Resolving Student Entities in the Facebook Social Graph

Jim Sproch
jsproch@stanford.edu
Stanford University

Jason Jong
jjong@stanford.edu
Stanford University

Abstract

Despite the popularity of social networking sites and the abundance of information available on the internet, finding a holistic overview of an individual remains difficult. Profiles are often fragmented across sites, so users must issue queries to several different services and manually combine the results. Search engines like Google and Pipl utilize keyword indices to suggest a list of potentially relevant documents, but lack the semantic understanding to accurately conflate these documents into a single holistic overview. Historically, the tedious task of examining the candidate results to extract & merge features has been punted off to the end user. Our system aims at accurately identifying equivalence groups among heterogeneous sets of user profiles, thus allowing the system to present a single aggregate view of the underlying data.

Introduction

Over the last decade, the explosive growth of social networking services like Facebook has resulted in a new wave of social networking sites, as competitors scramble to integrate “social” features into their sites. Service providers are reluctant to share data with their competitors, which means that each social network must be built from the ground up, resulting in an extraordinarily fragmented ecosystem. An individual’s identity becomes split across several different spheres of communication, which actually adversely affects the sense of community that the companies were trying to build in the first place.

Users must choose which networks they intend to join, and a painful registration process means that it’s impractical for every user to be on every network. Because different social networks often target different use cases, or even different user populations, the information on a given network is different from information available on another competing network. Therefore, an individual’s profile page on a single service gives an incomplete picture of the user’s online presence. Manually digging through several incomplete profiles is a tedious task.

Search engines like Google[1] and Pipl[2] allow users to specify a query, and the search engine will try to find documents related to that query. This mechanism is most effective when a single document provides all the desired information, but is less useful when the desired information is fragmented across several documents.

We present a system that performs a pairwise comparison of profiles from two or more disparate social networks to determine profile equivalence groups. Profiles within an equivalence group can be merged and presented as a single view to a user, allowing users to get a holistic overview of an individual, without manually re-executing the same query on a dozen different social networks.

[1] Google is a popular general purpose search engine, and is often one of the first places people try when searching for information about an individual.

[2] Pipl is a popular search engine specialized to finding documents about individuals.

After discussing the steps taken to ensure our system observes adequate privacy constraints, and surveying the related works, we examine the implementation of our entity resolution system, which utilizes a series of map-reduce jobs to batch process hundreds of thousands of profiles and find pairwise equivalence groups. We then define our evaluation criteria, and examine the effectiveness of our system by finding college students within Facebook's social network.

Privacy

When working with social data, we must confront the issue of privacy, and ensure we are not exposing any confidential information. Our objective is to make it easier for a user to find an accurate and complete overview of an individual's online presence, without revealing any previously unobtainable information. To achieve this, we opted to import ONLY publicly available information. Our system does not consider/utilize LDAP information marked as restricted, and only performs public bind lookups. Similarly when pulling data from social networks, we only read publicly visible fields. This ensures we are maintaining the strictest privacy and confidentiality standards.

Related Work

Even before the popularity of social networking, there have been substantial efforts to make it easier to find useful information about individuals. Phone books (and later, online versions like whitepages.com) are perhaps an early example, allowing users to find a person's address and phone number. Companies and universities started utilizing LDAP (Lightweight Directory Access Protocol) servers to disseminate directory information, often including additional fields like facsimile numbers and email addresses.

As the world became more interconnected, and the internet became a viable communication mechanism, social networking sites like MySpace and Facebook started allowing users to to manage their own profiles, including photos and posts to friends. The rapid influx of users and money caused other companies Google (G+) and Quora to enter the market. Each of these competing services strives to create a sense of community and identity for their users.

In the fallout of the massive social-networking-land-grab, many groups have been working to create systems to better understand social data. For instance, D-Dupe builds a largely automated system that utilizes "unsupervised clustering task, where each cluster represents references that map to the same entity" [2]. Then, D-Dupe users "resolve ambiguities either by merging nodes or by marking them distinct" [4]. D-Dupe can assist users in resolving entities, but the process still requires humans to examine the candidate pairs and make a decision. This is roughly equivalent to search engines, which produce a set of candidate documents and require users to look through the candidates.

