PRIVACY-PRESERVING SOCIAL NETWORK ANALYSIS

by

Raymond D. Heatherly

APPROVED BY SUPERVISORY COMMITTEE:

Dr. Murat Kantarcioglu, Chair

Dr. Jorge Cobb

Dr. Latifur Khan

Dr. Bhavani Thuraisingham

Dedicated to my loving wife, Delilah.

PRIVACY-PRESERVING SOCIAL NETWORK ANALYSIS

by

RAYMOND D. HEATHERLY, B.S., M.B.A., M.S.

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

August 2011

PREFACE

This dissertation was produced in accordance with guidelines which permit the inclusion as part of the dissertation the text of an original paper or papers submitted for publication. The dissertation must still conform to all other requirements explained in the "Guide for the Preparation of Master's Theses and Doctoral Dissertations at The University of Texas at Dallas." It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this dissertation to include as chapters authentic copies of papers already published[1], provided these meet type size, margin, and legibility requirements. In such cases, connecting texts which provide logical bridges between different manuscripts are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the student's contribution to the work and acknowledging the contribution of the other author(s). The signature of the Supervising Committee which precedes all other material in the dissertation attest to the accuracy of this statement.

[1] Partial or full content of chapters has been published in [9, 11, 29–32]

ACKNOWLEDGMENTS

First, to Murat Kantarcioglu, for his mentorship during my years at UTD. Thank you for your criticism – always constructive and frequently blunt. You were an excellent advisor who always let each of us know what was expected of us, but never hesitated to give freely of your time when needed.

An additional thank you to the members of my committee, Bhavanni Thuraisingham, Latifur Khan, and Jorge Cobb. Your comments are all appreciated, and your forgiveness for everything being last-minute is admired.

A further thanks for Jorge Cobb, for allowing me to work on his NSF GK-12 program. It was an incredible experience; frequently frustrating, but a positive memory of my time at UTD.

To my mother and father, who may not have always agreed with everything I did, but supported it; who were not afraid to let me fail, and help me get back up; and reminded me not to miss the forest for all the trees. Thank you for always putting education first.

To my sister, Renee Heatherly Rogge, who provided a role model that I have looked up to my entire life. Thank you for being yourself, and most of all for driving me to the library multiple times a week. You fostered in me a love of words and knowledge that would not let me be satisfied before finishing this work.

And finally, to my wife, Delilah. Words alone cannot express how thankful I am for your love and patience during this entire process. I know that without your support and encouragement, not a word would have been typed or an experiment run. This dissertation is as much your accomplishment as it is mine.

August, 2011

PRIVACY-PRESERVING SOCIAL NETWORK ANALYSIS

Publication No. _____

Raymond D. Heatherly, Ph.D.
The University of Texas at Dallas, 2011

Supervising Professor: Dr. Murat Kantarcioglu

Social networks have become ubiquitous in modern life. Whether a user is "tweeting" about events in their day, posting photos to Facebook, or using LinkedIn to find a job, even the average computer is aware of social networks. Their utility in such disparate domains as counter-terrorism and marketing has created a need for increasingly accurate classification techniques to counter attempts to hide useful information and connections. However, users of legitimate social networks have reasonable concerns about the use of their private data.

To this end, we provide a three-tiered approach to social network analysis: first, extend classification to make use of hidden data and patterns within a network. We show that these extensions provide benefits to classification tasks on a real-world social network. Second, provide a method for extensible access control of social networks so that users may control access to their data through normal use of said network. We show that the framework we propose is scalable and granular enough to provide fine-grained control over social network data on the scale of Facebook. Finally, provide a method for useful privacy-preserving methods of releasing social network data to interested third-parties. We show that these methods provide a measure of privacy control to users of a social network, while providing a guarantee of utility to interested third parties.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Social networks as an entity unto themselves are a fairly modern concept that has taken advantage of benefits provided by the Internet to become one of the most popular recent phenomena.

In fact, it may not be a stretch to say that the defining technology of the current decade – if not century – is the social network. Not only has Google launched its own (beta) version of a social network in Google+, but mobile phones natively send data to the existing social networks, desktop applications come pre-installed on new computers. Even the social network leaders are integrating with each other – Twitter will reproduce your tweets to your Facebook status feed or your LiveJournal blog.

However, the utility of social networks has extended far beyond the average internet user's activities on Facebook. Researchers have created social networks from such unique data sets as Enron's electronic mails and used them to analyse the fraud within that company.

It is exactly this ability to use social network analysis – even on data sets that are not conventional social networks – to determine hidden information – that makes them so valuable – and so dangerous.

If we separate the privacy concerns from the potential gain of information, we see that due to the unique nature of these networks, one is able to obtain extremly useful information. For example, researchers from the University of Washington and Columbia University mapped the sexual activities of a midwestern high school and showed the potential of such data sets for epidemiological and sociological research.

Further, researchers at the Massachusetts Institute of Technology showed that based upon a knowledge of the local area, given access to a subset of data in the Facebook network, they were able to accurately determine sexual orientation of users who had initially not provided this data to the network.

Due to this, it is increasingly important to determine methods by which we can efficiently obtain hidden information from these networks. Citing the specific example of counter-terrorism, the intelligence community has devoted a considerable amount of time and effort determining the composition of terrorist cells. However, following the attacks of September 11, 2001, these cells have also spent a considerable amount of time and effort attempting to ensure their anonymity.

To this end, we offer three different solutions to improving the accuracy of classifiers built on these data sets. The first is a classification improvement based upon the growing nature of link differentiation within social networks. For example, if one considers a Facebook user's edge set, one may find some combination of friends, co-workers, high school acquaintances, college classmates, and family members. Each of these types of links has a different bearing on what a particular user may believe or take part in. Our system of classification adds consideration of these various link types into probability classification.

Secondly, we offer a method of classification which considers that there are regional influences within graphs. Not every individual in a graph has the same influence on the entirety of the graph (or a smaller subgraph therein); in high schools there is the "popular clique" or in technology there are "early adopters." However, regardless of the domain, it is apparent that there are some people who have a higher regional influence.

Finally, we offer a partitioning-based method of classification which separates the data set into multiple groups. Each of these groups is modeled individually and we perform classification therein.

The previous examples illustrate the conflicting nature of research into social network analysis. While there are certain usages which are obviously important – epidemiology, counter-terrorism, fraut – there are other uses which serve no "greater good," yet violate an individual's privacy.

The former offers compelling evidence that analytical techniques for social networks should be improved so that we gain the maximum benefit from their use. The latter indicates that providers of online social networks should provide some mechanisms to protect the information of its' users.

However, privacy in social networks is distinctly divided into two areas: privacy through standard usage of the network and privacy with released data. To solve the former, we provide a method of social network access control that addresses the unique requirements of online social

networks. For the latter, we provide two methods of sanitizing social network data for release to a semi-trusted third party.

CHAPTER 2
RELATED WORK

In this section, we will give an overview of the current research in the three research domains identified in this work: social network data mining, access controls, and privacy within social networks.

2.1 Social Network Data Mining

Other papers have tried to infer private information inside social networks. In [28], He et al. consider ways to infer private information via friendship links by creating a Bayesian Network from the links inside a social network. While they crawl a real social network, Livejournal, they use hypothetical attributes to analyze their learning algorithm. Also, compared to [28], we provide techniques that can help with choosing the most effective details or links that need to be removed for protecting privacy. Finally, we explore the effect of collective inference techniques in possible inference attacks.

The Facebook platform's data has been considered in some other research as well. In [36], authors crawl Facebook's data and analyze usage trends among Facebook users, employing both profile postings and survey information. However, their paper focuses mostly on faults inside the Facebook platform. They do not discuss attempting to learn unrevealed details of Facebook users, and do no analysis of the details of Facebook users. Their crawl consisted of around 70,000 Facebook accounts.

The area of link based classification is well studied. In [50], authors compare various methods of link based classification including loopy belief propagation, mean field relaxation labeling, and iterative classification. However, their comparisons do not consider ways to prevent link based classification. Belief propagation as a means of classification is presented in [59]. In [52], authors present an alternative classification method where they build on Markov Networks. However, none of these papers consider ways to combat their classification methods.

In [61], Zheleva and Getoor attempt to predict the private attributes of users in four real-world datasets: Facebook, Flickr, Dogster, and BibSonomy. Their focus is on how specific types of data, namely that of declared and inferred group membership, may be used as a way to boost local and relational classification accuracy. Their defined method of group-based (as opposed to details-based or link-based) classification is an inherent part of our details-based classification, as we treat the group membership data as just another detail, as we do favorite books or movies. In fact, Zheleva and Getoor work provides a substantial motivation for the need of the solution proposed in our work.

In [45], the authors use iterative classication techniques to improve accuracy of classication methods of social networks. This approach is mirrored in [42], and the authors extend the work with a full comparison of several relational classication methods and several collective inference techniques. However, while [42] begins to acknowledge the use of link types, in their experiments using the IMDb dataset, they generate separate relational graphs based on this link type. Unfortunately, they consider only one link type in each experiment. Similarly, [26] considers the problem of information revelation on the Facebook social network, giving a focus to the users ability to decrease the amount of information they reveal, but highlights the relatively small number of users who actually do so.

In [57], the authors use hypothetical attributes from the real-world dataset of LiveJournal to construct a Bayesian Network which they use to analyze their algorithm. However, they do not consider any aspect of link types. [60] discusses methods of link re-identication. However, in our data, the links are not explicitly dened and are instead inferred directly from the stated facts. Anonymization of the data that allows us to generate the relationships would make the data meaningless.

In [35], Jensen et al analyze the importance of collective inference methods on relational networks. Senator gives a general review of the past research in link mining and link analysis, and goes on to issue a call for a unied framework to support a variety of data representations and for the analysis tools in [53]. Getoor and Diehl, in [25] offer a survey of the various tasks performed on both hetero- and homogeneous social networks.

In [51], Sweeney describes several problems and poses a potential solution to naively generating social network from data that did not specically come from a social network. The author

indicates that the use of a form of privacy-enhanced linking in algorithm development can be used to guarantee individual privacy while advancing the cause of technological research.

In [14], Chau and Faloutsos implement a method of detecting fraudulent transactions in the online auction site E-Bay. By modeling transaction histories of buyers and sellers, they are able to use characteristic features and design a decision tree that is able to detect fraudulent activities with an 82accuracy.

In [28] and [57], the authors use relational Bayesian classication techniques in an attempt to determine private information from a social network. In [1], authors conduct experiments and build a system to classify both relational and attribute-based models. They then show the accuracy of their system on a set of constructed data.

In the general area of social network data mining, [8] attempt to discover hidden communities in social network data analysis. In [6], the authors discuss a method of mining a series of text e-mails to create a social network representation and perform an analysis of the data therein. In [41], the authors compare multiple researchers' work in link-based classification of network data. This work, and those cited therein, provide the rationale of our choice in relational classifiers. Similarly, [31] uses Link Types in conjunction with the above-listed methods in order to improve the efficiency of standard classifiers.

In [25] Diehl and Getoor present a survey of various approaches in discovering hidden links within social networks. The major difference between those approaches and ours is that traditional link mining approaches attempt to discover links which exist in real life and do not yet exist in the current state of the network. In our approach, though we add in links for our relational classifier, we do not insinuate that these necessarily represent links that may exist in real life. The links we create are used only to connect highly important nodes for the sake of improved information flow.

While there has been some recent work in dyadic prediction of social networks, we believe that our work contributes in parallel to this work. The primary focus is on using the dyadism in order to predict node labels for social graphs [43], while here we attempt to uncover hidden links which could then be used for an arbitrary choice of classifiers to increase prediction accuracy.

In [21], the authors conduct distributed data mining in a peer-to-peer network to find usage data about the network itself. However, the situation mentioned in this paper is different from our

scenario; in their research, the assumption is that the data is distributed fully across the system with each site having only minuscule knowledge of the entirety. In our system, the overall social network's data is divided among several data warehouses where we perform classification.

In the area of large-scale, efficient data mining, [47] suggests a Map-Reduce framework using Hadoop to perform data mining on the petabyte-scale. Similarly, [18] shows that by using Google's Map-Reduce architecture, it is possible to increase computational efficiency on multi-core systems by using traditional single-core learning methods. However, the authors consider a generic data warehouse, and do not consider those unique elements of social networks – such as the link structure – and thus their solutions may not be as effective for processing social network data. To improve on this, we focus specifically on the unique nature of the data in a social network and work to provide a method to divide this data for use at multiple sites or on multiple cores.

We also examine potential graph measures that have been studied in social networks to devise methods by which to partition our data sets. In [34], Jackson describes the graph metrics that we use in our experiments to divide the data sets. However, he uses these metrics as a way of simply measuring aspects of the social network, and uses the figures from this to infer information about the network as a whole. We use these to measure specific information about an individual node in the graph and use that to later partition the graph into several subsets.

In [42], Macskassy and Provost conduct a thorough investigation of the use of collective inference techniques in classifying social network data. We use the results of their experiments in choosing the relational and collective inference techniques we use in the experiments for this work. Further, in [46], Neville and Jensen propose a series of techniques for mining data specifically from social networks. However, their approach does not consider the need to distribute the data contained in the social networks for ease of calculation.

2.2 Access Control of Social Networks

Past research on OSN security has mainly focused on privacy-preserving techniques to allow statistical analysis on social network data without compromising OSN members' privacy (see [10] for a survey on this topic). In contrast, access control for OSNs is a relatively new research area. As far as we are aware, the only other proposals of an access control mechanism for online social

networks are [38], [2] and [12]. The D-FOAF system [38] is primarily a Friend of a Friend (FOAF) ontology-based distributed identity management system for social networks, where access rights and trust delegation management are provided as additional services. In D-FOAF, relationships are associated with a trust level, which denotes the level of *friendship* existing between the users participating in a given relationship. Although [38] discusses only generic relationships, corresponding to the ones modeled by the foaf:knows RDF property in the FOAF vocabulary [7], another D-FOAF-related paper [15] considers also the case of multiple relationship types. As far as access rights are concerned, they denote authorized users in terms of the minimum trust level and maximum length of the paths connecting the requester to the resource owner. In [2], authors adopt a multi-level security approach, where trust is the only parameter used to determine the security level of both users and resources. In [12], a semi-decentralized discretionary access control model and a related enforcement mechanism for controlled sharing of information in OSNs is presented. The model allows the specification of access rules for online resources, where authorized users are denoted in terms of the relationship type, depth, and trust level existing between nodes in the network.

Compared to existing approaches, we use semantic web technologies to represent much richer forms of relationships among users, resources and actions. For example, we are able to represent access control rules that leverage relationship hierarchies and by using OWL reasoning tools, we can infer a “close friend” is also a “friend” and anything that is accessible by friend could be also accessible by a “close friend”. In addition, our proposed solution could be easily adapted for very different online social networks by modifying the underlying SNKB. A further discussion on the differences between the proposed framework and the access control mechanism in [12] is provided in Section 7.4.

Semantic web technologies have been recently used for developing various policy and access control languages for domains different from OSNs. For example, in [54], authors compare various policy languages for distributed agent based systems that define authorization and obligation policies. In [24], OWL is used to express role-based access control policies. In [58], authors propose a semantic access control model that separates the authorization and access control management responsibilities to provide solutions for distributed and dynamic systems with heterogeneous security requirements. None of these previous work deals with the access control issues related to

online social networks. Among the existing work, [23] is the most similar to our proposal. Compared to [23], we provide a much richer OWL ontology for modeling various aspects of online social networks. In addition, we propose authorization, admin and filtering policies that depend on trust relationships among various users.

2.3 Privacy in Social Networks

In [3], authors consider an attack against an anonymized network. In their model, the network consists of only nodes and edges. Detail values are not included. The goal of the attacker is simply to identify people. Further, their problem is very different than the one considered in this paper because they ignore details and do not consider the effect of the existence of details on privacy.

In [27] and [40], authors consider several ways of anonymizing social networks. However, our work focuses on inferring details from nodes in the network, not individually identifying individuals.

In [27], authors consider perturbing network data in order to preserve privacy. While their method considers graph structure, it ignores any extra details or traits that a node inside the social network may possess.

In [40], authors consider the problem of anonymizing a social graph through making the graph k-anonymous. In their approach, they apply k-anonymity toward the identification of nodes through measuring the degree of a node. However, in their research, they assume that the structure and identity of the nodes is the sensitive information, and neither nodes nor links contain any additional information which can be used to distinguish a node. We consider that the graph has considerably more information and that attributes of nodes, not their identity, is the end goal of an attacker.

