we are more likely to extract important entities from our documents.

If we were to utilise categorical information from Wikipedia as utilised as in the work from Hu et al. [28] on leveraging Wikipedia semantics for text clustering then this lack of precision would have to be addressed.

Instead we present an implementation that is both fast, has a small memory footprint, and takes up little disk space but can still extract entities with a high recall.

7.2 Results for Topic Modelling

We give the results from our methodology for topic modelling as outlined in Section 5.2. Table 6 gives the means, ranges and standard deviations of the number of true positives from the 30 runs of the Louvain, CJ0_ZS, and LDA algorithms.

Algorithm	Mean True Positives	Standard Deviation	Range
Louvain	13.2	0.847	[12,16]
CJ0_ZS	15.8	0.03	[15,16]
LDA	15.6	2.66	[12,21]
LDA-Modified*	19.3	2.48	[13,24]

Table 6: Breakdown of results of each topic modelling algorithm over 30 runs

These results represent the quality of communities as discovered by Louvain, CJ0_ZS, LDA, and LDA-Modified*.

> ***** LDA-Modified was the LDA algorithm as specified in Section 5.2.4, but running it on a corpus preprocessed with a Porter stemmer [45] (Section 3.2.1), as opposed to utilising our Wikipedia based entity extraction in Section 5.1.

We see that when utilising the same preprocessing for both CJ0_ZS and LDA our graph detection method is competitive with LDA, with a much smaller standard deviation, suggesting that the topics it discovers are stabler.

This mirrors Sayyadi and Raschid's [29] results. However we note that the community detection algorithm they utilised was the Girvan-Newman algorithm which has complexity of $O(n^3)$ on a sparse graph and $O((m+n)n^2)$ otherwise [32], with m representing the number of edges, and n representing the number of nodes. Our analysis of the complexity of CJ0_ZS gives us an upper bound of $O(mn\log n)$.

There is a caveat with this dataset, as has been previously mentioned, which is that the Downton Abbey scripts were written by one person pretending to be multiple people. Whilst it could be argued that if all characters had very similar dialogue then the show would not have drawn in 9.2 million viewers [75]. However, it is possible that the writer uses similar language styles for all the characters, or refers to the same entities using the same noun phrases. Doing so would artificially increase the effectiveness of all the algorithms evaluated here.

CJ0_ZS gave significantly better results than the Louvain algorithm, with a modest computation time increase. Once initial topics have been discovered, we can repeat this process infrequently and use the saved topic models when doing inference on new documents. Because we think better quality *ingroup* modelling will result in better topic modelling, the project focuses exclusively on CJ0_ZS instead of the Louvain algorithm in the other sections.

We note that using LDA with a traditional pre-processing step of stemming and tokenising consistently yields a higher true positive rate. This suggests that by focusing on entities, inadvertently, we could be missing additional information that would improve our topics, such as *slang*, *colloquialisms* and *dialects*. We introduced entity extraction as a method for reducing the number of nodes and edges in our graph to speed up computation time. We see 5,228 noun phrases vs. roughly 50,000 words in this dataset. Changing our preprocessing step would increase the runtime and memory consumption of the priority queue in the CJ0_ZS algorithm by up to two orders of magnitude. We explore algorithms that have the potential to converge quickly regardless of the number of nodes in the future work outlined in Section 8.1.

7.3 Results for Audience Predictions

We follow our methodology from Section 5.3; testing audience prediction using real-life data collected from Facebook and Twitter.

We give the ROC curves for each dataset and compare it against the state-of-the-art LDA. We give the AUC (area under curve) which is the area under the ROC curve. This represents the probability that a given pair of audience members will be correctly ordered; which is to say the audience member with the higher probability will correctly be more likely to appear in the audience of a document.

7.3.1 Quantifying Audience Prediction Accuracy using Topic Modelling

Figure 22 shows our results from the Twitter test data using CJ0_ZS clustering in the form of a ROC curve.

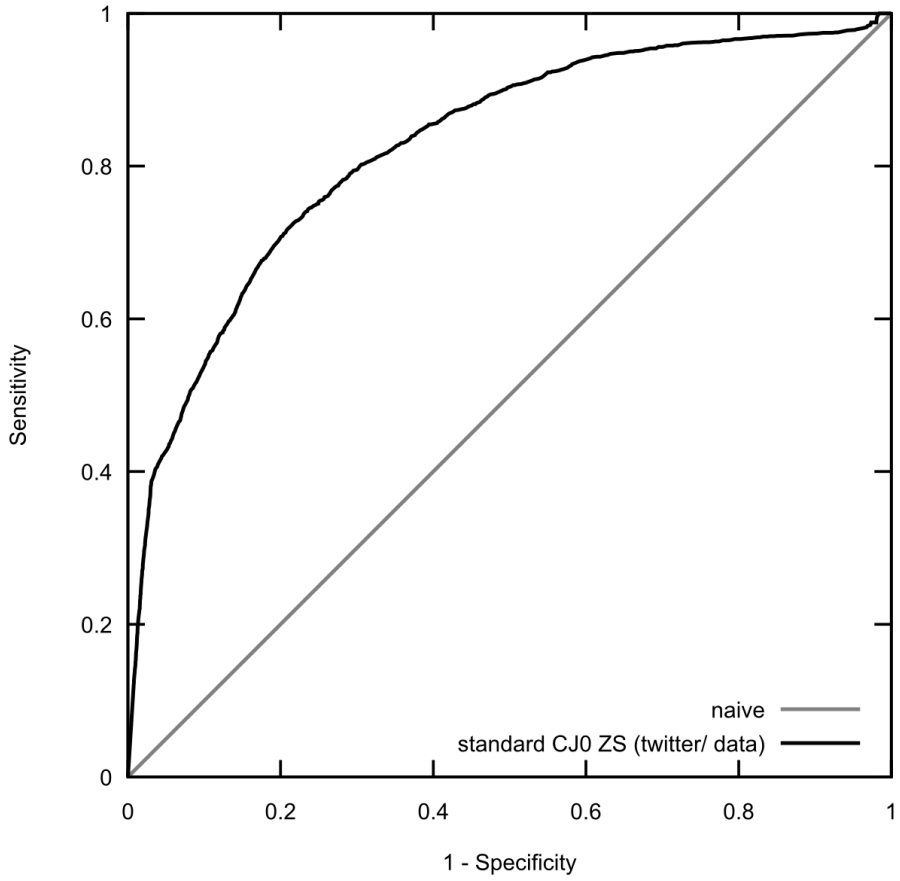

Figure 22: ROC curves for Twitter data using CJ0_ZS

For our implementation, we get an AUC of:

$$0.826 \pm 0.00118$$

Given that Facebook's current policy is to publish new status updates with the same privacy settings as the last status posted, which has an accuracy of 67% [26], swapping to our methodology utilising audience prediction would yield a relative improvement of 23%.