These attempts at matching are often based on naive string matching, often considering string distance metrics for name-matching. This method is "most useful for matching problems with little prior knowledge, or ill-structured data," or when the number of available features is thin [6]. When matching user profiles, however, there is often an abundance of both explicit and implicit information, allowing for more sophisticated solutions. More complex but generic entity resolution solutions have been attempted, including Swoosh, a generic approach to entity resolution [1].

Because of the complexity of data in social networks, matching has become an interesting problem with studies that have attempted to understand collective entity resolution. There have been “relational clustering algorithm that uses both attribute and relational information for determining the underlying domain entities” [3]. These algorithms analyze entities as collective groups, such as the analysis of Stanford students within the larger Facebook social graph. Entity data, such as a user’s friend list, are termed grouped-entities and have been shown to only increase accuracy [5].

Google has been doing some particularly interesting work with their Knowledge Graph, which is a promising first attempt at understanding important entities (famous people, places, and things) and delivering an overview of each entity. A search for “Justin Bieber” on Google will now display a side panel containing Justin’s photo, age, height, networth, popular songs/albums, and biography. Searchers need not even click the search results. This is incredibly cool, but currently supports only famous entities.

Our System Design

We broke our system into three component modules, two of which are data crawlers, and one of which is responsible for performing the heavy lifting (resolving entities). The two crawlers are responsible for intelligently collecting data from the input social networks. Additional crawler implementations can be added for other social networks without changing the overall system architecture. The third (core) module is a series of map-reduce jobs that utilizes a distributed hashing mechanism to identify candidate pairs, and a profile comparator function to find the best match among the candidates.

University Data Crawler
The top universities (Stanford, Berkeley, MIT, etc) provide programmatic access to their public LDAP servers, which list directory information for students, staff, and faculty [7]. These data sets are relatively small (on the order of 50,000-150,000 records), so we can import the entire directory into our database. LDAP usually supports a search-by-uid, which permits wildcard characters, but is often limited to ~500 results per query. To get around this restriction, we used a recursive tree walk, where we walked all possible uid prefixes, expanding child prefixes if the number of results returned was equal to MAX_RESULTS. This exhaustive crawl allowed us to fetch all directory information.

Facebook Data Crawler
Fetching the required Facebook profiles was difficult for several reasons:

1. Facebook has over 400,000,000 registered users, and stores over 20 petabytes of data. Crawling this much data would be impractical for obvious reasons, so we had to be careful about how we crawled the data.
2. Facebook utilizes active countermeasures to prevent crawling. This meant (among other things) we had to rate limit ourselves (1 query / second / ip, capped at ~15k requests / day / ip). Despite the rate limits, Facebook did a reasonably good job of detecting and killing our crawlers, forcing us to play their game of cat and mouse.
3. Facebook’s servers kept crashing. Various errors were surprisingly prevalent and nondeterministic, making it difficult to ensure that all error conditions were being properly handled.
4. Facebook’s data was surprisingly inconsistent. Some users are identified by their username, others by their Facebook Id. Sometimes you can convert between them, but

not always. Sometimes usernames might become invalid (can the user change them?). Handling all these variations and edge conditions made it hard to work with this data.

In the end, we wrote a series of crawler daemons, which would acquire job assignments from a master server, and upload the results to Amazon S3. We also wrote a manager, that was responsible for SSHing into the worker machines, launching the daemons, and collecting the logs for analysis.

Because we can't crawl all facebook users, we use a priority queue of users that we think are likely to appear in the LDAP data. This queue is generated by running the previous iteration's data through the core module (responsible for resolving entities) to find matches, and finding userids of uncrawled friends who commonly appear as friends of the matched entities. The process is iteratively repeated to keep the fringe populated. This works because we assume that the network of users within a university's ldap directory is likely to be a dense graph on Facebook. This may result in us missing some new/incoming students, who have not yet built up their social graph. Different crawlers may want to utilize a different approach.

Core Module (Entity Resolver)

The entity resolution module reads over all the input profiles (generally, from two distinct crawlers) and hashes the profiles to produce candidate pairs. The candidate pairs are subsequently scored, and the highest scoring match is emitted.