In [60], the authors propose several methods of social graph anonymization, focusing mainly on the idea that by anonymizing both the nodes in the group and the link structure, that one thereby anonymizes the graph as a whole. However, their methods all focus on anonymity in the structure itself. For example, through the use of k-anonymity or t-closeness, depending on the quasi-identifiers which are chosen, much of the uniqueness in the data may be lost. Through our

method of anonymity preservation, we maintain the full uniqueness in each node, which allows more information in the data post-release.

In [26], Gross and Acquisti examine specific usage instances at Carnegie Mellon. They also note potential attacks, such as node re-identification or stalking, that easily accessible data on Facebook could assist with. They further note that while privacy controls may exist on the user's end of the social networking site, many individuals do not take advantage of this tool. This finding coincides very well with the amount of data that we were able to crawl using a very simple crawler on a Facebook network. We extend on their work by experimentally examining the accuracy of some types of the Demographic Re-identification that they propose before and after sanitization.

In [43], authors use dyadic data methods to predict class labels. We show later that while we do not examine the effects of this type of analysis, the choice of technique is arbitrary for anonymization and utility.

CHAPTER 3
BACKGROUND INFORMATION

We begin by describing background information that will be necessary to the understanding of the latter works described herein. We start with a formal description of a social network and related concepts. We then dsicuss classification in regards to social networks, and we conclude with a description of the data sets used throughout this work.

3.1 Social Network Description

While most people have a vague notion of social networks, often in terms of one of the many online providers of such services. Formally, we define a social network as

Definition 1 *A scoial netowrk, $\mathcal{G}$, is a set $\mathcal{G}$= $\{\mathcal{V}, \mathcal{E}, \mathcal{D}\}$ where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is a set describing the vertices of the graph. $\mathcal{E} = \{(v_i, v_j), \ldots, (v_y, v_z)\}$ is a set describing the links within the graph. $\mathcal{D}$ is a set of additional "details" within the social network.*

In the context of the Facebook social network, this means that $\mathcal{V}$ are all users of Facebook, $\mathcal{E}$ are all friendship links, and $\mathcal{D}$ is all additional data that is stores about the user. In Chapter 4 we propose methods which utilize additional information within $\mathcal{E}$; however, for a large portion of this work, we focus upon various uses of the data contained in $\mathcal{D}$. For this reason, we now highlight specifics regarding $\mathcal{D}$.

Definition 2 *A detail type is a string defined over an alphabet Σ which corresponds to one of the domain-specific attribute categories. The set of all detail types is referred to by $\mathcal{H}$.*

Definition 3 *A detail value is a string defined over an alphabet Σ which corresponds to a possible response for a given detail type.*

Definition 4 *A detail is a single (detail type, detail value) pair that occurs within the data. The set of all unique details is referred to as $\mathcal{J}$.*

Name of value	Variable
Node numbered i in the graph	n_i
All details of node n_i	D_i
Detail j of node n_i	D_i^j
Friendship link between person n_i and n_k	$F_{i,k}$
The weight of a friend link from n_i to n_j	$W_{i,j}$

Table 3.1. Common Notations Used in the Paper

Definition 5 *The detail set, $\mathcal{D}$, is a set composed of m (v_i, detail) pairs, where $v_i \in \mathcal{V}$.*

To continue illustrating these concepts through Facebook, then a detail type is one of the categories a user may enter data under, such as "Favorite Books", "hometown", or "Education."

Possible detail values for these types are *Harry Potter and the Deathly Hallows, Dallas, Texas,* and *Millsaps College*, respectively.

We further define a set of private details $\mathcal{I}$, where any detail is private if for any $h_m \in \mathcal{H}$, $h_m \in \mathcal{I}$. Consider the following illustrative example:

$$\mathcal{I} = (\text{political affiliation, religion}) \tag{3.1}$$

$$n_1 = (\text{Jane Doe}) \tag{3.2}$$

$$n_2 = (\text{John Smith}) \tag{3.3}$$

$$D_2 = \{\text{John Smith's Details }\} \tag{3.4}$$

$$D_2^4 = (activities, fishing) \tag{3.5}$$

$$F_{1,2} \in E \tag{3.6}$$

$$F_{2,1} \in E \tag{3.7}$$

That is, we define two *detail types* to be private, a person's political affiliation and their religion (3.1). Then, say we have two people, named Jane Doe and John Smith respectively (3.2, 3.3). John Smith has specified that one of the activities he enjoys is fishing (3.5). Also, John and Jane are friends. Note that because our graph is undirected, examples 3.6 and 3.7 are interchangeable, and only one is actually recorded.

For further clarity, in Figure 3.1 we have a reference for many frequently-used notations found in the remainder of this paper.

Obviously, the detail types of $\mathcal{I}$ are varied based on an individual's choice. Generally, however, we consider a user's $\mathcal{I}$ to be any details that they do not specify. For experimentation, however, we choose to use only $\mathcal{I} = \{\text{political affiliation}\}$. We use this detail type as our C in all classification methods. Further, we consider only C_{lib} and C_{cons} as possible class values – that is, "Liberal" and "Conservative."

3.2 Classification

At any given time, a data representation of a social network could be incomplete. There are many potential reasons for this issue, including technical limitations of the network or conscious choices of the users therein. As an example of this, Facebook does not have an explicit "sexual orientation" field; however, it does have a "political affiliation" field which many users explicitly leave empty.

There are times when this incompleteness interferes with operations within the network. Consider, for instance, if Gearbox Studios pays for their newest game to be marketed on Facebook. There may be 2% of the entire graph that lists *video games* as a "Favorite Activity." If Facebook only markets to that segment, then there may be a considerable population who miss out on relevant advertisements. In order to determine other users who may be interested in video games – a fact which could be considered *hidden* – then Facebook may perform *classification* to find additional users to target with marketing.

Definition 6 *Classification is a mechanism where one attempts to determine the likelihood of a given event when the information directly related to that event is unknown or partially known.*

$\Lambda_c^{\mathcal{J}_y}(\mathcal{G})$ is the accuracy of a given classifier, c, on detail $\mathcal{J}_y$ using the graph $\mathcal{G}$. This accuracy is given as the ratio of correct classifications to total classifications.

3.2.1 Naïve Bayes Classification

Within this work, the most common type of classifier that will be used is the Bayesian classifier. The details of this classification method are described below:

Definition 7 *Given a node n_i with m details and p potential classification labels, $C_1, \ldots, C_p$, the probability of being in a class is given by the equation*

$$\underset{1 \leq x \leq p}{\operatorname{argmax}}[P(C_x|D_{1,i}, \ldots, D_{m,i})]$$

where $\arg\max 1 \leq x \leq p$ *represents the possible class label which maximizes the previous equation. However, this is difficult to calculate, since $P(C_x)$ for any given value of x is unknown. By applying Bayes' Theorem, we have the equation*

$$\underset{1 \leq x \leq p}{\operatorname{argmax}} \left[\frac{P(C_x) \times P(D_{1,i}, \ldots, D_{m,i}|C_x)}{P(D_{1,i}, \ldots, D_{m,i})} \right].$$

Further, by assuming that all details are independent, we are left with the simplified equation

$$\underset{1 \leq x \leq p}{\operatorname{argmax}} \left[\frac{P(C_x) \times P(D_{1,i}|C_x) \times \ldots \times P(D_{m,i}|C_x)}{P(D_{1,i}, \ldots, D_{m,i})} \right].$$

Notice, however, that $P(D_{1,i}, \ldots D_{m,i})$ is equivalent for all values of C_x. Which means that we need only compare

$$\underset{1 \leq x \leq p}{\operatorname{argmax}}[P(C_x) \times P(D_{1,i}|C_x) \times \ldots \times P(D_{m,i}|C_x)] \tag{3.8}$$

to determine a new class label for n_i.

While there are multiple benefits to this method of classification, one of the most important for this work is its reliability in classification, as well as its speed. Further benefits, specifically related to our privacy mechanisms, will be discussed in more detail in Chapter 9.

3.2.2 Collective Inference

One problem with the traditional Bayesian classifier described previously is that they generally consider only the data local to a node – which has led to their being classified as *local* classifiers. Social Networks, however, also contain a rich collection of data for the associated links. In order to leverage this information, *Collective Inference* is a method of classifying social network data using a combination of node details – such as favorite books and movies – and connecting links in the social graph. Each of these classifiers consists of three components: Local classifier, relational classifier, and Collective Inference algorithm.

Local classifiers are a type of learning method that is applied in the initial step of collective inference. As mentioned previously, social networks can be in various states of "knowledge." In the situation where this involves the detail to be classified on, this can create situations where after even a number of iterations of a relational classifier, some nodes may not have a classification value. In order to prevent this, the first step of all collective inference techniques is to determine a *prior* for each node through the use of a local classifier.

Relational classifiers are a separate type of learning algorithm that looks at the link structure of the graph, and uses the labels of nodes in the training set to develop a model which in turn is used to classify the nodes in the test set. Specifically, in [42], Macskassy and Provost examine four relational classifiers: Class-Distribution Relational Neighbor (cdRN), Weighted-Vote Relational Neighbor (wvRN), Network-only Bayes Classifier(nBC), and Network-only Link-based Classification (nLB).

A problem with relational classifiers is that while we may cleverly divide fully labeled test sets so that we ensure every node is connected to at least one other node in the training set, real-world data may not satisfy this strict requirement. If this requirement is not met, then relational classification will be unable to classify nodes which have no neighbors in the training set. Collective Inference attempts to make up for these deficiencies by using both local and relational classifiers in a precise manner to attempt to increase the classification accuracy of nodes in the network. By using a local classifier in the first iteration, collective inference ensures that every node will have an initial probabilistic classification. The algorithm then uses a relational classifier to re-classify nodes.

3.3 Data Gathering

We wrote a program to crawl the Facebook network to gather data for our experiments. Written in Java 1.6, the crawler loaded a profile, parsed the details out of the HTML, and stored the details inside a MySQL database. Then, the crawler loaded all friends of the current profile and stored the friends inside the database both as friendship links and as possible profiles to later crawl.

Because of the sheer size of Facebook's social network, the crawler was limited to only crawling profiles inside the Dallas/Forth Worth (DFW) network. This means that if two people share a

common friend that is outside the DFW network, this is not reflected inside the database. Also, some people have enabled privacy restrictions on their profile which prevented the crawler from seeing their profile details. [1] The total time for the crawl was seven days.

Because the data inside a Facebook profile is free form text, it is critical that the input be normalized. For example, favorite books of "Bible" and "The Bible" should be considered the same detail. Further, there are often spelling mistakes or variations on the same noun.

The normalization method we use is based upon a Porter Stemmer presented in [55]. To normalize a detail, it was broken into words and each word was stemmed with a Porter Stemmer then recombined. Two details that normalized to the same value were considered the same for the purposes of the learning algorithm.

Our total crawl resulted in over 167,000 profiles, almost 4.5 million profile details, and over 3 million friendship links. In the graph representation, we had one large central group of connected nodes that had a maximum path length of 16. Only 22 of the collected users were not inside this group.

In Table 3.2, we provide some general statistics of our Facebook dataset, including the diameter mentioned above. Common knowledge leads us to expect a small diameter in social networks [56]. To reconcile this fact with the empirical results of a 16 degree diameter in the graph, note that, although popular, not every person in society has a Facebook account and even those that do still do not have friendship links to every person they know. Additionally, given the limited scope of our crawl, it is possible that some connecting individuals may be outside the Dallas/Fort Worth area.

In Table 3.3, we show the original class likelihood for those details which will be used as experimental class values.

Figures 3.1 and 3.2 show how many people have the specified number of details or friendship links, respectively. For example, the point $(x = 4, y = 6100)$ for figure 3.2 means that 6,100 people have 4 friendship links. The point $(x = 4, y = 38979)$ for figure 3.1 means that 38,979 people have 4 listed details. It is important to note that both figures have a logarithmic Y scale. This shows that the vast majority of nodes in the graph have few details and friendship links.

[1] The default privacy setting for Facebook users is to have all profile information revealed to others inside their network.

Diameter of the largest component	16
Number of nodes in the graph	167,390
Number of friendship links in the graph	3,342,009
Total number of listed details in the graph	4,493,436
Total number of unique details in the graph	110,407
Number of components in the graph	18

Table 3.2. General information about the data

Probability of being Liberal	0.45
Probability of being Conservative	0.55
Probability of being Heterosexual	0.95
Probability of being Homosexual	0.05

Table 3.3. Odds of being Liberal or Conservative

Figure 3.3 shows the number of profiles where the most popular activities were listed. For example, the point $(x = 1, y = 2373)$ means that the most popular detail of the "Activities" type was listed inside 2,373 profiles while the point $(x = 4, 1482)$ means that the fourth most popular detail was listed 1,482 times.

Figure 3.3 is actually only part of the graph. The X axis can extend to the point 94,146 but was cut off at 6,000 for readability purposes. Combined with the fact that the Y axis has a a logarithmic Y scale, this shows that the few most popular details are represented strongly in the data set.

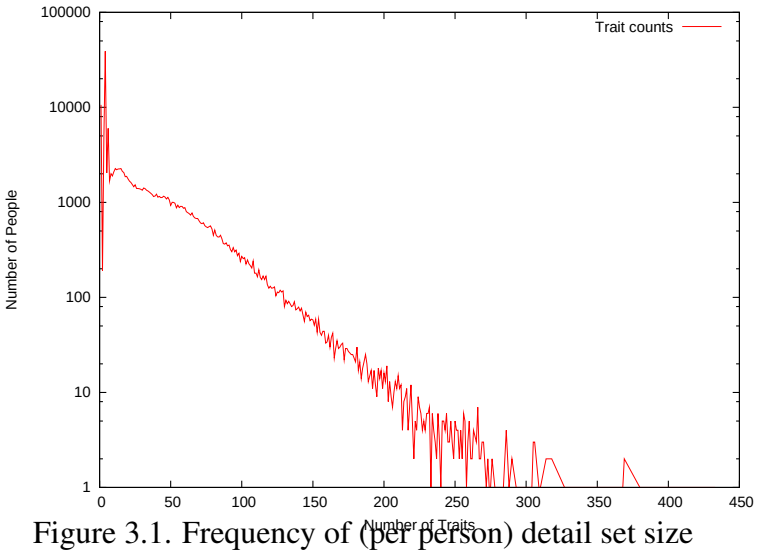


Figure 3.1. Frequency of (per person) detail set size

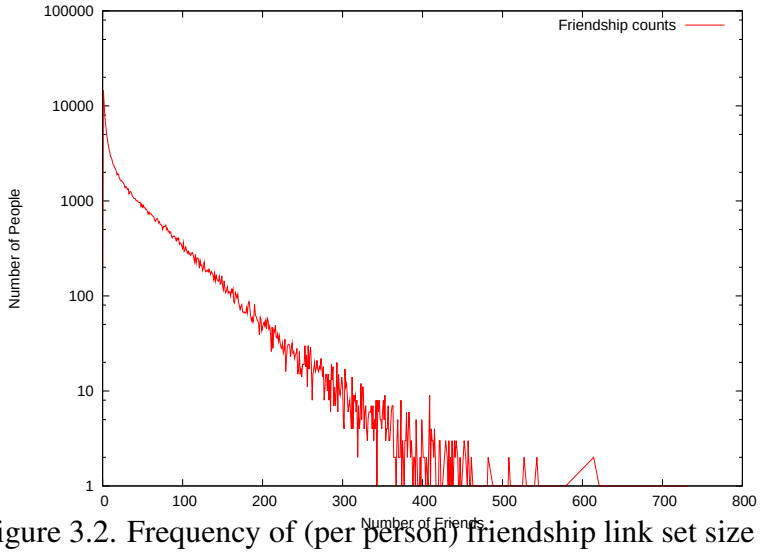


Figure 3.2. Frequency of (per person) friendship link set size

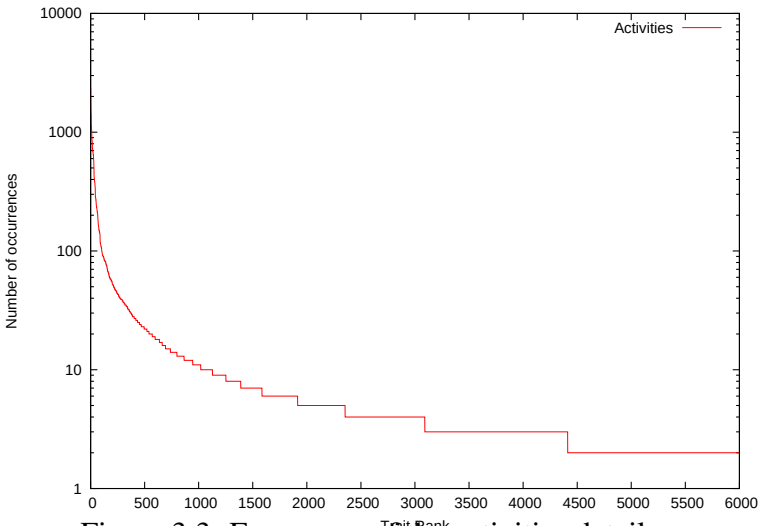


Figure 3.3. Frequency of the activities detail

CHAPTER 4

CLASSIFICATION OF SOCIAL NETWORKS INCORPORATING LINK TYPES

One feature that is frequently overlooked when classifying in social networks is the richness of data in the link set, $\mathcal{E}$. Consider, for instance, the variety of link types within only Facebook. An individual can have "regular" friends, "best" friends, siblings, cousins, co-workers, classmates from high school and college. Previous work has either ignored these link types or, worse, used them solely on the basis of whether to include or exclude them from consideration.