Given a pair of random possible audience members, the AUC indicates the percentage of pairs where the ordering correctly reflects that an audience member who was mentioned is more probable in our method.

We motivate the probabilities being generated in Figure 23, the users are ordered over the x-axis by mean probabilities over all test data. We see high variability between probabilities being generated for different test data. We can also see that for all test data the minimum probability is roughly 0, which indicates that even users who are normally highly probable audience members wont necessarily always be highly probable audience members for all tweets posted.

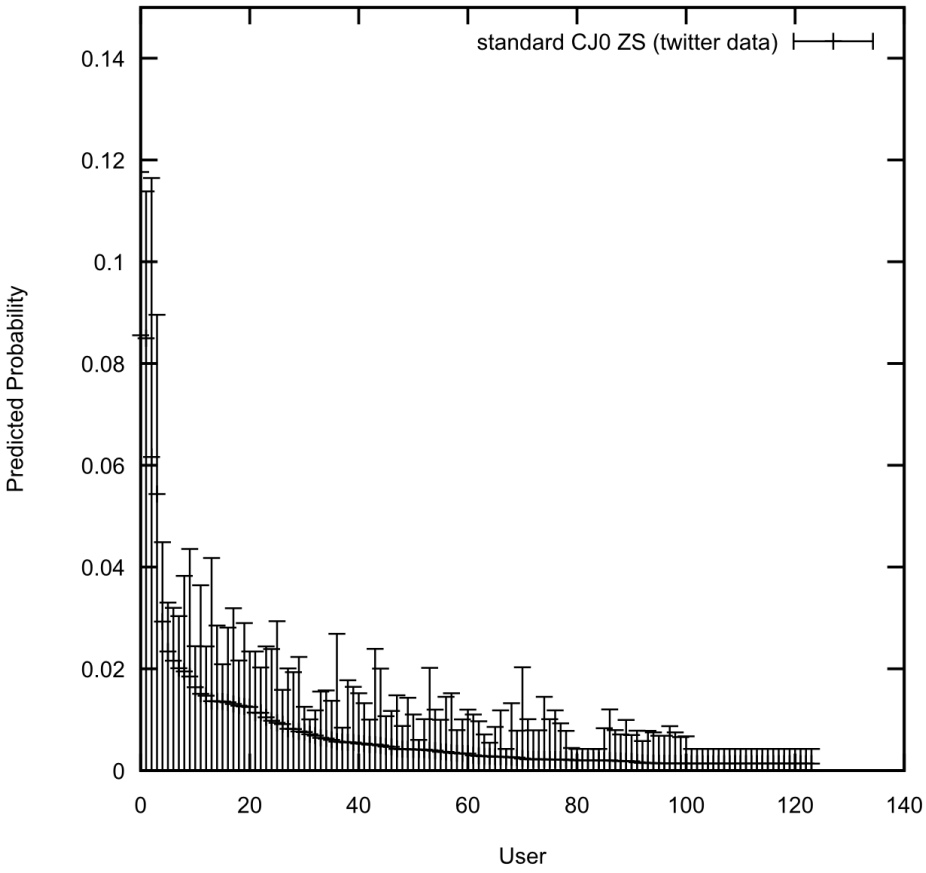

Figure 23: Graph showing mean, min, and max of predictions per user for all test data

7.3.2 Quantifying Improvement with Facebook Data

We also generated results from our Facebook dataset; Figure 24 shows our ROC curve for this. We see that it appears the resolution of the graph towards the origin is low. This is not due to a low sampling rate, but because a small number of people in the dataset have been messaged exactly the same number of times, thus the probabilities for these people are all exactly the same for our null model without topic information.

To give the AUC values, our method achieved a value of:

$$0.974 \pm 0.000294$$

When removing topic information we get an AUC value of:

$$0.871 \pm 0.0184$$

These results are better than our results from Twitter, with a relative improvement when swapping to use topic information of 11.8%. This could be due to the Facebook dataset being from a longer