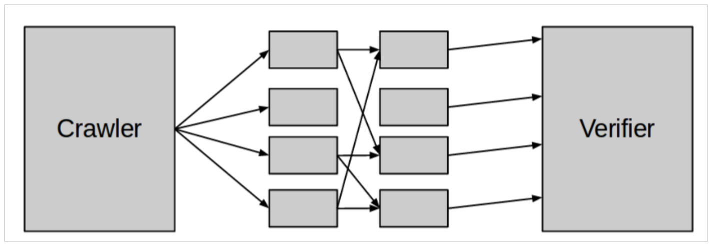

Figure 1: Overview of the data flow from crawlers through the core module and to the verifier.

The first map-reduce job is responsible for pre-processing the data and inferring information for potentially missing attributes. Under assumptions of homophily, we can generally guess an individual's major, home town, etc. For instance, a particular person may not have mentioned that they were a Computer Science major originally from the city of Saratoga, but if all their friends are either Computer Science majors or from Saratoga, both these fields can be assumed. Our comparator function is written in such a way that incorrect guesses don't penalize us too badly, but a good guess will be strongly represented in the score.

The second map-reduce job in the chain reads over all the crawled profiles, and hashes them into buckets. Keep in mind that the crawled data types are different (for instance, we might be mapping over both ldap data and facebook data, which contain different fields & attributes). The output of this first map-reduce job will be an order of magnitude larger than the initial input, because each profile will be hashed to several candidate buckets. Splitting an individual's name on spaces and dashes is a useful way to generate hash keys. Expanding nicknames is also important (eg. 'James' => 'Jim'). Hash keys should be distinctive enough to keep the buckets reasonably-sized; if buckets are too big, we recommend generating composite keys.

The third map-reduce job maps over each hash bucket, performing a nested loop join and emitting results where the two profiles are of different types. This allows us to ensure that all candidates for a particular profile get sent to the same reducer, which allows the reducer to pick the best matching profile (of the other profile type).

The third map-reduce job utilizes a complex profile comparator function to score each of the candidate pairs. The comparison function will give points for matches, partial points for partial matches, and deduct tiny fractions of a point for incorrect matches. For instance, if the comparator were examining names, it might give a full point for an exact match, half a point if one profile's name is a substring of the other profile's name, a tenth of a point if the first initials match (helps catch nicknames, which generally start with the same first letter), and subtract 1/100th of a point for no match within the name fragment. The comparator considers other fields, like home town/address, phone, username, name, major, graduation year, etc.

There were several other simple map-reduce jobs, which we used to massage the data, but which were ignored for clarity. For instance, privacy settings on facebook data could make relationships look asymmetric, but Facebook relationships are always symmetric. Filling in this missing data allows us to achieve better accuracy.

Experiments

To examine the accuracy of our data, we crawled three elite universities (Stanford, Berkeley, and MIT), predicted the profile matches, and compared results against a ground truth.

Data

Using the crawls as described in the previous section, we were able to aggregate up to fifty thousand Stanford student's data through the university's public LDAP directory. We were also able to obtain one hundred thousand Berkeley and a similar number of MIT students. The features collected per user would range based on the school's availability. For instance, Stanford & MIT gave fairly detailed features on work, research, and academic affiliation; street address; names and nicknames; and organizations. Berkeley, however, had much stricter privacy settings, generally only revealing a user's full display name and email address for certain individuals.

With the Facebook crawlers, we fetched all Facebook data publicly available. That includes all Facebook data that the individual users would publicly allow the users to see. Features extracted include full display name, university, graduation year, major, work experience, permanent address, email address, affiliated websites, birthday, network, and friend lists (if permitted).

The friend list proved to be an especially powerful feature when homophily is applied. By aggregating the data of friend lists, we can infer features about the individual user, even if privacy setting restrict us from being able to see the actual results.

Facebook data was stored in S3 as unparsed web documents. By the end of our experiments, we had successfully crawled about 200,000+ Facebook users crawled.

Experimental results

To evaluate our experimental results, we defined two key metrics:

- Coverage - This is the probability that the correct pairwise match has been crawled, given that the individual's profile does exist on both social networks. A naive crawler could always achieve 100% coverage by fetching every profile in the network, but this is not practical for large social networks. More sophisticated crawlers should be able to achieve a high percentage coverage, even when crawling a relatively small percentage of the full social network, by targeting profiles that are likely to appear in both data sets.
- Accuracy - This is the probability that we correctly matched two profiles, given that both profiles have been crawled. A good profile comparator function can achieve a high

accuracy even when the coverage is low.