Here, we show a new classification method which takes advantage of this data. We further show that by intelligently weighting those link types, it is possible to gain even more benefit from the link types.

4.1 Introduction

Social Networks are platforms that allow people to publish details about themselves and to connect to other members of the network through links. Recently, the population of such online social networks has increased significantly. For instance, Facebook now has over 150 million users[1]. Facebook is only one example of a social network that is for general connectivity. Other instances of social networks are LinkedIn (professional), Last.fm (music), orkut (social), aNobii (books), and the list continues.

Each of these networks allow users to list details about themselves, and also allows them to not specify details about themselves. However, these hidden details can be important to the administrators of a social network. Most of these sites are free to the end user and are thus advertising-supported. If we assume that advertisers want to reach the people most likely to be interested in their products, then identifying those individuals becomes a priority for maintaining much-needed advertisers. However, by just using specically dened information, the site may be missing a large number of potentially interested users. Taking the specied knowledge from some

[1] http://www.facebook.com/press/info.php?statistics

users and using it to infer unspecied data may allow the site to extend its target audience for particular advertisements.

Of course, the implications of classication in social network data extend far beyond the simple case of targeted advertising. For instance, such ideas could be used for addressing classication problems in terrorist networks. By using the link structure and link types among nodes in a social network with known terrorist nodes, we can attempt to classify unknown nodes as terrorist or non-terrorist. In such a classication process, the type of the link shared among nodes could be really critical in determining the nal success of the classier. For example, assume that there exist two different individuals, Alice and Bob, that are linked to some known terrorist, Malory. Furthermore, assume that Alice works at the same place as Malory, but they are not friends. (i.e., Alice and Malory have a relationship. The type of relationship is colleague). In addition assume that Malory talks frequently on the phone with Bob and they are friends (i.e., Bob is linked to Malory and the link type is "friend"). Given such a social network, to our knowledge, all the existing classication methods just use the fact that individuals are linked and they do not use the information hidden in the link types. Clearly, in some cases, one link type (e.g., friendship) can be more important than other link types for increasing the accuracy of the classication process. Therefore, social network classication techniques that consider link types could be very useful for many classication tasks.

To address the above problem, we choose the network-only Naive Bayes classier as our starting point, since it is shown in [42] that relational Naive Bayes classication combined with collective inference techniques provide an efcient solution with acceptable accuracy in practice. We modify this relational Naive Bayes classier to incorporate the type of the link between nodes into the probability calculations. First, we devise a naive Bayes classication method in which all link types are equally important and we then rene this method by adding an additional granularity to the calculation method. To our knowledge, this is the rst paper in using link types for relational Naive Bayes classication.

Our paper is organized as follows. In section 4.2, we discuss our relational Naive Bayes methods that incorporate link types. In section 4.3, we show the effectiveness of our learning method on real world data set. Finally, we conclude with the brief discussion of the future work.

4.2 Learning Methods

Currently, classication in social networks is done in a variety of ways. The simplest of these is to use the data about nodes for which the label is known, and to use a simple classication mechanism, such as a naive Bayes Classier, to classify based on only those attributes that are local to the node being classied. Methods that classify in this manner are considered *local classiers*.

Another method is referred to as *relational classiers*. These classication algorithms model the social network data as a graph, where the set of nodes are a specic, homogeneous entity in the representative network. Edges are added based on specic constraints from the original data set. For instance, in Facebook data, the edges are added based on the existence of a friendship link between the nodes. In [42], the authors determine a link between two nodes in their IMDb dataset if the two nodes share the same production company. Once this graph structure is created, a classication algorithm is then applied that uses the labels of a nodes neighbors to probabilistically apply a label to that node. These algorithms use the theory of *homophily*, that is, that a node is more likely to associate with other nodes of the same type, to perform their calculations.

However, one of the problems with even relational classiers is that if the labels of a large portion of the network are unknown, then there may be nodes for which we cannot determine a classication. To truly represent homophily, the classied nodes should be used to re-assess the classications of its neighbors. This is the thought behind *collective inference*. These algorithms use a local classier to create a set of class priors for every node, to ensure that each node has an initial "guess" of a classication. Then, the algorithm uses a relational classier to generate a probability assignment for each node based on those class priors. This allows us to use a relational classier and ensure that each node will have a classication. The collective inference algorithm species a weighting scheme for each iteration. It also species either a number of iterations to run for or to wait for convergence in the classication.

Because we focus our efforts on improving the overall relational classication by increasing the data available for this class of classiers to use, we use a single local classier and vary the relational classiers. Each of these methods is based on the traditional naive Bayes classier.

4.2.1 Local Classification

As shown in [28], using a simple Bayes classifier is an effective method of discerning private information from a social network. Because of this finding and its relatively low computational requirements, we use a basic Naive Bayes classifier. We use the classifier to determine the probability that a particular node, x_i , is of a particular class, c_i , given the entire set of its traits, τ by the formula

$$Pr(x_i = c_i|\tau) = \frac{Pr(\tau|x_i = c_i)Pr(x_i = c_i)}{Pr(\tau)} = Pr(x_i = c_i) \times \prod_{t_i \epsilon \tau} \frac{Pr(t_i|x_i = c_i)}{Pr(t_i)} \tag{4.1}$$

4.2.2 Network-only Bayes Classification

This is the first, and most basic, of the three relational classifiers that we examine here. The general nBC assumes that all link types are the same, and that the probability of a particular nodes class is influenced by the class of its neighbors. We use the local classifier to assign an inferred prior of each node in the test set. Since the details of a particular node are factored in when we establish these priors, we do not duplicate this in the nBC calculations. We alter the probability calculations as follows:

$$Pr(x_i = c_i|\mathcal{N}) = \frac{Pr(\mathcal{N}|x_i = c_i)Pr(x_i = c_i)}{Pr(\mathcal{N})} = Pr(x_i = c_i) \times \prod_{n_i \in \mathcal{N}} \frac{Pr(n_i|x_i = c_i)}{Pr(n_i)} \tag{4.2}$$

This method is a basic classifier, yet Chakrabarti et al and Macskassy and Provost ([13], [42], respectively), both use nBC to effectively classify nodes on a variety of data sets. Because of their findings, we use this algorithm as the basis for ours.

4.2.3 Link Type nBC

The next relational classifier that we test is the first to include link types. We noticed that there were no classification algorithms to specify constraints about what type of link two individuals shared, in an important way, for the probability calculations. We believed that including these differences when determining probabilities could be an important and useful extension. For instance, consider Ron Howard. As a writer, director, producer, and actor, he is linked to over a hundred different television shows and movies. However, when classifying whether a movie would be a success, there must be a difference in how important his role in a production is to calculate his weight on the movies success.

We represent this by including the link type as an additional parameter to a Naive Bayes classification. Whereas originally, we define the probability of any specific node, xi to be in a particular class, ci , to be $Pr(x_i = c_i|\mathcal{N})$ that is, the probability of a node being in a class is dependent only upon its neighbors we now define the probability to be $Pr(x_1 = c_1|\mathcal{N}, \mathcal{L})$. That is, the probability of any particular node being in a class is determined by its neighbors, and the set of links that define those neighbors. So, we amend the traditional Naive Bayes calculation to be:

$$\begin{aligned} Pr(x_i = c_i|\mathcal{N}, \mathcal{L}) = \\ \frac{Pr(\mathcal{N}, \mathcal{L}|x_i = c_i)Pr(x_i = c_i)}{Pr(\mathcal{N}, \mathcal{L})} = \\ \prod_{n_i \in \mathcal{N}} l_i \in \mathcal{L} \frac{Pr(n_i, l_i|x_i = c_i)Pr(x_i = c_i)}{Pr(n_i, l_i)} \end{aligned} \tag{4.3}$$

4.2.4 Weighted Link Type rBC

The final relational classifier considers that certain link types are indicative of a stronger relationship. For instance, our IMDb data set contains data about all of the crew of a show. This includes the directors, producers, actors, costume design- ers, grippers, and special effects technicians. We theorized that not all of these jobs can be equally important. So, we alter our Link Type rBC to include weights. To this, we add the idea of variable weights to the calculation in Equation 4.3.

$$Pr(x_i = c_i|\mathcal{N}, \mathcal{L}) = \prod_{\substack{n_i \in \mathcal{N} \\ l_i \in \mathcal{L}}} \left[\frac{w_i}{W} \times \frac{Pr(n_i, l_i|x_i = c_i)Pr(x_i = c_i)}{Pr(n_i, l_i)} \right] \tag{4.4}$$

where w_i is the weight associated with link type l_i , and W is the sum of all weights for the network.

4.2.5 Relaxation Labeling

We choose to use Relaxation Labeling as described in [42], a method which retains the uncertainty of our classified labels. Relaxation Labeling is an iterative process, where at each step $i + 1$ the algorithm uses the probability estimates, not a single classified label, from step i to calculate probability estimates. Further, to account for the possibility that there may not be a convergence, there is a decay rate, called α set to 0.99 that discounts the weight of each subsequent iteration compared to the previous iterations.

We chose to use Relaxation Labeling because in the experiments conducted by Macskassy and Provost [42], Relaxation Labeling tended to be the best of the three collec- tive inference methods.

We decided to choose the best of the three methods as reported in [42] and use that to focus on our new relational classifiers.

4.3 Experiments

The data used in our experiments was gathered from the Internet Movie Database (IMDb). This decision was made based on the importance of not only having links between members in our social network representation, but on the ability to specifically define what specific relationship is indicated by them. Although, we considered using data from the Facebook online social network, we discovered that while Facebook allows users the ability to specify the relationship type linking friends, most relationships are not defined. Rather than add additional inference operations that could cast doubt on the validity of the relationship types, we chose to use a dataset where relationship types are given.

We implemented a crawler using Java 1.6 that crawled the site and parsed information from a selection of pages of all movies. The pages we selected were Full Cast and Crew, Main Details, Company Credits, Business/Box Office, and Awards. We store the data about movies in an RDF datastore which gives us a table of 3-tuples, where each tuple is a single fact about a movie. For instance, the movie *Transformers*, if assigned the IMDb unique ID of tt0418279, would be represented by the tuple <tt0418279, Title, Transformers>. It is important to note that when we record facts about individuals who participate in a movie, we use their IMDb unique ID, so there is no confusion between people who may have the same name.

We then define two movies to be related if they share at least one individual in the same job. For instance, *Batman Begins* would be related to *The Dark Knight* because they share (as a single instance) Christopher Nolan in the job *Director*. Similarly, *Batman Begins* and *Equilibrium* would be related because they both have Christian Bale as an actor in them, even though he does not play the same role in both films. Additionally, *Batman*, *Batman Begins*, and *Batman and Robin* are all related because they share the same character Batman. All fields were initially used to create relationships. however, we discovered in experimentation that the use of the two languages and properties <*, Language, English> and <*, Country, USA> resulted in a very tight clique where a majority of movies were all related to each other. As a result of this, we removed those two entries for all movies to reduce the number of relationships in our dataset.

4.3.1 Datastore alterations

We began with all information stored in a single table in RDF-format. In the process of experimentation, we made several alterations to this in order to increase the efficiency of our classification routine.

The first change we made was to create a separate table to store all of the data about those nodes for which we have the Earnings property recorded. The motivation for this came from the queries used to generate relationships and the local classifier. To ensure that only relationships among these valid nodes were considered, we either maintained a full list of valid nodes in memory to check against, or used queries with a series of joins. However, both of these steps required that we conduct them each time a relationship was found in every iteration. We chose to implement a pre-

processing step that pulls the classification criteria from a configuration file, and inserts all the tuples of nodes that meet the requirements into the validnodes table. We then use this as our main datastore for the remainder of experiments. Also, the preprocessing step adds a tuple defining the field that the validnodes table is created on. This allows our implementation to intelligently decide if this pre-processing step needs to be conducted or if it may be skipped in later iterations. As a direct benefit of this step, our local classification step requires considerably less time to run.

The last alteration that we made was an additional pre- processing step to pre-compute the relationships in our dataset. We decided to pre-compute the relationships because of the time involved in determining them. Each individual test in each experiment spent time re-discovering the same static relation- ships. To reduce this time, we added a pre-computing step that creates a table to maintain a list of all relationships between nodes in our *validnodes* table. We maintain this as another 3- tuple of the form < NodeA, NodeB, <Relationships>>. Each row indicates that there exists at least one type of relationship between NodeA and NodeB. The link types and the values of each link type are stored in a vector. So, for instance, let us use the earlier examples of Batman Begins and The Dark Knight. We would have an entry of the form <"Batman Begins", "The Dark Knight", <Director=Christopher Nolan; Actor=Christian Bale>>. This allows us to generate one relationship table that we can then use various ways in all of our experiments.

4.3.2 Experimental Setup

As our classification criterion, we follow the example in [42] and use earnings figures in an attempt to determine whether a movie will make in excess of $2 million. We begin with a set of 259,937. We then eliminate those movies which do not have a figure recorded for Earnings. This leaves us with a set of size 5,382. Of these nodes, 3,324 earned over $2 million. This gives us a baseline accuracy of 59.9% for simply guessing that each movie will make over $2 million.

We conduct experiments at each ratio of 10/90, 20/80,..., 90/10 for training data versus test data. To do this, we randomize the order of the nodes and then construct the partition at the appropriate spot for the ratio we are testing at the time. Once we have these two random sets, we conduct four experiments using the Relaxation Labeling method of Collective Inference. For each test set, we consistently use a Naive Bayes classifier as the local classifier, but we vary the relational classifier. We repeat each test 20 times and take an average of all runs for the presented results.

4.3.3 Results

Our first experiment was to use only the local Bayes Classifier with Relaxation Labeling to establish a baseline accuracy of a non-relational classifier on our particular dataset. As can be seen in 4.1 localbayesonly, initially, increasing the ratio of labeled nodes to unlabeled nodes drastically increases the classification accuracy. However, after 40% of the nodes are labeled, the gains from additional nodes in the training set are minimal. This does show that even though we do not consider any relationships at all, by simply using a method of supervised learning, we can improve on the naive method of guessing the most populous group. This improvement is evident even in a situation where most of the class values for the nodes are unlabeled.

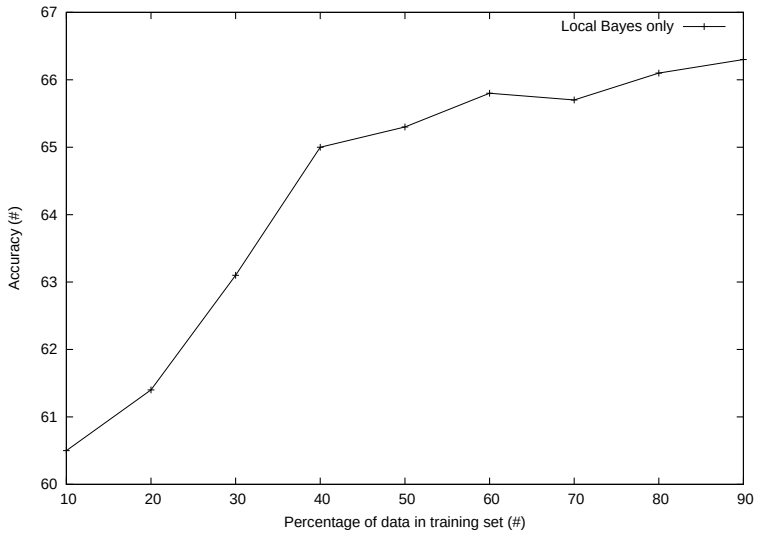


Figure 4.1. Local Bayes only

Our second experiment was conducted to establish a performance baseline of an existing relational classifier on our extended dataset. We compare these results to [42]. Specifically, the *imdb prodco - nBC* results. While our classifier does under perform in comparison to theirs, as shown in Figure 4.2, we do still see improvements upon the local Bayes classifier. We believe that the performance degradation in this result is because of overfitting. Consider that their experiment was conducted using only one attribute the production company as the determinant of relationships between movies, whereas we consider all attributes to be indicative of relationships. This large number of relationships appears to inject a higher degree of error into our trials as opposed to simply using a single attribute.