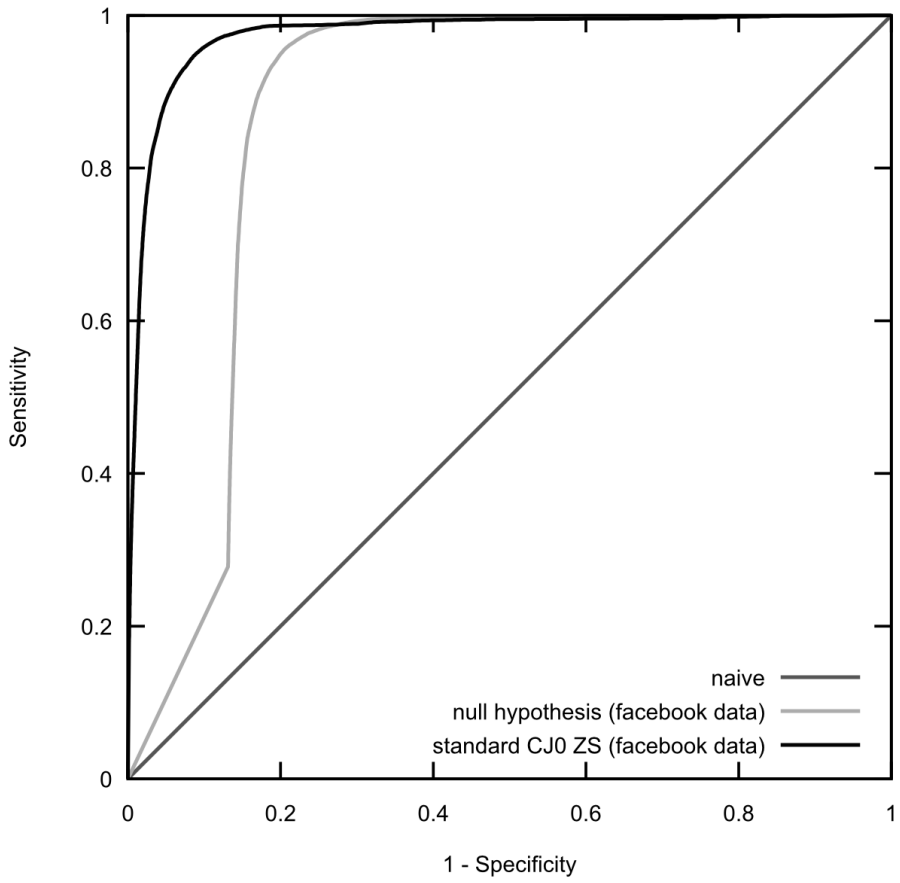

Figure 24: ROC curves for CJ0_ZS on Facebook Data

period of 8 years as opposed to our Twitter dataset's 7 years, and the fact that on Facebook private messages you have to specify a recipient, where as many tweets do not mention anyone at all, thus we have more information about *ingroups*.

Given that our plugin is for Facebook these results support our hypothesis that audience predictions will help the user make accurate privacy policy decisions on a per status basis.

7.3.3 Comparison to LDA

In order to make LDA comparable to our implementation of CJ0_ZS, we have pre-processed both datasets in the same way: by extracting named entities and discarding everything else.

We make this change by swapping $\mathcal{T}$ to use our $C(v,t)$ from LDA, rather than from CJ0_ZS. The results are presented in Figure 25. Our AUC for LDA is just slightly below our AUC for CJ0_ZS:

$$0.806 \pm 0.000950$$

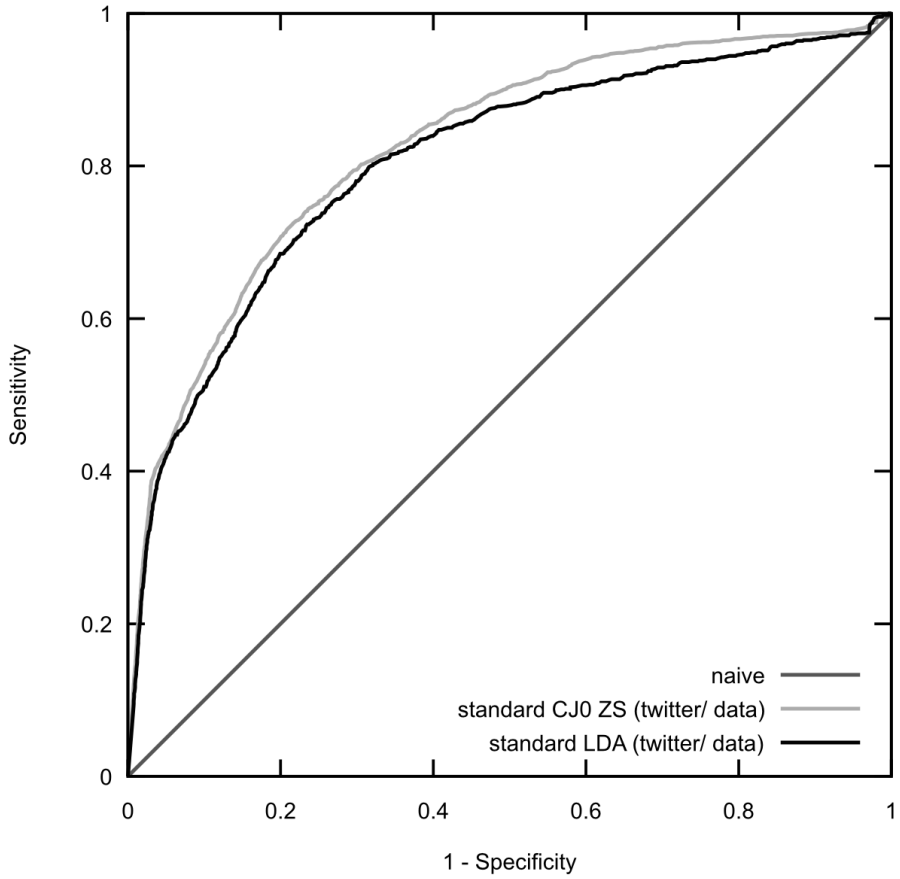

Figure 25: CJ0_ZS results for exact probability model compared to LDA results

As compared to the previously stated performance of our clustering method:

$$0.826 \pm 0.00118$$

We feel this shows that using clustering is competitive with LDA. However, as noted in Section 7.2 pre-processing using entity extraction seriously undermines the performance of LDA, we are still investigating whether our methodology will remain more accurate than LDA when pre-processing using stemming and tokenising.

8 Discussion

Recall from Section 7.1, our entity extraction method has the highest recall of the three approaches covered, at 0.62, but the lowest precision at 0.29. This lack of precision is not of dramatic significance to our methodology, given that we do not want to miss any entities, and that having many entities is not detrimental to the results of our methodology.

In Section 5.2.6 we see that CJ0_ZS is competitive with LDA utilising the same preprocessing step of entity extraction using *lookup* from Section 5.1. With CJ0_ZS having a true positive rate of 0.59 to LDA's true positive rate of 0.58, when classifying users into social groups.

In Section 7.2 our results for LDA using a traditional Porter stemming and stop word removal preprocessing step yielded a higher true positive rate of 0.71. We consider the Expectation-Maximisation algorithm in 8.1, which would be able to scale to the number of words our corpus contains, and thus not require the entity extraction step.

We note that in the same section we discovered that the standard deviation of the community detection algorithms was smaller than that of LDA, which signifies that CJ0_ZS has a very stable number of true positives across multiple runs with a deviation of 0.03, compared to 2.66 for LDA. This would suggest if stability across repeated runs is important that community detection algorithms might be a more appropriate choice for topic modelling. If we showed an end user the groupings of friends we generate, then if our communities were unstable, the user may think they have very unstable friendship groups. As it happens we do not display this information to an end user, but they may still sense instability when writing similar statuses over a period of time when they get vastly different audience predictions.