Achieving high coverage and high accuracy is both necessary and sufficient for this system to be useful.

Calculating accuracy (and to a lesser extent, calculating coverage) requires an answer key. Our system uses email addresses from the university's LDAP directories, and feeds them into Facebook's find-a-friend-by-email-address search mechanism, to generate a validation set. Only about 10% of the students from our LDAP crawls could be found using the find-a-friend feature. We make the simplifying assumption that this small validation set is a statistically random subset, though this is probably an invalid assumption[3]. To calculate coverage, we calculate the percentage of profiles from the validation set that also occur in our Facebook crawl. To calculate accuracy, we count the number of correct pairings we discovered and divide by the size of the validation set. Because we make exactly one guess per LDAP-profile, the number of incorrect mappings is the inverse of the number of correct mappings, the accuracy metric already takes incorrect guesses into account.

Our final run achieved 89.5% coverage and 92% accuracy.

Future Work

- Incorporating profile information from other social networks, like GooglePlus, LinkedIn, and Quora.
- Automate the process. Currently, data is processed by manually executing chains of map-reduce jobs. Due to the iterative nature of the system, this process should be streamlined.
- When comparing across social networks that contain relationship information, consider a friend's identity as a feature, allowing the two social graphs to be mapped even when few other features match.

Conclusion

We describe an implementation of a complete entity resolver that matches Facebook user data to university LDAP data of the same person. In addition to providing our hypothetical "best matches," we also provide a technique to verify our results. This implementation includes descriptions of both crawling data and matching data to the best candidates using comparator techniques. In our implementation, we learned that both crawling and resolving were largely intertwined. Good matches would lead to a better network to crawl while a dense social network gave a candidate pool that would likely contain the correct answer. Our fully automated solution to social network entity resolution yields accurate results that rivals the capability of the decisions made by a human being.

Acknowledgements

We would like to thank our research advisors for their invaluable feedback & support. In

[3] Users who fill in the email address information used to generate the validation set may be less concerned with privacy and therefore more likely to reveal other information which could assist our profile comparator in making an accurate match. This selection bias could cause our reported accuracy to be higher than the true accuracy.

particular, we would like to thank Jeffrey Ullman (our project advisor) for his help in designing algorithms for effectively matching entities, and Jure Leskovec (CS246 professor) for teaching us many of the algorithms, techniques, and data structures used in this project. We would also like to thank Andreas Weigend & Anand Rajaraman for their early feedback & contributions.

References

[1] Benjelloun O, Garcia-Molina H, Menestrina D, Su Q, Whang S, Widom J. Swoosh: a generic approach to entity resolution. The VLDB Journal, 2009, 18: 255–276

[2] Bhattacharya, I. and Getoor, L. “Entity resolution in graphs.” Technical Report 4758, Computer Science Department, University of Maryland, 2005.

[3] Bhattacharya, I. and Getoor, L. “Collective entity resolution in relational data”, ACM Transactions on Knowledge Discovery from Data (TKDD), v.1 n.1, p.5-es, March 2007.

[4] Bilgic, M.; Licamele, L.; Getoor, L.; Shneiderman, B.; , "D-Dupe: An Interactive Tool for Entity Resolution in Social Networks," Visual Analytics Science And Technology, 2006 IEEE Symposium On , vol., no., pp.43-50, Oct. 31 2006-Nov. 2 2006
doi: 10.1109/VAST.2006.261429

[5] Byung-Won On; Elmacioglu, E.; Lee, D.; Jaewoo Kang; Jian Pei; , "Improving Grouped-Entity Resolution Using Quasi-Cliques," Data Mining, 2006. ICDM '06. Sixth International Conference on , vol., no., pp.1008-1015, 18-22 Dec. 2006
doi: 10.1109/ICDM.2006.85

[6] Cohen, W. W., Ravikumar, P. and Fienberg, S. 2003. “A Comparison of String Distance Metrics for Name-Matching Tasks.” IIWeb 2003: 73--78.

[7] Koutsonikola, V.; Vakali, A.; , "LDAP: framework, practices, and trends," Internet Computing, IEEE , vol.8, no.5, pp. 66- 72, Sept.-Oct. 2004
doi: 10.1109/MIC.2004.44