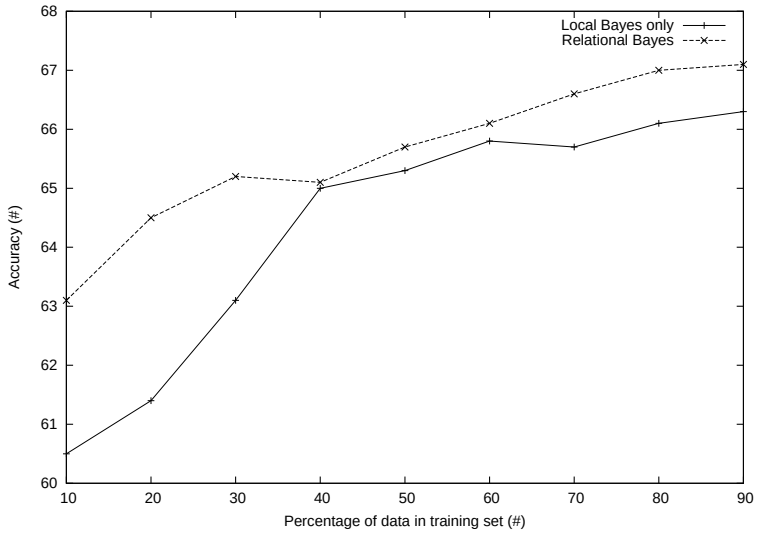


Figure 4.2. Relaxation Labeling using Relational Bayes Classifier

In our third experiment, we use the values for the link types. In this set of tests, we used the probabilities shown in Equation 4.3 to consider the link type as an additional parameter. We see quite clearly in Figure 4.3 that including the link types in the calculations increases our accuracy dramatically. Even when we had the least number of nodes in the training set, we achieved over a ten percentage point increase in accuracy over the generic relational Bayes Classifier. We believe that because we give some difference to the link types, this allows our classification method to make up for the performance loss in the previous experiments.

In our final series of experiments, we use the weighted Link Type Relational Bayes Classifier. In [22], the authors identify seven components of a movie that they use to design a Decision

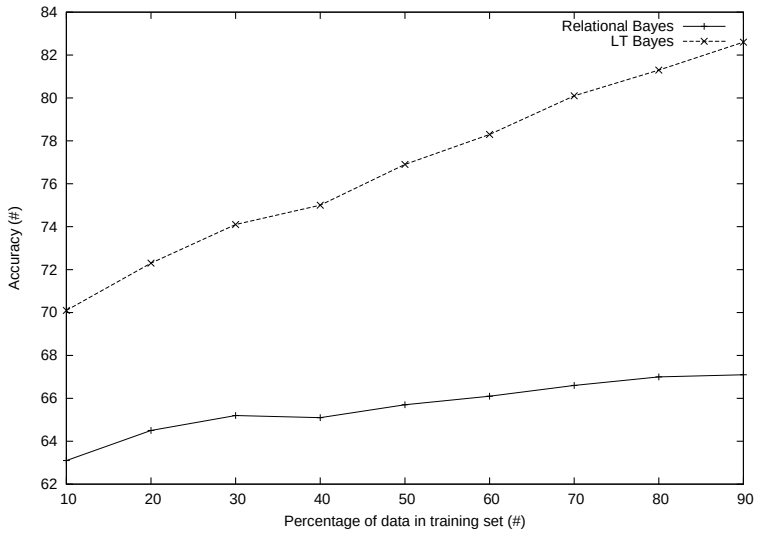


Figure 4.3. Relaxation Labeling using Link Type Relational Bayes Classifier

	$y = 0.5$	$y = 2$
$x = 0.5$	74.1	72.0
$x = 2$	83.2	78.6

Table 4.1. Summary of Accuracy Weights with Varied Tests

Support System for a Hollywood production com- pany. The areas they identify are: MPAA Rating, Competition, Star Value, Genre, Special Effects, Sequel, and Number of Screens. Some of these attributes, such as Competition, Special Effects, and Number of Screens, are opinion-based values, and are difficult to quantify for a dataset spanning the number of years ours covers. We took the their description of Star Power, and considered what current trailers for movies were listing as being an inducement for the movie. We divide the jobs in to three groups. The first group is composed writers, directors, and actors. The second group is made up of technical crew. That is, editors, costume, wardrobe, camera, and electrical. The third group is comprised of the remaining roles: producers, production company, special effects studio, etc. To test our division of roles, we conducted individual tests on a 50/50 division into training and test sets. That is, half of the nodes in the data set for these tests was unlabeled. For these tests, we specify a weight of x for the actors, directors, and writers; and a weight of y for the technical crew. The results of this experiment are shown in Table 4.1.

As shown in the table, our assumptions about the importance of respective jobs was confirmed. In those experiments where we increased the weight value of actors, directors, and writers, the accuracy of our classification calculations increased. Simi- larly, when we increase the weight value of the technical crew, the accuracy decreases from 74.1% to 72% when $x = 0.5$ and from 83.2% to 78.6% when $x = 1$.

With this consideration, we give all writers, actors, and directors a weight of 2, and specify that technical crew is given a weight of 0.5. We have a default weight of 1.0 for all other jobs, such as animation, music, choreography, etc. We show in Figure 4.4 that using these weights, we achieve substantial performance gains. Even at a relatively low percentage of known data, we are able to accurately classify nodes over 80% of the time, and at 90% of nodes labeled, our accuracy is slightly over 85%. These results indicate an improvement over the results shown in [42], where the accuracy does not seem to reach above 80% for any of the collective inference methods when using the IMDb dataset.

We believe that these figures make it apparent that with even a small amount of domain knowledge, we can use that knowledge to devise weights for link types to increase the accuracy of classification algorithms .

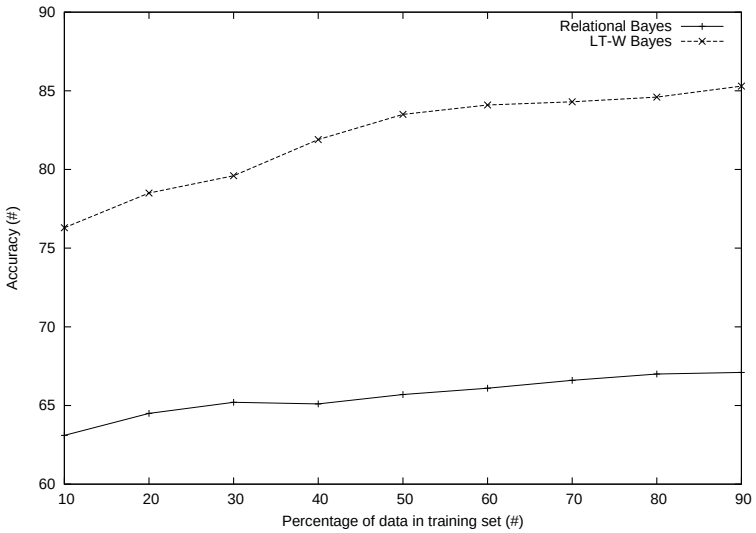


Figure 4.4. Relaxation Labeling using Weighted Link Type Relational Bayes Classifier

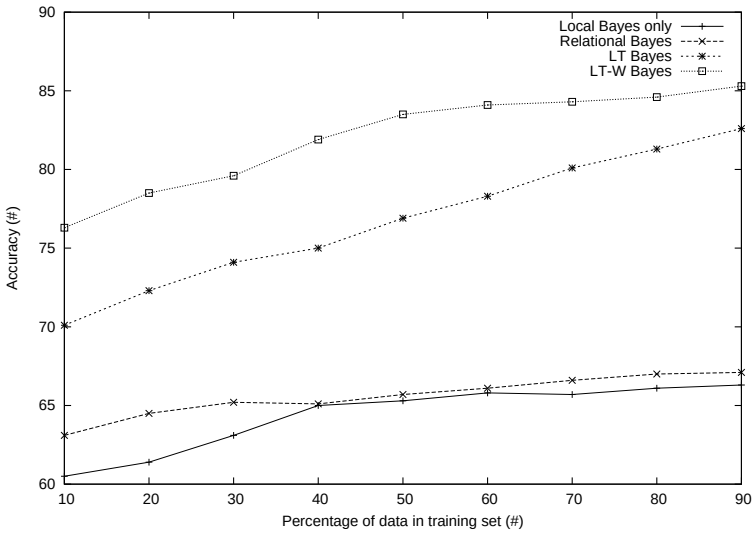


Figure 4.5. Comparison of all experiments

4.4 Conclusion and Future Work

We present for ease of comparison a summarization of all of our experiments in Figure 4.5. We believe that it shows that our enhancements to the traditional Relational Bayes Classifier, which we call Link Type rBC and Weighted Link Type rBC, are both successful extensions to the well-researched area of classification in social networks. We have shown how our extensions have improved upon the implementation of the traditional relational Bayes Classifier on our dataset, and how the overall accuracy compares to similar experiments done on a similar dataset in other research work. We believe this is a solid foundation for further examination of the inclusion of link types in social network classification research.

In the future, we believe that we need to apply these new calculations to other domains in order to determine when these new methods may be applied or when we must use the more traditional rBC implementation. Also, additional research should be conducted in programmatically determining weights for the wltrBC, perhaps by using social network metrics.

CHAPTER 5
EXTENDING CLASSIFICATION OF SOCIAL NETWORKS THROUGH INDIRECT FRIENDSHIPS

Within social network analysis, one of the most frequently referenced properties is that of *homophily* – that is, the tendency of individuals to form direct connections with those who are most like them. However, in some medical research, studies have shown that some behaviors and attitudes – such as smoking cessation – tend to propagate out from a regional focus.

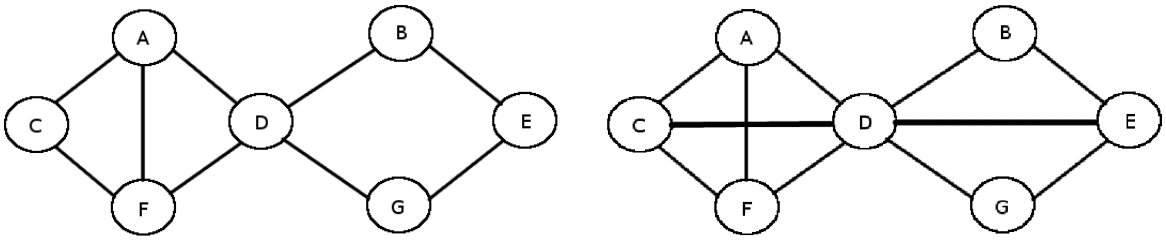
This finding contradicts the central assumptions that are used when performing relational classification. One reason for these assumptions is that they allow us to complete full-network calculations within a reasonable number of iterations. However, we seek to merge these two considerations by creating "artificial" links between regional *influencers* and those who follow in their wake.

5.1 Introduction

The utility of social network analysis has long been recognized as important for the fields of epidemiology, counter-terrorism, and business applications. The importance of having accurate classification techniques in these areas cannot be understated. By utilizing social network analysis, we are able to perform classification tasks upon nodes in the graph, allowing us to determine whether an individual may be infected with a virus, a potential terrorist, or a likely customer of an industry.

Social network analysis techniques are generally based upon the principle of homophily. That is, that individuals who are in near proximity to one another in the social graph will form relationships to one another. This property is a reflection of the adage "birds of a feather flock together."

Through application of this property, modeling of social networks typically relies upon only the first-degree neighbors of a node [42]. However, medical studies have determined that smoking cessation and obesity networks may benefit from an extended analysis [16, 17, 49]. Instead, there



(a) Original (b) Altered

appear to be nodes which influence those around them, and this influence spreads throughout the nodes in a near, non-direct relation to them.

If we focus on the counter-terrorism domain, frequently, we see that terrorist cells are decentralized with few direct contacts among members. This presents a unique challenge when attempting to use traditional social network analysis techniques in order to attempt to determine hidden patterns or perform classification tasks on these networks. Collective classification, or collective inference, techniques attempt to propagate information from one side of the social graph to another, which generally results in improved classification accuracy.

Generally, these techniques are sufficient to determine classification labels within networks where users are participating in a generally honest manner. For example, in a Facebook dataset, one may be reasonably certain that an individual isn't attempting to completely hide his relationship with another member of the network. In certain real-world networks where these classification tasks may be life-and-death challenges, such as terrorist detection tasks, the network structure is being actively hidden.

In this scenario, we need to determine who the important individuals are in the graph and attempt to extend their influence by directly connecting them to individuals who are affected by said influence. Standard inference methods do not allow this information to propagate through the graph, instead considering that all links have some pre-determined weight. Instead, we utilize network metrics in order to automatically identify the most important nodes in the graph to improve our node classification tasks.

5.1.1 Our Contributions

While other work has concentrated on the ability to determine hidden links from underlying data within the network, to the best of our knowledge, this is the first work that attempts to use the information hidden several nodes away from a target node in order to identify graph details that may be used to improve classification of this target node. For example, consider Figure 5.1(a). Node D is situated at the center of the graph, and has the highest degree centrality. We use network measures such as this in an effort to determine what nodes may have the highest regional influence within a graph and create links between these important nodes and those which may be influenced by their activities. In the case of Figure 5.1(a), node D would be connected to nodes C and E (shown in Figure 5.1(b)). We further show that by performing these techniques in a real-world social network, we can improve the accuracy of within-network classification tasks.

In Section 5.2 we give relevant background in social network classification. In Section 5.3, we explain the approach we took to evaluating our approach to information propagation. In Section 5.4 we describe the data that we use to evaluate our method. In Section 5.5, we discuss the results of experimentation of our method of classification. Lastly, in Section 5.6, we summarize our findings and provide possible avenues for further research.

5.2 Background

The traditional method of social network classification involves the use of a local classifier, a relational classifier, and a collective inference algorithm applied sequentially in an attempt to determine hidden information which is then used in each subsequent step or iteration of the algorithm.

Definition 8 *A Social Network is represented as a graph, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{D}\}$, where $\mathcal{V}$ is the set of nodes in the graph, $\mathcal{E}$ is the set of edges connecting those nodes, and $\mathcal{D}$ is the set of details containing some personal information about the members of $\mathcal{V}$. Here an individual detail is a fact that a person releases about himself, such as "v_i's Favorite Book is 'Harry Potter"'.*

Definition 9 *A classification task is a function $C : \mathcal{V} \rightarrow \mathcal{C}$. That is, given a node $v \in \mathcal{V}$, $C(v) = c_1$ is an assignment of v into class c_1, which is a valid label within the set of all possible classifications $\mathcal{C}$.*

The most basic method of classification is local classification. Local classification is a classification task with the additional restriction that $\mathcal{G} = \{\mathcal{V}, \mathcal{D}\}$. That is, the classifier may not take advantage of the information contained within the set of edges, hence the name "local."

While our proposed technique can work with any classifiers, here we explain only those we used in the experiments shown here. For our purposes, we make use of a Naive Bayes local classifier. This classifier models a classification task as

$$\begin{aligned} C(v) &= \operatorname*{argmax}_{c \in \mathcal{C}} \left[Pr(c|\mathcal{D}_v) \right] \\ &= \operatorname*{argmax}_{c \in \mathcal{C}} \left[\frac{Pr(\mathcal{D}_v|c) \times Pr(c)}{Pr(\mathcal{D}_v)} \right]. \end{aligned}$$

That is, while we wish to determine the probability of being in a certain class, c, given a specific node, v, and its details, $\mathcal{D}_v$, through application of the Bayes Theorem, we wind up with a reduced complexity in calculation. Further, we assume that our detail set is independent. This means that, for example, the probability of an individual liking the "Harry Potter" book series has no effect on whether he likes fishing as an activity.

With this assumption, it is simple to calculate the probability of any node to belong to each valid class within our classification task. The potential class label that maximizes the calculated value – that is, which has the highest probability of being true – is assigned to that class. When using collective inference, this probabilistic assignment is referred to as the *prior*.

Building on top of the local classifier is a relational classifier. Relational classifiers consider the full graph $\mathcal{G}$. However, it is important to note that generally, while the relational classifier has access to the full set $\mathcal{E}$, due to computability, only the first-degree relations of a node are considered.