Our hypothesis is that modelling these social groups, and utilising social groups during topic modelling will improve the quality of the topics. There is a copy of the communities discovered by CJ0_ZS in Appendix A. We present our results of our topic modelling compared to LDA in Section 7.3.3, with a true positive rate of 0.806 for our methodology, and 0.826 for the topics generated by LDA. Whilst this is not a significant increase, we can see that our methodology is competitive with that of LDA.

Our results of 0.971 for the AUC for our Facebook dataset are encouraging. This means for a pair of audience members, 97.1% of the time our algorithm will order them correctly, with a higher probability for an audience members have actually being mentioned in our test data. With Facebook users having on average 338 friends [6], this means that roughly only 8 pairs will be incorrect for the average user of our plugin.

Our graph based approach, whilst utilising a different model to Sayyadi and Raschid's methodology [29], mirrors their results: that graph based, community detection algorithms are competitive with LDA.

Sinha et al. [26] utilised a machine learning based approach using MaxEnt. They found that the Facebook mechanism of defaulting to the privacy policy used for the last status only yielded an accuracy of 67%. Using past status updates in the form of *n-grams* and the privacy policies that

had been attached to them also performed poorly but they obtained high accuracy when using a training set where participants had explicitly defined the privacy policy, yielded an accuracy of 81%.

The work from Sinha et al. [26] highlights two assumptions from our work:

1. We take our ground truth for our Facebook dataset from private Facebook conversations, mirroring our assumption that the privacy policies of private messages are a good indicator for the privacy policies of status updates (although we note that status updates tend to have a bigger audience).

2. We still need a user to determine ϑ as used in our basic utility function. Where as Sinha et al. [26] are determining complete privacy policies for users automatically. Our methodology as it stands requires user interaction to specify this boundary in some way. Whilst we use tick boxes in our plugin to refine an audience prediction and turn it into a privacy policy, ideally this could be calculated using the status itself. Users do not like seeing other people's private information [6]. However, it is not trivial to utilise *sentiment analysis* (Section 3.2.3) to discover how widely a status should be disseminated.

We are able to utilise the friendship groups discovered in Section 5.2.1, despite not having access to the underlying friendship lists on the Facebook platform, this opens up possibilities for existing privacy solutions that utilise these friendship groups and enable them to continue to work [14, 15, 16].

We presented a model that is general enough that it could be applied to other conversational corpuses other than Twitter, Facebook, and screen plays. We speculate that being able to avoid using entity extraction would yield an improvement in line with those seen for LDA in Section 5.2.6 and would open up this approach for use processing larger data sets.

Tang et al. [27] proposed a probabilistic model for predicting likely advert click-throughs. They found that combining topic models with advert impression logs improved the quality of both advert click-through and topic models. We think this domain is somewhat similar to our own, except where instead of adverts with a topic distribution we have messages that have a latent topic distribution that are explicitly shared amongst users. They used the EM algorithm and we think that this algorithm would also be appropriate for calculating our topic models that include audience information.

8.1 Future Work

We propose two related and promising avenues for future development:

1. As mentioned above, a full probabilistic model for generating *registers* (the combination of social groups and the language they shared) and not just using them would be an appropriate candidate for the Expectation-Maximisation algorithm. We would expect results inline with the boost in accuracy observed by Tang et al. [27]. This will remove the limitation on corpus

size, and allow our method to be applied to larger data sets that would be encountered outside of online social networks.

2. The name of this plugin comes from Barzilai-Nahon's work "Toward a Theory of Network Gatekeeping: A Framework for Exploring Information Control" [24]. The survey contains a wealth of possibilities for Gatekeeping. Currently Gatekeeper is only able to *withhold* information, but, for example, there is the potential for *addition* (to add additional information to a status update for select users) and *timing* (to release a status update at a good time to take advantage of the largest or smallest audience on the network). Our methodology is general enough to work with images as documents, and with developments from Karpathy and Fei-Fei [76] on determining the content of images, it is feasible for Gatekeeper to be used in these situations. There are limitations in the current Facebook API, but open source implementations of social networks like Diaspora*[10] could provide an ideal testing ground for future online social network gatekeeping developments.

[10] https://diasporafoundation.org

9 Conclusions

Privacy and over-sharing concerns plague online social networks [4, 5, 6]. We aspired to try and resolve this by building a plugin for online social network users enabling and empowering them to make privacy decisions for the real-time natural language based content they were publishing. There is already support for this within the Facebook API, but users are critical that classifying their friendships takes too long, and that the automatically created groupings of friends seem incomplete [12]. We devised a novel approach based on work within the social sciences building a *register* graph. This bipartite *register* graph of friends and the language they share in common is the core of our method. We split the task into three components: entity extraction, topic modelling, and audience prediction. For each component we explored and built on the theory, implemented it, and evaluated the results against existing state-of-the-art.

We show that topic models generated utilising *register* graphs are competitive with LDA, and that these topic models have a higher stability in accuracy over multiple runs. We show that these topic models can be utilised for audience prediction on conversational Twitter data, conversational Facebook data and screen plays, yielding relative improvement for predicting audiences for content with an accuracy of 82.6 - 97.4%. Furthermore, we took multiple implementation decisions along the way that enabled us to package this powerful audience prediction engine into a format that can be run on the commodity hardware owned by end-users, integrating directly into the Facebook website; making privacy accessible for users of online social networks again.

The motivation for this work is best summarised by the words of Carmen Hermosillo [77]. To quote "Pandora's Vox: On Community in Cyberspace":

> ‘‘when i went into cyberspace i went into it thinking that it was a place like any other place and that it would be a human interaction like any other human interaction. i was wrong when i thought that."

The theory of gatekeeping has been around in the literature since 1980 [10]; and a part of the analogue for much longer; this project represents a small part of a wider movement to rebuild privacy for an increasingly digital world; to rebuild human interaction like any other human interaction.

A Communities from Twitter Data

We give a tag cloud for each of the 15 communities discovered in the Twitter dataset in Figures 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, and 40. We include entities that were mentioned at least twice, and we limit each tag cloud size to a maximum of 75 terms.

We see that in almost every community there is the name of a programming language, and the name of a town or city. This reflects the fact that within this user's social network these entities have a common understanding and are thus used by everyone, but the community detection algorithm has hard assigned them to a community.

We can see community 3 (Figure 28) contains lots of terms associated with music, such as "artist", "arts", "audio", "dj", "demos", "festival", "music", "my music", and "remix".

In community 8 (Figure 33) we can see lots of terms about technology start-ups such as: "dublin", "jobs", "sales", "soundwave", "startup", "vc", "android", and "arch".

In community 9 (Figure 34) we can see LGBT terms such as "gay", "guy", "pronoun", "russia", "vice", along with other associated words.

For the other communities it is difficult to find a single unifying theme. The reasons for this are listed in Section 5.2.1.

Figure 26: Community 1

Figure 27: Community 2

Figure 28: Community 3

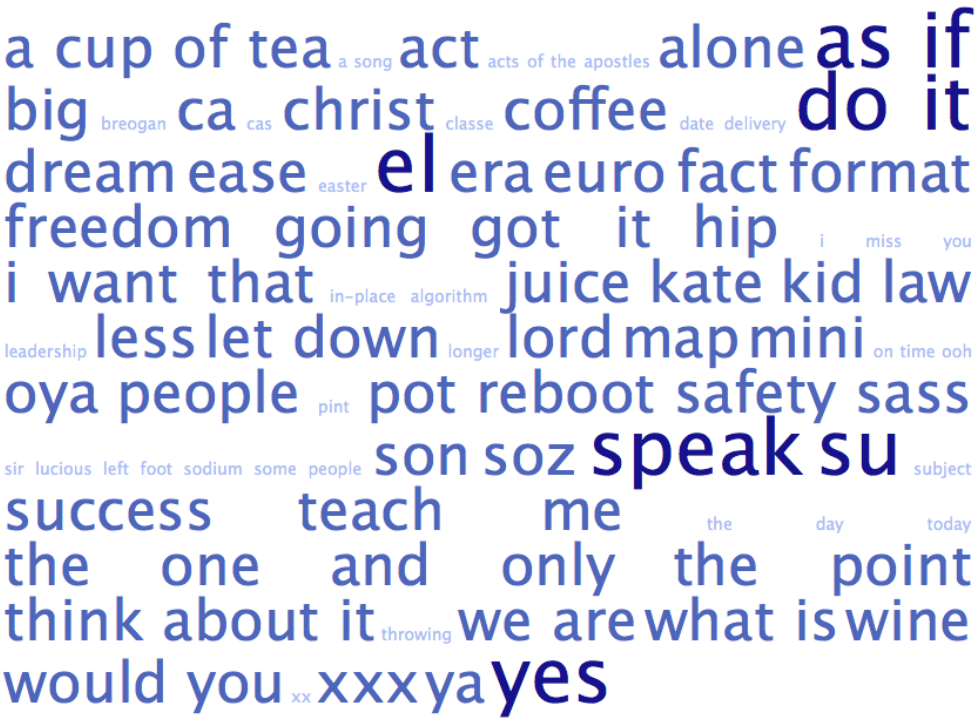

Figure 29: Community 4

Figure 30: Community 5

Figure 31: Community 6

Figure 32: Community 7

Figure 33: Community 8

Figure 34: Community 9

Figure 35: Community 10

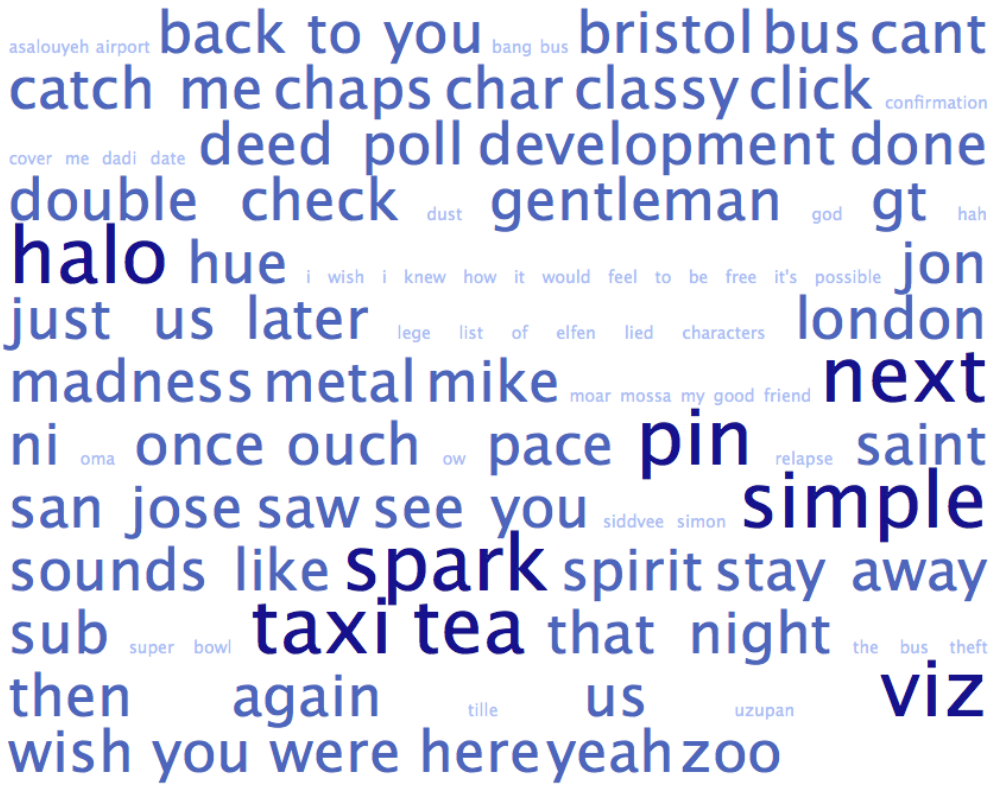

Figure 36: Community 11

Figure 37: Community 12

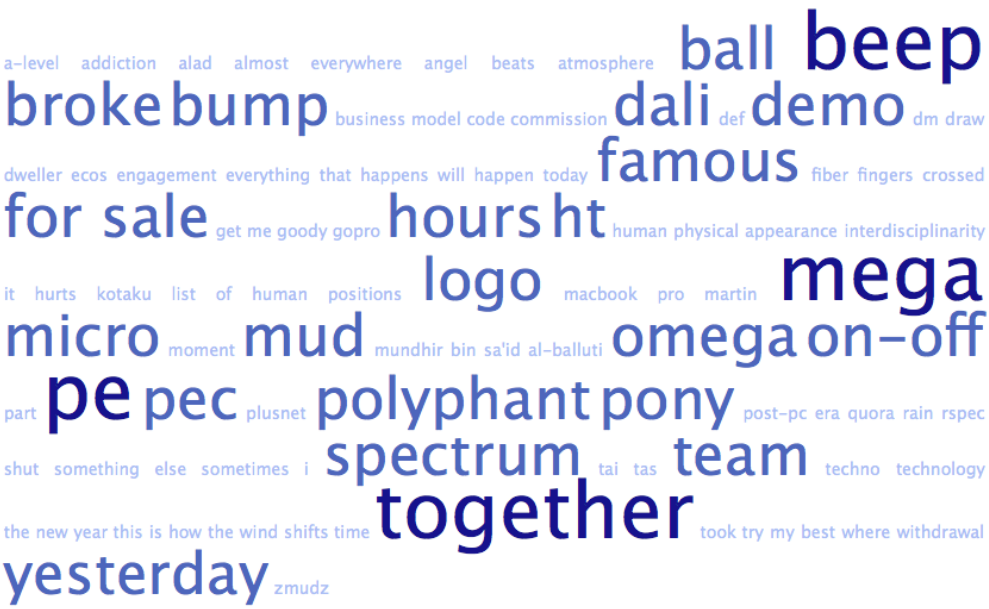

Figure 38: Community 13

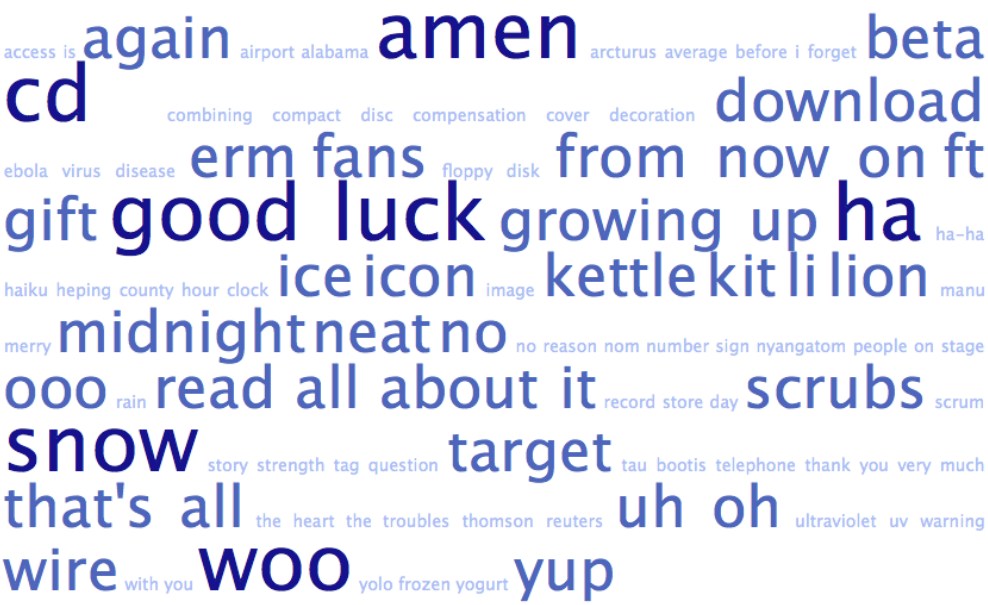

Figure 39: Community 14

Figure 40: Community 15

B Glossary

audience A subset of users amongst which a document was explicitly shared. In the case of tweets this will be users explicitly @-mentioned in the text of a document. In Facebook private conversation history this will be the recipient of a message. In screen plays this is someone who is in the same scene as the character speaking. 22, 33, 34, 50–52, 54, 56–58, 60, 61, 66, 72

$\mathcal{D}$ A corpus consisting of a set of documents d_i. 22, 33, 50, 51, 53, 54, 66