In our experiments, we use the network-only Bayes Algorithm. We choose this method due to its reliability and accuracy on these data sets from other work performed [32]. Similarly to the local Bayes classifier, this algorithm uses the naive independence assumption on links. This

means that the likelihood of having a link to one person is no higher or lower because you have a link to another user.

Lastly is the collective inference algorithm. This algorithm is the method by which local and relational classifiers are used and controls the terminating condition, either a set number of iterations or convergence.

Here, we use the Relaxation Labeling method of collective inference. This method provided excellent results in previous studies ([32]), as well as ensuring that the graph is always fully-labeled. Relaxation Labeling initializes the nodes with a prior by using a chosen local classifier, and then runs repeated instances of the relational classifier until convergence.

5.3 Our Approach

Clearly, it is infeasible for us to determine the exact weight that a group or individual carries with all other members of a social network.

Then, our task is now to develop a method that will allow us to effectively estimate what nodes could be influenced by what we determine to be "important" nodes for some region of our network.

Definition 10 *The neighborhood of a node* $\mathcal{N}_v = \{u|(u,v) \in \mathcal{E}\}$.

The nodes $u \in \mathcal{N}_v$ are also referred to as the *first-degree* friends of v.

Definition 11 *The second degree friends of a node* v*, is the set of nodes* $\mathcal{N}_v^2 = \{w|u \in \mathcal{N}_v, w \in \mathcal{N}_u, w \notin \mathcal{N}_v\}$.

It is worth noting here that in our definition, $\mathcal{N}_v$ and $\mathcal{N}_v^2$ are necessarily disjoint sets for an arbitrary v.

Third degree friends are determined in a similar manner to second-degree friends.

We examine the effect of second and third degree friends in a real-world social network and attempt to determine if we can increase the classification accuracy of a collective inference classifier on this data set.

To test this, we gauge the effectiveness of several separate factors:

1. The general importance of extended relations in a graph

2. The importance of cohesive groups in near proximity to a node

3. The effective distance of cohesive groups from a node

These questions raise the issue of how to determine groups automatically. Our attempts to determine this importance are based on the use of well-established graph metrics. Specifically, based upon previous work in the area, we constrain our examination to two metrics: Clustering Coefficient and Degree Centrality.

The Clustering Coefficient is a measure of how tightly grouped a region of the graph is. Generally, it is a measure of how many of a node's friends are also friends. The Clustering Coefficient of a node, n_i is represented by the following equation:

$$CC_i = \frac{2|\{(j,k)|j,k \in \mathcal{N}_i, (j,k) \in \mathcal{E}\}|}{d_i(d_i - 1)} \tag{5.1}$$

where d_i is the degree of n_i.

The Degree Centrality of a node is the ratio of how many edges are incident to a node compared to how many are possible. This is represented by the equation

$$DC_i = \frac{d_i}{|\mathcal{V}| - 1} \tag{5.2}$$

where $\mathcal{V}$ is the set of vertices for the graph.

Since it's infeasible to add every possible friendship link of extended degree in the data set, we use a parameter, τ to define a threshold of the required metric value for propagating the information contained in a node to a target node.

That is, for an altered edge set $\mathcal{E}'$, potential edge $(u, v) \in \mathcal{E}'$ if and only if $\frac{DC_u}{Z} > \tau$ and $v \in \mathcal{N}_u^i$, where DC is the arbitrary graph metric chosen for the altered graph, Z is a normalization constant, and i is a given parameter for allowable distance of nodes.

What this measure indicates is that only those nodes which are determined as "important" – according to a pre-determined graph metric – will be selected for a region of the graph. If we instead chose a percentage of links to add, then we may be adding links which have a low probability of increasing the overall knowledge inside our graph.

However, consider a subgraph, such as that shown in Figure 5.1(a), where all shown nodes are members of class label "α". Now, consider an example where nodes "A" and "G" are labeled β while all other nodes are labeled α. Based upon the previous metrics, each of these graphs will be treated exactly the same when determining new edges. Intuitively, we reason that there should be some differences when considering them. To reflect this, we alter the traditional graph metrics in an attempt to more indiscriminately determine the flow of information through the graph. We refer to these as class-weighted graph metrics, and alter their equations as follows:

$$\begin{aligned} wCC_i &= \frac{|\{(j,k)|j,k \in \mathcal{N}_i, (j,k) \in \mathcal{E}, C(j) = C(k)\}|}{d_i(d_i-1)} \\ &- \frac{|\{(j,k)|j,k \in \mathcal{N}_i, (j,k) \in \mathcal{E}, C(j) \neq C(k)\}|}{d_i(d_i-1)} \end{aligned} \tag{5.3}$$

and

$$wDC_i = \frac{|\mathcal{N}_i(c)| - |\mathcal{N}_i(\bar{c})|}{|\mathcal{V}|-1} \tag{5.4}$$

where $\mathcal{N}_i(c)$ are the neighbors of n_i who share the same class as n_i and $\mathcal{N}_i(\bar{c})$ are the neighbors of n_i who are in the opposite class of n_i.

It is important again to note that unlike the work discussed in Chapter 2, we are not attempting to determine links which represent real-world relationships. Instead, we are attempting to model the flow of information or preferences through the graph in such a way that those individuals who are trend-setters or network influencers have an importance that is reflected in our classification tasks.

5.4 Data

The data that we use for the experiments in this was pulled from the Dallas/Fort Worth, Texas network of Facebook in May of 2007 and from the Internet Movie Database (IMDb) in September of 2007. The data was obtained from the publicly available profiles on Facebook. That is, profiles which had no privacy settings, allowing anyone within the geographic region to see the full details of their profile.

For our experiments we attempted to determine the political affiliation of nodes within the data set. Specifically, we attempt to determine whether an individual classifies himself as "Conservative" or "Liberal" – which are the two primary affiliations listed in the data. Anyone listing another affiliation, such as "Libertarian" or "Human" is considered to have listed no affiliation and is removed from the experiments.

After removing the nodes which do not list a political affiliation, we are left with 35,000 nodes to examine and approximately 400,000 edges. Of these 35,000, 55% list their affiliation as "Conservative" and 45% list their affiliation as "Liberal."

IMDb data was pulled from all movies available on the site (as of September 2007). Of all available pages, we maintained information regarding the full cast and crew, box office receipts, production company credits, and awards received. Due to the nature of this network, links are determined *a priori* based on the relationship between two movies. For our analysis, we examine the network with the following link types defined: 1) actors, 2) directors, 3) producers, 4) writers, 5) all of the previous. These link types were chosen based on accuracy of classification in previous work [31].

5.5 Experiments

When conducting our experiments, we tested for multiple situations. Because we are using the propagation of information through links we need to use relational classifiers to take full advantage of this. For the determination of the priors, we use a local Naïve Bayes classifier. Following this step, we use the Relaxation Labeling method of collective inference, utilizing a network-Bayes Classifier.

We initially tested these options on the possible second-degree nodes. That is, our experiments labeled "Neighbor" (Figures 5.1(c), 5.2(a), 5.3(a), 5.4(a)), when determining whether to add links to a given node n_i, we use the calculated values for all $n_j \in \mathcal{N}_i$. For each n_j that is included (based on the chosen value of τ) we add a link between n_i and n_k where $n_k \in \mathcal{N}_j$.

To illustrate, using this method of link addition on Figure 5.1(a), the new links would be $(A, B), (A, G), (B, F), (B, G), (F, G)$.

For the experiments labeled "2nd Degree" (Figures 5.1(d) and 5.3(b)), for any node n_i which has a minimal distance of 2 from a node with the allowed τ-strength. This is the form of link addition used to illustrate in Figure 5.1(b). Based upon the results in these figures, for the remaining tests, we use this method of link addition.

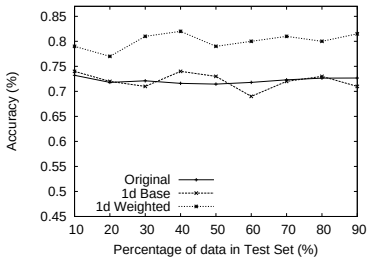
We then test for multiple possible values of labeled versus unlabeled data. That is, we divide our data set into a training and testing set with different percentages of labeled data in order to examine our method's affect on graphs which are in various states of knowledge. For each point, we run 200 experiments using randomly partitioned data. The results of these tests are shown in Figures 5.1 and 5.3. As we can see from these results, the Clustering Coefficient method of determining importance is more effective than the degree centrality, but both metrics improve upon the base accuracy of a classifier without information propagation.

Additionally, we see from the differences in Figures 5.1(d) and 5.1(e) and Figures 5.3(b) and 5.3(c) that while any method of including the second-degree neighbors is effective at increasing the accuracy of a classifier, adding the third degree nodes (instead of second, not in addition to, it should be noted) causes a reduction in classification accuracy.

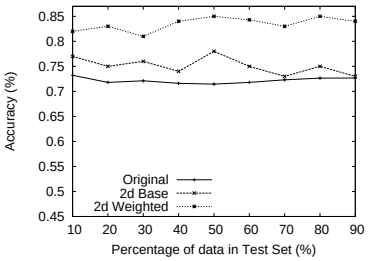
Further, we significantly improve the accuracy of our classifier. For example, using the IMDb dataset shown in Figure 5.2(b), at the 50% data point, the average classification accuracy was 0.83 with a max of 0.851 and a minimum of 0.8023 for the weighted metric. The next closest metric was the original clustering coeffcient calculation, which had an average classification accuracy of 0.6983, a maximum of 0.710, and a minimum of 0.6891. The other data points show a similar improvement.

This observation is further confirmed when we test using fourth-degree neighbors (Figures omitted for space), where the inclusion of fourth degree neighbors results in a dramatic decrease in classification accuracy.

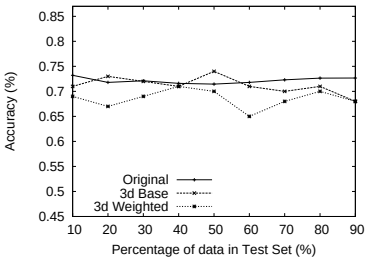
Additionally, we test on various values of τ. We note that any τ value above 1.00 is unattainable by any node in the set, and **is the original dataset**. This point is included as a baseline figure to allow for an examination of the change in accuracy as we vary for other values of τ. This lets us vary the allowed importance of the extended neighborhood. Our tests for reliable values of τ, as shown in Figure 5.5, indicate that for very high values of τ, the Clustering Coefficient is already



(c) Neighbor

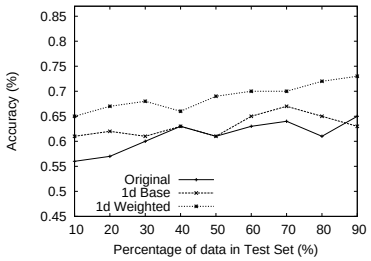


(d) 2nd Degree

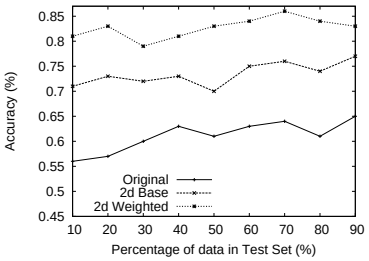


(e) 3rd Degree

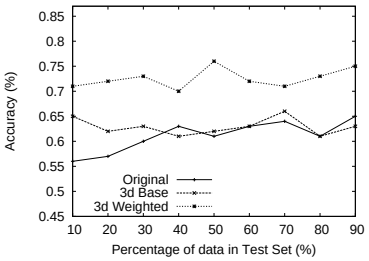
Figure 5.1. Clustering Coefficient with $\tau = 0.8$ (Facebook)



(a) Neighbor

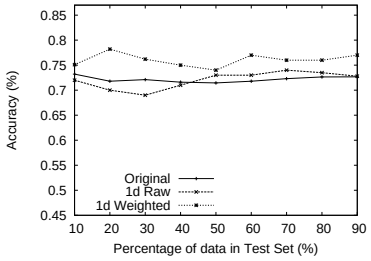


(b) 2nd Degree

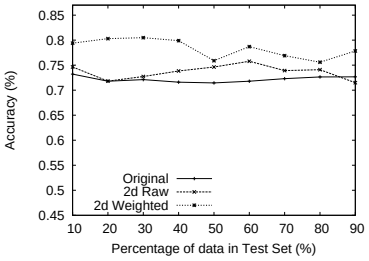


(c) 3rd Degree

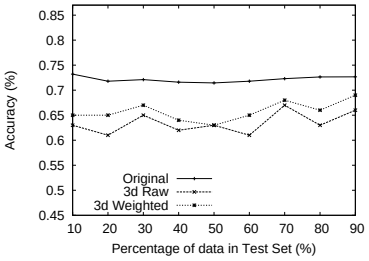
Figure 5.2. Clustering Coefficient with $\tau = 0.90$ (IMDb)



(a) Neighbor

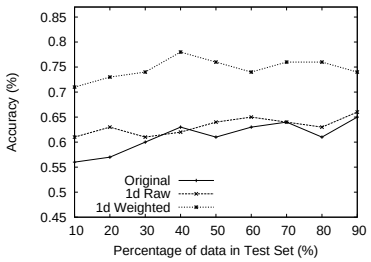


(b) 2nd Degree

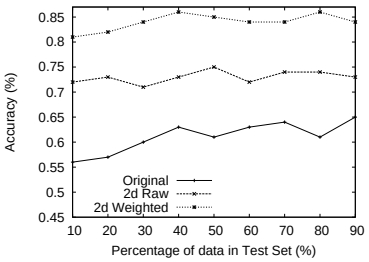


(c) 3rd Degree

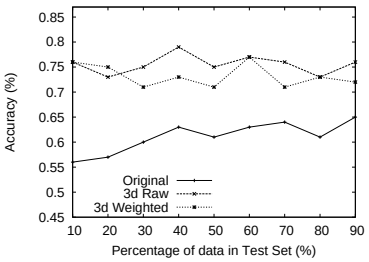
Figure 5.3. Degree Centrality with $\tau = 0.85$ (Facebook)



(a) Neighbor



(b) 2nd Degree



(c) 3rd Degree

Figure 5.4. Degree Centrality with $\tau = 0.95$ (IMDb)

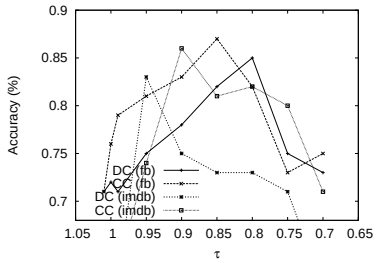


Figure 5.5. Experiments with varied values of τ

able to produce a significantly increased accuracy almost immediately.

5.5.1 Automatic Selection of Parameters

The Degree Centrality measure, however, required a lower τ to be able to reach its peak accuracy at $0.80 = \tau$. Both classifiers, however, would quickly decrease their accuracy with even lower values of τ. This effect makes sense, because the lower we set τ, then more nodes with a lower importance will be linked, until eventually all possible edges are added to the graph.

What these figures indicate to us is that while the inclusion of hidden sources of importance can increase the accuracy of classification tasks in social networks, each metric and each social network requires a specific value of τ to be effective. In a real-world scenario, one will not necessarily be able to use supervised learning techniques in order to pinpoint a specific value of τ.

To resolve this issue, we performed k-means clustering on the normalized results of each centrality measure, using 2 clusters, roughly defined as "important" and "not important". We establish initial centroids at 0.95 and 0.25, in order to weight the clusters to have a wider range of possible values in the "not important" cluster. Once k-means reaches convergence, we look at the value of the lowest node in the "important" cluster, and use this value for τ. The results of experiments using this clustering-based assignment of τ are presented in Table 5.1.

To get these figures, we performed the analysis using 50% of the data in the test set and 50% of the data in the training set using the weighted version of our second degree friends method of link addition. We show the result of the clustering method of choosing τ first for each metric within each dataset, and then for the sake of comparison, the best performer from our iterating selection. As can be seen from this figure, use of the clustering method of determining τ provides results

very similar to the iterative approach over multiple τ. While the method may not necessarily provide the optimum assignment of τ, the value it provides is very close, and is easily computable on many data sets.

Facebook		
Metric	Value (Method)	Accuracy
CC	$\tau = 0.82$ (k-means)	0.855
	$\tau = 0.80$ (Iterative)	0.853
DC	$\tau = 0.85$ (k-means)	0.751
	$\tau = 0.85$ (Iterative)	0.759
IMDb		
CC	$\tau = 0.89$ (k-means)	0.811
	$\tau = 0.90$ (Iterative)	0.832
DC	$\tau = 0.97$ (k-means)	0.873
	$\tau = 0.95$ (Iterative)	0.855

Table 5.1. k-means clustering values for τ

Lastly, it is important to note that these changes in accuracy are caused only by changes to the network structure. Because of our choice of local classifier, the local model does not change, and thus the probability of being assigned to the same prior is equal across all experiments. However, we see here that by choosing additional edges in the graph well, we can improve the classification accuracy of the network classifier.