document A document in our body of work comes from online social networks, thus tends to be short in length and informal in style. A document d_i is of the form $d_i : (h_i, \mathbf{m}_i, \mathbf{n}_i)$ where h_i is the *publisher* of the ith document, $\mathbf{m}_i : \{\top, \bot\}^{|\mathcal{U}|}$ is a vector indicating whether a user is the audience of that document and $\mathbf{n}_i : \{\top, \bot\}^{|\mathcal{N}|}$ is a vector indicating which entities are in a document. 4, 7, 12–15, 21, 22, 26, 29–34, 44, 49–57, 66, 68

entity A noun or noun phrase that denotes a concept. Examples include words that denote places, names, organisations, times and dates. 22–31, 33, 34, 44, 49–54, 58, 66

ingroup A social group to which a person psychologically identifies as being a member. For example someone may identify as being a part of a trade, profession, institution, or academic field.. 20, 31, 33, 45, 49, 74

$\mathcal{N}$ The universal vector of all entities within a corpus document. We sometimes break convention and use set notation to iterate through the entities instead of using index notation where this will ease understanding of the formula. 33, 34, 44, 50, 51, 53, 57, 58

publisher A user who has published a document, for Twitter and Facebook these are users who have published the document, for screen plays, the publisher is the character who spoke that line. 22, 33, 34, 51–54, 57, 66

register A collection of words, jargon, entities, colloquialisms or slang that represents language that requires a shared understanding of the term or phrase between conversation participants before being used. 2, 31–33, 58

$\mathcal{T}$ The universal set of topics discovered in a corpus $\mathcal{D}$, as defined in Section 5.2.1. 34, 44, 46, 52, 53, 55–57, 59–61

$\mathcal{U}$ The universal vector of all publishers and audiences in a corpus $\mathcal{D}$. We sometimes break convention and use set notation to iterate through the users instead of using index notation where this will ease understanding of the formula. 33, 34, 50–52, 54, 58, 66

References

[1] B. Johnson, "Privacy no longer a social norm, says facebook founder. [online]," (http://www.theguardian.com/technology/2010/jan/11/facebook-privacy), Jan 2010.

[2] M. Kirkpatrick, "Facebook's zuckerberg says the age of privacy is over. [online]," (http://readwrite.com/2010/01/09/facebooks_zuckerberg_says_the_age_of_privacy_is_ov), Jan 2010.

[3] H. A. Popkin, "Privacy is dead on facebook. get over it. [online]," (http://www.nbcnews.com/id/34825225/ns/technology_and_science-tech_and_gadgets/t/privacy-dead-facebook-get-over-it/#.VMUWi8ZLFEc), Jan 2010.

[4] S. Stieger, C. Burger, M. Bohn, and M. Voracek, "Who commits virtual identity suicide? differences in privacy concerns, internet addiction, and personality between facebook users and quitters," *Cyberpsychology, Behavior, and Social Networking*, vol. 16, no. 9, pp. 629–634, 2013.

[5] M. Madden, "Public perceptions of privacy and security in the post-snowden era. [online]," (http://www.pewinternet.org/2014/11/12/public-privacy-perceptions/), Nov 2014.

[6] A. Smith, "Pew research center. [online]," (http://www.pewresearch.org/fact-tank/2014/02/03/6-new-facts-about-facebook/), Feb 2014.

[7] S. Kairam, M. Brzozowski, D. Huffaker, and E. Chi, "Talking in circles: selective sharing in google+," in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2012, pp. 1065–1074.

[8] R. Dey, Z. Jelveh, and K. Ross, "Facebook users have become much more private: A large-scale study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE, 2012, pp. 346–352.

[9] S. D. Warren and L. D. Brandeis, "The right to privacy," *Harvard law review*, pp. 193–220, 1890.

[10] L. C. Freeman, "The gatekeeper, pair-dependency and structural centrality," *Quality and Quantity*, vol. 14, no. 4, pp. 585–592, 1980.

[11] S. Tow, "Facebook. [online]," (https://www.facebook.com/notes/facebook/ improved-friend-lists/10150278932602131), Sep 2011.

[12] S. Jones and E. O'Neill, "Feasibility of structural network clustering for group-based privacy control in social networks," in *Proceedings of the Sixth Symposium on Usable Privacy and Security*. ACM, 2010, p. 9.

[13] M. E. Brashears, "Humans use compression heuristics to improve the recall of social networks," *Scientific reports*, vol. 3, 2013.

[14] L. Fang and K. LeFevre, "Privacy wizards for social networking sites," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 351–360.

[15] F. Adu-Oppong, C. K. Gardiner, A. Kapadia, and P. P. Tsang, "Social circles: Tackling privacy in social networks," in *Symposium on Usable Privacy and Security (SOUPS)*, 2008.

[16] L. Fang, "Mechanisms of controlled sharing for social networking users," Ph.D. dissertation, The University of Michigan, 2013.

[17] Facebook, "Graph api reference friend list /friendlist [online]," (https://developers.facebook.com/docs/graph-api/reference/v2.3/friendlist).

[18] J. Brustein, "Bloomsberg businessweek. [online]," (http://www.businessweek.com/articles/2014-07-23/heres-how-much-time-people-spend-on-facebook-daily), Jul 2014.

[19] R. W. Neal, "Facebook gets older: Demographic report shows 3 million teens left social network in 3 years. [online]," (http://www.ibtimes.com/facebook-gets-older-demographic-report-shows-3-million-teens-left-social-network-3-years-1543092), Jan 2014.

[20] R. Dunbar, "You've got to have (150) friends. [online]," (http://www.nytimes.com/2010/12/26/opinion/26dunbar.html), Dec 2010.

[21] M. B. Brewer, "Multiple identities and identity transition: Implications for hong kong," *International Journal of Intercultural Relations*, vol. 23, no. 2, pp. 187–197, 1999.

[22] R. Mehrotra, S. Sanner, W. Buntine, and L. Xie, "Improving lda topic models for microblogs via tweet pooling and automatic labeling," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2013, pp. 889–892.

[23] W. X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, and X. Li, "Comparing twitter and traditional media using topic models," in *Advances in Information Retrieval*. Springer, 2011, pp. 338–349.

[24] K. Barzilai-Nahon, "Toward a theory of network gatekeeping: A framework for exploring information control," *Journal of the American society for information science and technology*, vol. 59, no. 9, pp. 1493–1512, 2008.

[25] A. Squicciarini, S. Karumanchi, D. Lin, and N. DeSisto, "Identifying hidden social circles for advanced privacy configuration," *Computers & Security*, vol. 41, pp. 40–51, 2014.

[26] A. Sinha, Y. Li, and L. Bauer, "What you want is not what you get: predicting sharing policies for text-based content on facebook," in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM, 2013, pp. 13–24.

[27] J. Tang, N. Liu, J. Yan, Y. Shen, S. Guo, B. Gao, S. Yan, and M. Zhang, "Learning to rank audience for behavioral targeting in display ads," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 605–610.

[28] J. Hu, L. Fang, Y. Cao, H.-J. Zeng, H. Li, Q. Yang, and Z. Chen, "Enhancing text clustering by leveraging wikipedia semantics," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2008, pp. 179–186.

[29] H. Sayyadi and L. Raschid, "A graph analytical approach for topic detection," *ACM Transactions on Internet Technology (TOIT)*, vol. 13, no. 2, p. 4, 2013.

[30] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[31] S. Cucerzan, "Large-scale named entity disambiguation based on wikipedia data." in *EMNLP-CoNLL*, vol. 7, 2007, pp. 708–716.

[32] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[33] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[34] R. Rotta and A. Noack, "Multilevel local search algorithms for modularity clustering," *Journal of Experimental Algorithmics (JEA)*, vol. 16, pp. 2–3, 2011.

[35] M. Naaman, J. Boase, and C.-H. Lai, "Is it really about me?: message content in social awareness streams," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 2010, pp. 189–192.

[36] Google, "Google+. [online]," (https://support.google.com/plus/answer/ 1297219?hl=en-GB).

[37] Twitter, "Twitter help centre. [online]," (https://support.twitter.com/articles/20169886-protecting-and-unprotecting-your-tweets.