5.6 Conclusions

We have shown here that by intelligently choosing "important" nodes in a subgraph, one is able to increase classification accuracy on real-world social network data. We present a method of selecting nodes by using weighted graph metrics and a programmatically determinable threshold parameter.

Further, we show that our τ-threshold is a useful tool for determining the general increase in graph size versus a desired amount of classification accuracy, and that it may be possible to determine a beneficial value of τ through calculation of the chosen degree metric.

It is important to note, however, that the data set we used is not sufficient for determining the direction of information flow through a graph. For this, we would need to be able to obtain a data

set that allows us to observe changing values over time, which is not available in our Facebook data set. As a future direction for research, obtaining access to data which has changes over time is necessary to determine the actual flows of data through a network.

As an additional direction, testing additional graph metrics, or the creation of specific metrics for use in information flow, could be useful for increasing the accuracy of classifiers even more.

CHAPTER 6
SOCIAL NETWORK CLASSIFICATION THROUGH DATA PARTITIONING

One consideration for social network providers is that this data will accumulate over time. Because of the highly desired nature of social network longitudinal data, it is unlikely that a network such as Facebook will ever delete data they obtain. Even if later revisions directly contradict previous entered data, this is still valuable in tracking change in attitudes over time.

Because of this, performing calculations and inference on this data will become more difficult over time, because of the size of this data. However, we now show that by intelligently examining this data, we are able to partition the full data into smaller sets, which both increases classification accuracy and reduce the time required for classification.

6.1 Introduction

Each social network allows users to list certain details about their personal or professional lives. While social network administrators can see all of the information that users specify, the data that users hide from the network may also be of interest to these administrators. Most of these sites are free to the end user and are thus advertising-supported. If we assume that advertisers want to reach the people most likely to be interested in their products, then identifying those individuals becomes a priority for maintaining much-needed advertisers. However, by just using specifically defined information, the site may be missing a large number of potentially interested users. Taking the specified knowledge from some users and using it to infer unspecified data may allow the site to extend its target audience for particular advertisements.

For example, if Pepsi is using Facebook to advertise a new, grapefruit flavor of Mountain Dew, then one approach is advertising to people who mention that Mountain Dew or grapefruit sodas are their favorite. However, what about people who do not list their drink preferences? We may be able to infer from other activities, such as being a member of video gaming-based groups, that an individual would probably buy the new soda.

We are able to model various methods of human interaction as a social network, and, in some, predictive models are considerably more important than targeted advertising. Consider, for example, counter-terrorism goals of classifying whether an individual is involved in terrorist activities or not. We can easily model this as a social network classification task. By using telephone records, e-mail logs, or other similar data sets, we can model almost any human interaction in the form of a social network, as shown in [37].

As social networks become more popular, more data is stored about each individual and more individuals are joining each social network. The large increases in data provide both a benefit and a detriment to these operations. As a benefit to classification, as there is more information, we are able to define a much more fine-grained model that may improve the classification accuracy. On the other hand, because many social network classification tasks depend on repeated iteration of calculations performed on the entire network, each of these iterations will take considerably more time and memory, as both the number of individuals and the number of connecting links increase.

6.1.1 Our contributions

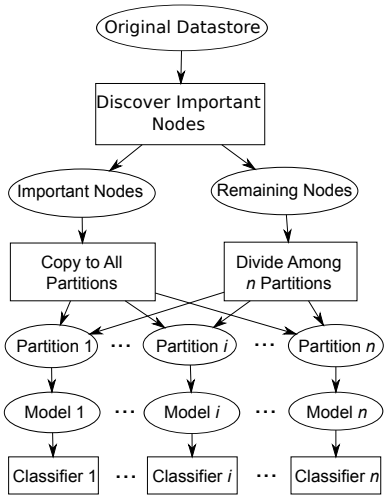


Figure 6.1. System Architecture

We provide a reference for the architecture of our system in Figure 6.1. Our goal is to efficiently classify a large amount of social network data without loss of accuracy. To satisfy this goal, we propose a partitioning technique that can leverage multi-core or Map-Reduce architectures. Using well-known centrality measures, we identify important nodes in the graph. Later, we

heuristically allocate important nodes from the original data store to multiple (partitioned) data stores, and then intelligently divide the remaining nodes in the original data store to the partitioned data stores. We then develop in-partition models that we use to perform a given classification task on each partition.

In addition, we also utilize the concept of Information Gain-based partitioning. We use information gain on each data set to determine the best single attribute, and define a division on this attribute to be a partitioning scheme unique to each data set, as that attribute may not exist in other data sets.

Obviously, partitioning the data in the social network alone will decrease the time required for classification. However, care must be taken when partitioning. Naïve methods of partitioning may decrease execution time at the expense of accuracy. Our contribution is important because we focus on maintaining efficient classifiers while decreasing execution time. Further, while we focus our work on multi-core architectures, we consider partitioning in a way that allows the operations on each processor to operate independently, and in a way that does not rely on any cross-partition communications, which we believe makes them easily applicable to a Map-Reduce architecture.

The remainder of this paper is structured as follows: In Section 6.2, we describe all of the metrics that we will use in the subsequent sections. We also define the process of collective inference and describe how it will be used in the classification section of the experiments. In Section 6.3, we detail the specifics of the partitioning schemes we developed. In Section 6.4, we provide the results of our experiments. In Section 6.6, we offer some conclusions and potential future work in this area.

6.2 Background Information

We begin with a discussion of the graph metrics we use to measure the importance of nodes. We chose these graph metrics because they have been extensively studied in other graph-related applications, such as [34]. By using these as part of a baseline, we can see the effects of division based purely on the structural composition of the graph.

6.2.1 Graph Metrics

For our work, we model a social network as an undirected graph, $G = \{\mathcal{V}, \mathcal{E}, \mathcal{A}\}$. $\mathcal{V}$ is the set of all nodes, $\mathcal{E}$ is the set of all edges, and $\mathcal{A}$ is the set of all attributes. For example, nodes $v_i, v_j \in \mathcal{V}$ are two nodes with an edge, $(v_i, v_j) \in \mathcal{E}$, between them. v_i represents a person named 'John Smith' who is 20 years old. That is, $a_i = \{Name = `JohnSmith'; Age = 20\}, a_i \in \mathcal{A}$. Further, we use the notation $d(v_i)$ to indicate the degree measure of node v_i, that is the number of links v_i is involved in.

To measure the importance of nodes in the graph, we chose several metrics as discussed in [34]. Specifically we chose Clustering Coefficient and Degree Centrality for the discussion of our work here[1]. We briefly describe the two relevant metrics here for completeness.

The Clustering Coefficient is a general measure of cliquishness. Given a node v_g, we're interested in how often two friends of v_g, v_j and v_k, where $v_j \neq v_k$, are themselves friends. Specifically, we compute the Clustering Coefficient for a node v_g through the following formula:

$$CC(v_g) = \frac{COUNT((v_k, v_j) \in \mathcal{E} | v_k \neq v_j; v_j, v_k \in \mathcal{N}(v_g))}{d(v_g)(d(v_g) - 1)/2} \tag{6.1}$$

where $\mathcal{N}(v_g)$ is the set of neighbors of v_g.

The Degree Centrality of a node is a simple normalized measure of how many links a node has compared to the highest possible number of neighbors in any graph composed of $|\mathcal{V}|$ nodes. Formally, we calculate this as

$$DC(v_g) = \frac{d(v_g)}{|\mathcal{V}| - 1}.$$

We define "Information Gain" metrics to be a heuristic defined for a specific data set that uses some feature of that data chosen, chosen through Information Gain, as a method of partitioning the data. For instance, in our Facebook data set, a potential information gain-division would be the specific geographic region within the Dallas/Fort Worth area. We could consider each town/city (Dallas, Allen, Plano, etc.) as well as each university (University of Texas at Dallas, Southern Methodist University, etc.) to be a unique location for this partitioning. For our experiments, we use information gain to compare the ability of various attributes to partition the data set. We

[1] Initial work was also conducted using Closeness Centrality and Betweenness Centrality, but discussions of these metrics are omitted for space, since these metrics performed significantly poorer than CC and DC.

choose the attribute with the highest information gain ratio to partition the network. However, both of our data sets contain information that may have multi-valued attributes. For example, an individual could list fifty books as their 'Favorite'. Because of this, we use the information gain ratio proposed in [48]. First, information gain is calculated as the difference between the entropy of the full data set and the conditional entropy if the data was partitioned based on a particular attribute. To do this, we need to first determine the Entropy of the test set for a given attribute A^n (i.e. 'Favorite Books') by

$$Entropy(\mathcal{V}_t) = \sum_{i=1}^{|A^n|} -Pr(A_i^n) log_2[Pr(A_i^n)],$$

where $\mathcal{V}_t$ is the set of all nodes in the training set, $|A^n|$ is the total number of entries for attribute A^n and A_i^n is the value of the i^{th} attribute (i.e. 'Harry Potter'). We then calculate the information gain for the attribute under consideration by the equation

$$IG(\mathcal{V}_t, A^n) = Entropy(\mathcal{V}_t) - \sum_{A_i \in A^n} \frac{|\mathcal{V}_{t,A^i}|}{|\mathcal{V}_t|} Entropy(\mathcal{V}_{t,A^i})$$

where $A_i \in A^n$ represents a single possible attribute value (i.e. 'The Shining' as a favorite book) and $\mathcal{V}_{t,A^i}$ represents a division of the training set based on attribute A^n that contains attribute A_i. We can now calculate the information gain ratio by

$$IGR(\mathcal{V}_t, A^n) = \frac{IG(\mathcal{V}_t, A^n)}{-\sum_{k=1}^{|A^i|} \frac{|\mathcal{V}_{t,A^k}|}{|\mathcal{V}_t|} log_2[\frac{|\mathcal{V}_{t,A^k}|}{|\mathcal{V}_t|}]}$$

We also use a completely random partitioning scheme as a baseline to identify what advantage we would gain just by distributing nodes to partitions uniformly at random and performing classification on these smaller sets. As such, we also do not replicate any nodes for this metric.

6.2.2 Collective Inference

Collective Inference is a method of classifying social network data using a combination of node details – such as favorite books and movies – and connecting links in the social graph. Each of these classifiers consists of three components: Local classifier, relational classifier, and Collective Inference algorithm.

Local classifiers are a type of learning method that is applied in the initial step of collective inference. Generally, local classifiers build models based on attributes of nodes in the training set. Then the models are applied to nodes with an unknown label to classify them.

Relational classifiers are a separate type of learning algorithm that looks at the link structure of the graph, and uses the labels of nodes in the training set to develop a model which in turn is used to classify the nodes in the test set. Specifically, in [42], Macskassy and Provost examine four relational classifiers: Class-Distribution Relational Neighbor (cdRN), Weighted-Vote Relational Neighbor (wvRN), Network-only Bayes Classifier(nBC), and Network-only Link-based Classification (nLB).

In the wvRN relational classifier, to classify a node n_i, each of its neighbors, n_j, is given a weight. The probability of n_i being in class C_x is the weighted mean of the class probabilities of n_i's neighbors. That is,

$$P(n_i = C_x|\mathcal{N}(v_i)) = \frac{1}{Z} \sum_{n_j \in \mathcal{N}(v_i)} [w_{i,j} \times P(n_j = C_x)]$$

where $\mathcal{N}(v_i)$ is the set of neighbors of n_i, and Z is a normalization factor.

A major problem with relational classifiers is that while we may cleverly divide fully labeled test sets so that we ensure every node is connected to at least one other node in the training set, real-world data may not satisfy this strict requirement. If this requirement is not met, then relational classification will be unable to classify nodes which have no neighbors in the training set. Collective Inference attempts to make up for these deficiencies by using both local and relational classifiers in a precise manner to attempt to increase the classification accuracy of nodes in the network. By using a local classifier in the first iteration, collective inference ensures that every node will have an initial probabilistic classification. The algorithm then uses a relational classifier to re-classify nodes.

We choose to use Relaxation Labeling as described in [42] because in the experiments conducted by Macskassy and Provost [42], Relaxation Labeling tended to be the best of the three collective inference methods.

6.3 Distributed Social Network Classification

6.3.1 Node Replication to Partitions

Since the goal of this work is to examine the efficacy of a distributed system for classifying social network data, we developed a multi-core algorithm to calculate the standard metrics which can easily be extended to a standard distributed architecture. For Degree Centrality, we simply distribute the adjacency list for each node to a separate processor. For Information Gain based Partitioning (IGBP), when calculating the Information Gain for each attribute, we distribute the processing of each attribute to a separate processor.

Algorithm 1 CC_Multi-core()

1: **for** $v_i \in \mathcal{V}$ **do**
2: $\mathcal{N}_i \leftarrow \{v_j | (v_i, v_j) \in \mathcal{E}\}$ {first-degree neighbors of v_i}
3: $r_i =$ FindPossibleTriples($v_i, \mathcal{N}_i$)
4: **end for**
5: **for** $(v_i, v_j, v_k) \in r_i$ **do** {H is a Hashmap}
6: IncrementCount(Hash(v_i, v_j, v_k), H)
7: **end for**
8: **for** $v_i \in \mathcal{V}$ **do** {initialize counters}
9: $c_i = 0$
10: **end for**
11: **for** $(k, v) \in H$ **do**
12: **if** $v = 3$ **then**
13: **for** $v_i \in k$ **do**
14: c_i++
15: **end for**
16: **end if**
17: **end for**
18: **for** $v_i \in \mathcal{V}$ **do**
19: CC_i = ClusteringCoefficient($c_i, \{(v_i, v_j) \in \mathcal{E}\}$)
20: **end for**

We give the general algorithm for the multi-core Clustering Coefficient in Algorithms 1. Algorithm 1 is the general program flow, where the master processor distributes the list of neighbors to a different processor (Lines 2–3). When the processors complete, we have a series of return results, where each entry of the result is a node-triple. For each triple, we store the number of times that triple has been returned by some processor (Line 6) in a hashed datastore, called H.

Algorithm 2 *FindPossibleTriples($v_i, \mathcal{N}_i$)*

Remark 6.3.1
```
1: r ← ∅
2: Sort(N_i)
3: for v_j ∈ N_i do
4:     for v_k ∈ N_i | v_k > v_j do
5:         r ← Sort(v_i, v_j, v_k | v_i < v_j < v_k)
6:     end for
7: end for
8: Return r
```

After we have gone through all results of the processors, we iterate through the keys in H and if a key has been seen three times, then for each node in the triple, we iterate a counter for that node of how many triples that node is involved in (Lines 11 – 17). After this, we distribute the calculation of the Clustering Coefficient to other processors, which is a simple application of Equation 6.1.

The FindPossibleTriples (FPT) method gets a list of all the neighbors of a given node, v_i and then returns a sorted list of all triples of nodes that have a shared friend of v_i. The logic of this is the following: Suppose that nodes v_i, v_j, and v_k are a clique. Then $v_j, v_k \in F_i; v_i, v_k \in F_j; v_i, v_j \in F_k$. So FPT for each of these will return (v_i, v_j, v_k) (assuming $v_i < v_j < v_k$). This means that the main algorithm will have a count of 3 for this triple, and each node will be listed in at least one cluster for the actual CC calculation.

We began by devising methods by which we could partition the nodes in a social network. As noted in [4], there are some nodes in the social network which are large influencers of the remainder of the network. Since these influencers have an effect that may not be simply related to the direct nodes they touch, their effects propagate throughout the graph. To retain this property, we devised methods of keeping these important nodes.

For experiments using the metrics described in 6.2.1, we use the value calculated for each of those metrics to define the 'important' nodes. For IGBP, we assume that the major influencers for any particular group stay in that group and will end up in the same partition. Therefore, we do not replicate any nodes. For the completely random partition, we also do not attempt to replicate any nodes[2].

The benefit of this is that while the learning model built on each partition for classification has fewer overall nodes to classify, those nodes that are present are, in theory, very indicative of

[2]Used for comparison only.

the overall trends in the graph. By making sure that each partition has these nodes, then we can work to ensure that each partition's model is more accurate while maintaining the efficiencies of smaller datasets.

6.3.2 Partitioning

Algorithm 3 CC_DC_Partition(M, r, p)

1: $c_i = \{M(v_i)|v_i \in \mathcal{V}\}$ $\{M$ is the calculation for the chosen metric$\}$
2: Sort $\mathcal{V}$ {In descending order on c}
3: **for** i = 1 to $\lfloor|\mathcal{V}| \times r\rfloor$ **do** {replication}
4: **for** j = 1 to p **do**
5: $P_j = P_j \cup v_i$
6: **end for**
7: Remove v_i from $\mathcal{V}$
8: **end for**
9: **while** $\mathcal{V} \neq \emptyset$ **do** {partioning}
10: $j \leftarrow ARGMIN_{|P_j|}(p_j \in P)$
11: $P_j = P_j \cup v_1$
12: **for all** $v_k \in \mathcal{N}(v_1)$ **do**
13: $P_k = P_k \cup v_k$
14: Remove v_k from $\mathcal{V}$
15: **end for**
16: Remove v_1 from $\mathcal{V}$
17: **end while**

Algorithm 3 describes the general partitioning step used for DC and CC, which requires as input the metric, M to be used (CC or DC), the replication rate r, and the number of partitions to be used p. We use the metrics described in Section 6.2.1 to partition the data. We do this by taking the specific metric to be used in the experiment and finding the specific numerical value for that metric using algorithms described in Sections 6.2.1 and 6.3.1 (Line 1). We then use the replication rate to choose the highest percentage of nodes to replicate across all partitions (Lines 3–8). Among the remaining nodes, we pick the next node with the highest metric value and add it to the first partition[3]. We then take each of that node's first-degree neighbors and add it to the same partition (Lines 12–14). We repeat on the remaining nodes in the list, adding each subsequent choice to the partition with the lowest cardinality (Lines 14–22).

[3]The choice of partition could be random without loss of generality

Algorithm 4 DS_Part(p)

1: h = Entropy($\mathcal{V}_t$)
2: **for** $A^n \in \mathcal{A}$ **do**
3: $\quad i_{A_n}$ = IG($\mathcal{V}_t, A^n$)
4: $\quad c_n$ = IGR($\mathcal{V}_t, A^n$)
5: **end for**
6: $c \leftarrow ARGMAX_n\{c_n\}$
7: **while** $\mathcal{V} \neq \emptyset$ **do** {partioning}
8: $\quad v_i \leftarrow_R \mathcal{V}$
9: $\quad a_i \leftarrow ARGMAX_j\{Count(a^c_{i,j})\}$
10: $\quad j \leftarrow ARGMIN_{|P_j|}(p_j \in P)$
11: $\quad P_j = P_j \cup v_1$
12: $\quad$ **for all** $v_k \in \{v_k \in \mathcal{V} | a_i \in a^c_k\}$ **do**
13: $\quad\quad P_k = P_k \cup v_k$
14: $\quad\quad$ Remove v_k from $\mathcal{V}$
15: $\quad$ **end for**
16: $\quad$ Remove v_1 from $\mathcal{V}$
17: **end while**

The IGBP is based on the specific nature of the social network and is shown in Algorithm 4 and takes as an input only the number of partitions to be used. We first calculate the IGR for each attribute (Lines 1–5). We then find the attribute that has the highest IGR (Line 6). We then pick a random node (Line 8) and find the most frequently observed value of the attribute. For some single-valued attributes this is straiht-forward. For multi-valued attributes, we simply choose the most frequent value. We then place all the nodes with that value into the partition with the fewest items (Lines 12–15). We then repeat this with another randomly chosen node.

It is important to note that we do not attempt to maintain an equal number of nodes in each partition. We merely add nodes to a partition with the lowest cardinality in an attempt to keep the partitions roughly balanced in size. Based on the information gain calculations, for the Facebook data set, we divide the data based on group memberships (i.e. the group named ‘Linux’), and for the IMDb data set, we partition the data based on the decade in which the movie was made.

6.4 Experiments

For these experiments, we have identified three major parameters that we vary in our experiments:

1. Replication rate - We assume that there are some nodes in the graph which are extremely important for classification tasks. These nodes will be copied to all partitions to increase the accuracy of the classifier. Obviously, if we have too low a replication rate then we will not get the most benefit from its use; similarly, if we have too high a replication rate then we will gain no computational benefit from dividing the data into partitions.

2. Training/test ratio - this measure indicates the uncertain nature of a variety of released social networks. We vary the percentage of nodes in our training set from 50% to 90%. In this way, we are able to test our partitioned classifiers on a variety of graphs with unknown data.

3. Number of partitions - As our goal is to attempt to optimize social network classification for multi-processor or multi-core environments, we divide our data into one through eight partitions. This allows us to test for use on dual- and quad-core computers while overlapping as little work as possible.

When conducting our experiments, we chose to focus on two different situations to identify the ability of our system to accurately classify partitioned data. First, we wished to determine how varying the size of the training set in each partition would affect the accuracy of our distributed classification technique. We do note here that in a real-world scenario, the actual availability of nodes about whom the target data is known versus the number of nodes about whom the target data is being inferred will vary by partition. However, in testing the partitions repeatedly across many test/training divisions, we believe that we accurately indicate the ability of our partitioning and classification scheme. For the sake of brevity, we include here only results from tests where 50%, 60%, 70%, 80% and 90% of the nodes in each partition are in the training set. By including these, we give the accuracy of our approach on what is frequently the most difficult scenario to model – where the training method has very little information to build the classification model on.

In each of these scenarios, we measured the following:

1. The total run-time required to partition and classify the data

2. The average classification accuracy of the partitioned data

To calculate the classification accuracy for each partition, we report accuracy over 50-fold cross validation runs. We compute the accuracy of each run by averaging the accuracy measurements of each classifier built on a partition.

Our second consideration was for the amount of data replicated across all partitions in a test. We establish the test set prior to calculation of the metrics, which means that the nodes in the test set are randomly distributed among the partitions with no attempt made to balance the number of nodes with the unknown class value in each partition.

6.4.1 Data Sources

The Facebook data was collected by a program written to crawl the Facebook network. Written in Java 1.6, the crawler loads a profile, parses the details out of the HTML, and stores the details inside a MySQL database. Then, the crawler loads all friends of the current profile and stores the friends inside the database both as friendship links and as possible profiles to later crawl.

Because of the sheer size of Facebook's entire social network, the crawler was limited to only crawling profiles inside the Dallas/Forth Worth (DFW) network as of July 2008. This means that if two people share a common friend that is outside the DFW network, this is not reflected inside our data set. Also, some people have enabled privacy restrictions on their profile which prevented the crawler from seeing their profile details. For our experiments, we attempt to classify individuals as either 'Conservative' or 'Liberal' with regard to their "Political Views". We use only nodes that specify their political views in their Facebook profile. With this additional constraint on the composition of the data, this data set is comprised of 35,000 nodes.

The second set of data for our experimentation came from the Internet Movie Database (IMDb). This is an online repository of data regarding movies and television shows with every member of the cast and crew, from directors and actors to grippers and gaffers. We developed a Java 1.6 crawler that downloaded the details of every title (as of November 2008) in five categories: Main Details, Full Cast and Crew, Company Credits, Box Office/Business, and Company Credits. Since the standard classification attribute for IMDb is 'Box Office Earnings,' and because there is no entry for this in television episodes, we keep track of only movies. For this data set, we have 26,000 nodes.

6.4.2 Results

Accuracy

It is important to note that the accuracy for the **one-partition** entry is the accuracy of classification performed on the **entire dataset** with no partitioning scheme used.

In Figure 6.2, we show the results of the experiments conducted on the Facebook data set. Because of this, all one partition data points are exactly the same, regardless of replication or partitioning metric, since neither are actually used in that experiment. We include this point on each figure for ease of comparison of the improvement of each method.

Figures 6.2(a) – 6.2(e) indicate that use of the IGBP heuristic for partitioning provides the greatest increase in accuracy when we use even a single additional partition, and this continues across all tested sizes of training set. The final difference in accuracies (with a full eight partitions) is most apparent in those experiments where the size of the training set was smallest – that is, with a training set of 50% or 60% (Figures 6.2(a) and 6.2(b), respectively). For training sets of 70% (Figure 6.2(c)), at the eight partitions mark, Clustering Coefficient is approximately 2 percentage points away from IGBP in terms of accuracy. At this point, Degree Centrality has more than a 10% difference in accuracy. For training sets of 80% and 90% (Figures 6.2(d) and 6.2(e), respectively) this difference again shrinks with Degree Centrality being more accurate at the 80% test and Clustering Coefficient being more accurate at 90%.

We see slightly different results for experiments with the 20% replication rate. In the 50% experiment, we again see that Domain specific is significantly more accurate than either of the other metrics. However, what we can see from each graph is that the largest jump comes from the change from a single partition to two. We see that there is a significant difference at even the two partitions entry for the same partitioning metric. This should be attributable only to the greater replication rate, since that is the only difference in the two experiments. One would expect that differences attributable to the random selection of the training set would be less significant over the multiple runs of the experiment we performed. It is important to note that this does not improve the accuracy of the IGBP metric, since, as mentioned previously, there is no node replication for this metric.

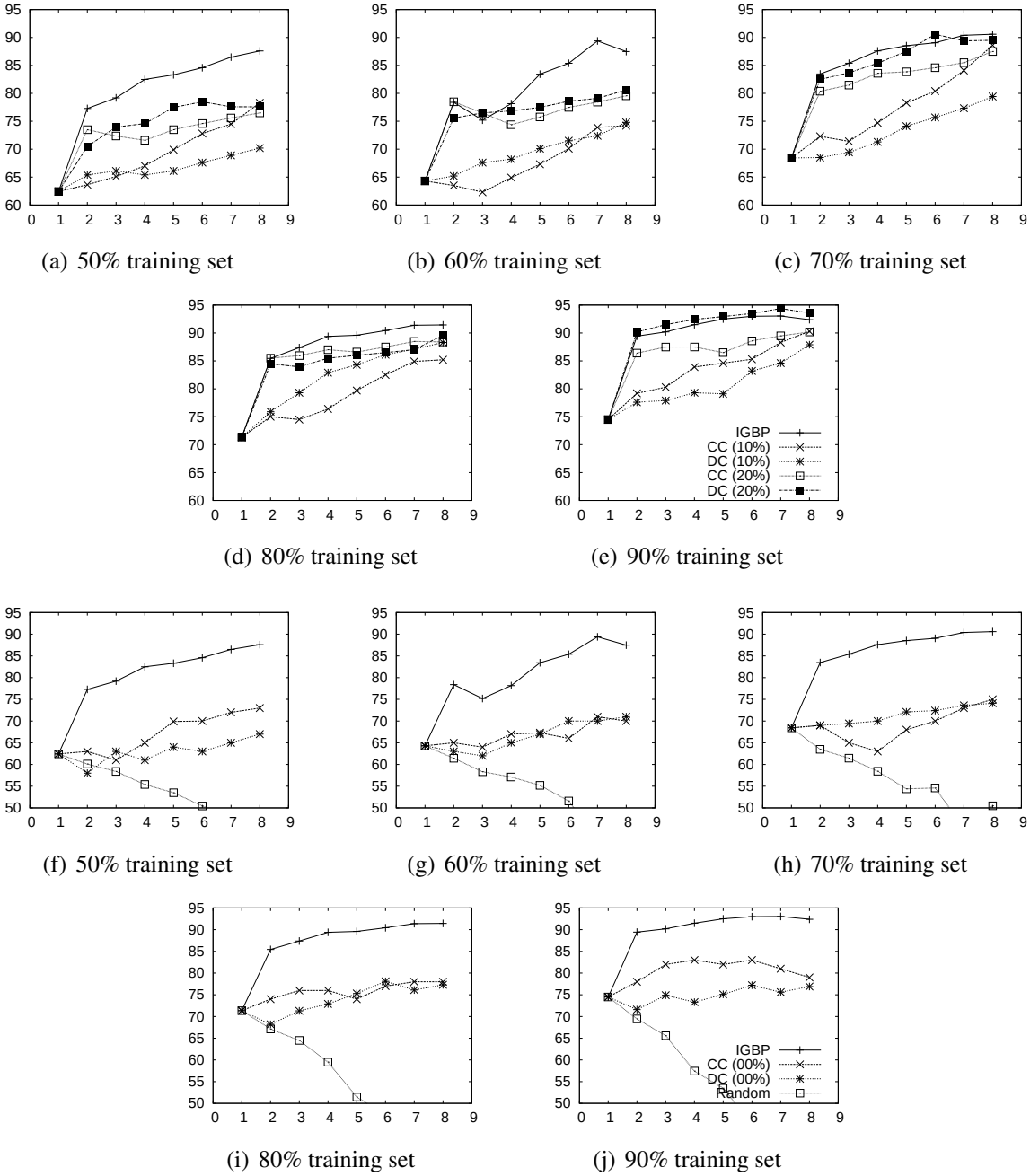
Continuing, we found that the wide disparity in classification accuracies found in the earlier experiments does not continue with the 20% replication rate. The Clustering Coefficient and Degree Centrality measures quickly increase their accuracy, while only infrequently classifying more accurately than the IGBP metric, and steadily increase throughout most of the experiments. One interesting finding is that the change from 80% to 90% training set sizes does not increase the classification accuracy of the Clustering Coefficient metric as much as it had previously, indicating that for any partitioning of a graph that has few unknowns, it is the worst performing metric we used.

We show the results of our baseline tests in Figures 6.2(f) – 6.2(j). We also include the IGBP line for further ease. The Random partition, as shown, far underperforms any of the intelligent partitioning. For the 0% replication rate, the accuracy only increases slightly, compared to those where we replicate nodes, though the accuracy increase is generally fairly stable. It is worth noting here that while we do not use the metrics to find 'important' nodes in the graph, we do still use them to determine the order that we pick nodes for partitioning. Since we choose in descending order based on the calculated value for the metric, this explains the reason for the difference between DC and CC for 0% replication.

In Figure 6.3, we see results from the IMDb data set similar to those observed in the Facebook data set. Overall, the IGBP metric outperforms both Degree Centrality and Clustering Coefficient, with only a few incidents where one of the other two obtains a higher classification accuracy (such as 2 partitions in Figure 6.3(c)).

One interesting feature of the IMDb data set is that for graphs where there are only a few nodes in the graph whose class values are unknown, the classifier is less affected by the replication rate than was shown by the Facebook dataset. This may merely be a factor of the different natures of the data sets – Facebook is a dynamic, ever-changing representation of the same among individuals. IMDb will rarely change older nodes due to updates for newer movies. However, our experiments indicate that even for these diverse types of data sets, increasing the replication rate from 10% to 20% does not hurt the classification accuracies, so for the levels we tested, there appears to be no detriment to the replication rate.