[38] T. Telegraph, "Average twitter user is an an american woman with an iphone and 208 followers. [online]," (http://www.telegraph.co.uk/technology/news/9601327/Average-Twitter-user-is-an-an-American-woman-with-an-iPhone-and-208-followers.html).

[39] T. teams up with SoundCloud and iTunes to play audio within tweets. [Online], "Stuart dredge," (http://www.theguardian.com/technology/2014/oct/17/twitter-soundcloud-itunes-audio-tweets).

[40] "How twitter gets in the way of knowledge - buzzfeed news. [online]," http://www.buzzfeed.com /nostrich/how-twitter-gets-in-the-way-of-research#.qo8nngD7WZ, (Visited on 06/03/2015).

[41] M. Melanson, "Twitter kills the api whitelist: What it means for developers & innovation - readwrite. [online]," http://readwrite.com/2011/02/11/twitter_kills_the_api_whitelist_what_it_means_for, (Visited on 06/03/2015).

[42] J. Koetsier, "Twitter api updates: more authentication, fewer tweets, more rules, certification, and ... talk to the hand | venturebeat | dev | by john koetsier. [online]," http://venturebeat.com/2012/08/16/twitter-api-updates-more-authentication-fewer-tweets-more-rules-certification-and-talk-to-the-hand/, (Visited on 06/03/2015).

[43] S. Tow, "Timeline: Now available worldwide. [online]," https://www.facebook.com/notes/facebook/timeline-now-available-worldwide/10150408488962131, (Visited on 06/03/2015).

[44] J. H. Martin and D. Jurafsky, "Speech and language processing," *International Edition*, 2000.

[45] C. Umbel, R. Ellis, and R. Mull, "Naturalnode/natural. github. [online]," https://github.com/NaturalNode/natural, (Visited on 06/03/2015).

[46] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003, pp. 142–147.

[47] R. Florian, A. Ittycheriah, H. Jing, and T. Zhang, "Named entity recognition through classifier combination," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003, pp. 168–171.

[48] J. W. Pennebaker, C. K. Chung, M. Ireland, A. Gonzales, and R. J. Booth, "The development and psychometric properties of liwc2007," 2007.

[49] A. B. Warriner, V. Kuperman, and M. Brysbaert, "Norms of valence, arousal, and dominance for 13,915 english lemmas," *Behavior research methods*, vol. 45, no. 4, pp. 1191–1207, 2013.

[50] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[51] L. Developers, "lda 0.3.2: Python packages. [online]." (https://pypi.python.org/pypi/lda).

[52] K. Becker, "Lda on npm. [online]." (https://www.npmjs.com/package/lda).

[53] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun, "Plda+: Parallel latent dirichlet allocation with data placement and pipeline processing," *ACM Transactions on Intelligent Systems and Technology, special issue on Large Scale Machine Learning*, 2011, software available at http://code.google.com/p/plda.

[54] G. Tang, Y. Xia, W. Wang, R. Lau, and T. F. Zheng, "Clustering tweets using wikipedia concepts."

[55] S. Fortunato and M. Barthélemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 2007.

[56] S. B. McGrayne, *The theory that would not die: how Bayes' rule cracked the enigma code, hunted down Russian submarines, & emerged triumphant from two centuries of controversy.* Yale University Press, 2011.

[57] "Probabilistic programming and bayesian methods for hackers," http://nbviewer.ipython.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter1_Introduction/Chapter1.ipynb, (Visited on 06/05/2015).

[58] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, pp. 415–444, 2001.

[59] X. Han, L. Wang, S. Park, A. Cuevas, and N. Crespi, "Alike people, alike interests? a large-scale study on interest similarity in social networks," in *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*. IEEE, 2014, pp. 491–496.

[60] B. Kenny and W. Savage, *Language and development: Teachers in a changing world*. Longman Pub Group, 1997.

[61] A. Ritter, S. Clark, O. Etzioni *et al.*, "Named entity recognition in tweets: an experimental study," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1524–1534.

[62] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[63] A. Johannsen, D. Hovy, H. Martinez, B. Plank, and A. Søgaard, "More or less supervised super-sense tagging of twitter," in *The 3rd Joint Conference on Lexical and Computational Semantics (*SEM)*, Dublin, Ireland, 2014.

[64] "Content scripts - google chrome. [online]," https://developer.chrome.com/extensions/content_scripts, (Visited on 06/08/2015).

[65] T. Bayes, D. W. Bunn, H. Raiffa, R. Schlaifer, and D. Von Winterfeldt, "An essay toward solving a problem in the doctrine of chances," *Philosophical Transactions of the Royal Society of London*, vol. 53, 1984.

[66] "Rest apis | twitter developers. [online]," https://dev.twitter.com/rest/public, (Visited on 06/09/2015).

[67] "Facebook platform upgrade guide. [online]," https://developers.facebook.com/docs/apps/upgrading#upgrading_v2_0_user_ids, (Visited on 06/14/2015).

[68] "node/node_root_certs.h at master · joyent/node · github. [online]," https://github.com/joyent/node/blob/master/src/node_root_certs.h, (Visited on 06/14/2015).

[69] "A javascript library for building user interfaces | react. [online]," https://facebook.github.io/react/, (Visited on 06/14/2015).

[70] "Grunt: The javascript task runner. [online]," http://gruntjs.com/, (Visited on 06/14/2015).

[71] "Browserify. [online]," http://browserify.org/, (Visited on 06/14/2015).

[72] "Babel · the compiler for writing next generation javascript. [online]," https://babeljs.io/, (Visited on 06/14/2015).

[73] "Bloomfilter calculator. [online]," http://hur.st/bloomfilter?n=4&p=1.0E-20, (Visited on 06/07/2015).

[74] "Native messaging - google chrome. [online]," https://developer.chrome.com/extensions/nativeMessaging, (Visited on 06/08/2015).

[75] "Downton abbey return watched by 8.1m viewers - bbc news. [online]," http://www.bbc.co.uk/news/entertainment-arts-29310525, (Visited on 06/08/2015).

[76] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," *arXiv preprint arXiv:1412.2306*, 2014.

[77] C. Hermosillo, "Folksonomy | introducing humdog: Pandora's vox redux. [online]," http://folksonomy.co/?permalink=2299, (Visited on 06/11/2015).