We again show the results of Random Partitioning and 0% node replication in Figure 6.3 with the IGBP line for reference. We get different results from the baseline view on Figure 6.2. In the



(a) 50% training set (b) 60% training set (c) 70% training set

(d) 80% training set (e) 90% training set

(f) 50% training set (g) 60% training set (h) 70% training set

(i) 80% training set (j) 90% training set

Figure 6.2. Facebook accuracy with 10% and 20% node replication (6.2(a) – 6.2(e)) and Random partitioning and 0% node replication (6.2(f) – 6.2(j))

Facebook							
	2	3	4	5	6	7	8
CC	1.273	1.211	1.19	1.03	0.934	0.866	0.831
DC	0.943	0.932	0.876	0.831	0.712	0.603	0.519
IGBP	1.43	1.399	1.28	1.058	0.953	0.64	0.416
IMDb							
CC	1.473	1.348	1.223	1.043	1.006	1.034	0.948
DC	0.853	0.749	0.711	0.634	0.598	0.594	0.475
IGBP	1.834	1.547	1.023	0.955	0.685	0.504	0.448

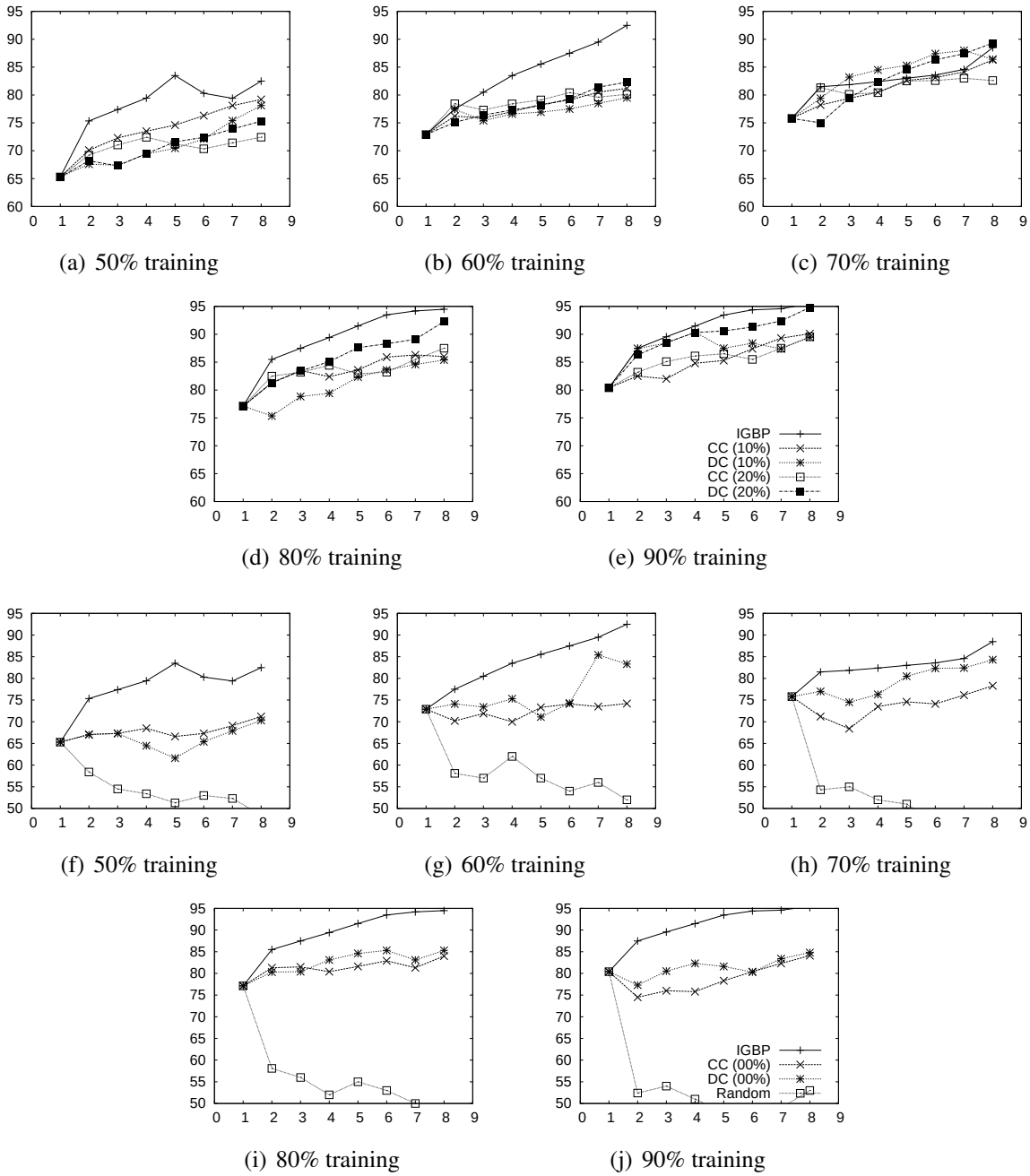
Table 6.1. Average Experiment Run-time as a Percentage of Single Partition

IMDB data set, the replication does not seem to have as large an impact on the final accuracy as it does for Facebook. The reason for this may be in the nature of the data set. Since IMDb isn't specifically a social network, but is merely modeled as one, it is possible the 'important nodes' theory doesn't hold as strictly with this type of data set.

Execution Time

All timing experiments were conducted on an IBM x3500 server with two Intel Xeon quad-core 2.5 GHz processors and 16GB of RAM. For recording the elapsed time, we used Java 1.6 time-keeping methods. Because our tests are intended to give an indication of what level of time/accuracy trade-off one can expect for dividing social networks, we are interested only in the total run-time of an experiment. That is, the total time, including numerical calculation for the division metric, the time required for the segregation and complete classification of nodes. For these, we do not record the time taken for each classification task, but the real-world time that has elapsed while the threads were performing classification.

When examining the amount of time that it takes to separate each data set as a percentage of the time that it takes to classify the entire original data store (73 minutes for Facebook, and 253 minutes for IMDb). We see several trends that make sense. First is that using Degree Centrality as a metric for data set division is always a low-cost option. Further, we see that the IGBP metric has a high overhead for the calculation of Information Gain. In fact, using the IGBP metric for fewer than five portions actually increases the amount of time that can be taken for classification tasks.



(a) 50% training (b) 60% training (c) 70% training

(d) 80% training (e) 90% training

(f) 50% training (g) 60% training (h) 70% training

(i) 80% training (j) 90% training

Figure 6.3. IMDb accuracy with 10% and 20% node replication (6.3(a) – 6.3(e)) and Random partitioning and 0% node replication (6.3(f) – 6.3(j))

6.5 Discussion of Results

These results led us to consider the reason for the increase in accuracy through partitioning. To further clarify our results, we broke the tasks down to examine the effects of the local classifier versus the benefit gained by using relaxation labeling – that is, the use of the relational classifier. When calculating the accuracy in all tests, we find the classification accuracy after the local classifier. We then find the classification accuracy at the end of the relaxation labeling step of that experiment. For both data sets, local classifier accuracy for the IGBP, DC, and CC methods is approximately 85% (plus or minus 4%) of the total accuracy after iterative classification. The local classifier accounts for only 60% of the classification accuracy on the original data set and 54% through the random partitioning method. This means that when only a local classifier, such as Naive Bayes, is used on our partitioned segments, we achieve a higher accuracy without including any of the link structure over the initial dataset.

Even though it appears that our partitioning methods seem to have their benefit because they simply build better local classifiers, this isn't necessarily true. We can see through the accuracy of the Random partitioning that the improved local classification isn't achieved through simply modeling on smaller data sets. However, with our methods which intelligently determine group membership, we see improvement not only in local classification accuracy, but overall classification accuracy. We also, however, gain additional classification accuracy through the use of the collective inference mechanism. By combining the information gain partitioning method, the naive bayes local classifier and the relaxation labeling collective inference algorithm, we achieve the greatest accuracy in our social network classification.

6.6 Conclusions and Future Work

Our research indicates that information gain-based partitioning methods may be used to efficiently divide and then classify the data in a social network. While our research does indicate that for small-scale applications or very few divisions, we may spend more time on calculating the segments if we use a IGBP heuristic. We also show that for a more moderate or high number of processors, any segmentation method saves time for the entire classification.

CHAPTER 7

SOCIAL NETWORK ACCESS CONTROL - A FRAMEWORK

In the three previous chapters, we have shown methods of using data within the social network to improve local and relational classification methods. While the fields of epidemiology and counter-terrorism are generally agreed to be important areas for these tasks. However, a major emergent field is using this data in marketing, by partering with social network providers. This usage has prompted privacy concerns with many users of these social networks.

Here, we provide a framework for giving users far greater control over access to their information within the social network. By utilizing semantic web technologies, we are able to propose very granular controls for managing everything from user photos to who is able to send messages or videos to underage children.

7.1 Introduction

On-line Social Networks (OSNs) are platforms that allow people to publish details about themselves and to connect to other members of the network through links. Recently, the popularity of OSNs is increasing significantly. For example, Facebook now claims to have more than a hundred million active users.[1] The existence of OSNs that include person-specific information creates both interesting opportunities and challenges. For example, social network data could be used for marketing products to the right customers. At the same time, security and privacy concerns can prevent such efforts in practice [5]. Improving the OSN access control systems appears as the first step toward addressing the existing security and privacy concerns related on-line social networks. However, most of current OSNs implement very basic access control systems, by simply making a user able to decide which personal information are accessible by other members by marking a given item as public, private, or accessible by their direct contacts. In order to give more flexibility, some online social networks enforce variants of these settings, but the

[1] http://www.facebook.com/press/info.php?statistics

principle is the same. For instance, besides the basic settings, Bebo (http://bebo.com), Facebook (http://facebook.com), and Multiply (http://multiply.com) support the option "selected friends"; Last.fm (http://last.fm) the option "neighbors" (i.e., the set of users having musical preferences and tastes similar to mine); Facebook, Friendster (http://friendster.com), and Orkut (http://www.orkut.com) the option "friends of friends"; Xing (http://xing.com) the options "contacts of my contacts" (2nd degree contacts), and "3rd" and "4th degree contacts". It is important to note that all these approaches have the advantage of being easy to be implemented, but they lack flexibility. In fact, the available protection settings do not allow users to easily specify their access control requirements, in that they are either too restrictive or too loose. Furthermore, existing solutions are platform-specific and they are hard to be implemented for various different online social networks.

To address some of these limitations, we propose an extensible, fine-grained OSN access control model based on semantic web technologies. Our main idea is to encode social network-related information by means of an ontology. In particular, we suggest to model the following five important aspects of OSNs using semantic web ontologies: (1) user's profiles, (2) relationships among users (e.g., Bob is Alice's close friend), (3) resources (e.g., online photo albums), (4) relationships between users and resources (e.g., Bob is the owner of the photo album), (5) actions (e.g., post a message on someone's wall). By constructing such an ontology, we model the Social Network Knowledge Base (SNKB). The main advantage for using an ontology for modeling OSN data is that relationships among many different social network concepts can be naturally represented using OWL. Furthermore, by using reasoning, many inferences about such relationships could be done automatically. Our access control enforcement mechanism is then implemented by exploiting this knowledge. In particular, the idea is to define security policies as rules (see Section 7.4), whose antecedents state conditions on SNKB, and consequents specify the authorized actions. In particular, we propose to encode the authorizations implied by security policies by means of an ontology, obtaining the Security Authorization Knowledge Base (SAKB). Thus, security policies have to be translated as rules whose antecedents and consequents are expressed on the ontology. To achieve this goal, we use the Semantic Web Rule Language (SWRL) [33]. As consequence, the access control policies can be enforced by simply querying the authorizations, that is, the SAKB. The query can be easily directly implemented by the ontology reasoner by

means of instance checking operations, or can be performed by a SPARQL query, if the ontology is serialized in RDF. In this paper, we focus on how to model such a fine-grained social network access control system using semantic web technologies. We also assume that a centralized reference monitor hosted by the social network manager will enforce the required policies. Since our proposed approach depends on extensible ontologies, it could be easily adapted to various online social networks by modifying the ontologies in our SNKB. Furthermore, as we discuss in details later in the paper, semantic web tools allow us to define more fine grained access control policies than the ones provided by current OSNs.

The paper is organized as follows. In Section 7.2, we provide a brief discussion of current security and privacy research related to online social networks. In Section 7.3, we discuss how to model social networks using semantic web technologies. In Section 7.4, we introduce a high level overview of the security policies we support in our framework. In addition to access control policies, we state filtering policies that allow a user (or one of her supervisors) to customize the content she accesses. We also introduce admin policies, stating who is authorized to specify access control and filtering policies. In Section 7.5, we introduce the authorization ontology and the SWRL rule encoding of security policies. In Section 7.6, we discuss how security policies could be enforced. In Section 7.7, we give an overview of the framework that integrates the previous components. Finally, we conclude the paper in Section 7.8.

7.2 Related work

7.3 Modeling Social Networks Using Semantic Web Technologies

Recently, semantic web technologies such as Resource Description Framework (RDF) and the Web Ontology Language (OWL) have been used for modeling social network data [44]. Although our goal in this paper is not to propose new semantic approaches for modeling online social network data, we would like to give a brief overview of current approaches for the sake of completeness by pointing out also other social network information that could be modeled by semantic technologies. In our discussion, we will use Facebook as a running example. At the same time, we would like to stress that our discussion could be easily extended to other social networking frameworks. In general, we identify five categories of social network data that could

be modeled by semantic technologies. These are: (1) personal information; (2) personal relationships; (3) social network resources; (4) relationships between users and resources; (5) actions that can be performed in a social network. In the following, we discuss how these social network data can be represented.

7.3.1 Modeling Personal Information

Some of the personal information provided on OSNs such as Facebook can be modeled by using the Friend-of-a-Friend ontology (FOAF) [7]. FOAF is an OWL-based format for representing personal information and an individual's social network. FOAF provides various classes and properties to describe social network data such as basic personal information, online account, projects, groups, documents and images. However, these basic mechanisms are not enough to capture all the available information. For example, there is no FOAF construct to capture the meaning for *lookingFor* (e.g., John Smith is looking for friendship). Thanks to the extensibility of the RDF/OWL language, this is easily solvable. For example, consider the following case where we capture the information related to an individual with Facebook Profile Id 999999 using a new Facebook ontology written in the RDF/OWL language.[2] In this example, we assume that "fb" ontology has a property name *lookingFor* to capture the required information.

@prefix rdf: ¡http://www.w3.org/1999/02/22-rdf-syntax-ns¿ .

@prefix foaf: ¡http://xmlns.com/foaf/0.1/¿ .

@prefix fb: ¡http://example.org/facebook¿ .

¡http://www.facebook.com/profile.php?id=999999999¿

foaf:name "John Smith" .

¡http://www.facebook.com/profile.php?id=999999999¿

fb:lookingFor "Friendship" .

As the example suggests, existing ontologies such as FOAF could be easily extended to capture personal information available on online social networks.

[2]We use Turtle notation for representing OWL.

7.3.2 Modeling Personal Relationships

Currently, online social networks do not support fine-grained definitions of relationships. For instance, Facebook allows you to specify whether you attended school or work with a friend, but offers no way to express what that truly means, that is, the strength of the relationship. It is this fine-grained structure that we wish to capture. Mika [44] proposes a reification based model for capturing relationship strength. Instead, to comply with W3C specifications [19], we adopt the use of the n-ary relation pattern rather than use simple statement reification, which is a violation of the specification [20]. If we were to violate the specification, then relationships would be modeled using a series of four RDF statements to create an identifier for the relationship. Unfortunately, as a result of that, SWRL would be unable to understand these relationships. We believe that using a specification-recommended pattern and retaining the ability to use SWRL to do inference on relationships is the best solution.

For the reasons stated above, we choose to model personal relationships using n-ary relation pattern. To comply with n-ary relation specification [20], we define a *FriendshipRelation* class which has subclasses that denote a general strength of friendship. The root *FriendshipRelation* class implies an unspecific friendship while the three subclasses, *Family*, *CloseFriend*, and *DistantFriend*, give an indicator of the closeness between people. The *CloseFriend* subclass has a further extension: *BestFriend*.

This basic structure allows us to easily mimic the existing structure of Facebook relationship types. However, as mentioned previously, these relationship types have no predefined meanings. In order to begin to quantify the meaning of relationship assignments, each instance of *FriendshipRelation* has a data property *TrustValue*. This represents the level of trust that the initiator has with the friend.

As an example suppose that an individual (e.g., John Smith) defines a relationship with a colleague (e.g., Jane Doe). This creates an instance of the *FriendshipRelation* class with the *TrustValue* data property, which represents the level of trust between the initiator and his friend. The instance also has an object property that links it to the instance of the friend. This instance of the *FriendshipRelation* class is then tied back to John Smith through the use of the *Friendship* object property.

It is important to note that any (uni-directional) relationship in the social network is a single instance of the *FriendshipRelation* class. Thus, to model the standard bidirectional nature of social network relations, we need two instances of this class. However, the simple logical inference that if B is a friend of A, then A is a friend of B can not be implemented by SWRL, in that this would imply to create a new instance of the Friendship class. Unfortunately, this is outside the realm of SWRL's capability. So this must be taken care of outside of the SWRL framework by an external application. It is also important to note that the *TrustValue* property of relationships is a value that is computed automatically outside the OWL/SWRL component of the social network. This value is used to do various inference tasks further in the network. At the most basic level, where the *TrustValue* is a static number based on the friendship type, this is a trivial component. We assume that there will be a more complicated formula used in calculating the *TrustValue* that may be beyond the bounds of the built-in mathematical operators of SWRL.

We experience a similar difficulty with indirect relationships. To define an inferred relationship, we would once again need to create a new instance of *FriendshipRelation*. We can, however, create these indirect relationships similar to how we maintain symmetry of relationships, detailed above. The only difference in the indirect relationship is that instead of creating an instance of the class *FriendshipRelation*, we create an instance of a separate class, *InferredRelation*, which has no detailed subclasses, yet is otherwise identical to the FriendshipRelation base class.

7.3.3 Modeling Resources

A typical OSN provides some resources such as Albums or Walls to share information among individuals. Clearly RDF/OWL could be used to capture the fact that Albums are composed of pictures and each picture may have multiple people in it. In our framework, we model resources as a class, beginning with a generic *Resource* class. As subclasses to this, we can have, for example, *PhotoAlbum*, *Photo*, and *Message*. Each of these has specific, unique properties and relationships. For instance, *PhotoAlbum* has a name and a description as data properties and has an object property called *containsPhoto* that links it to instances of *Photo*. These have a name, a caption, and a path to the stored location of the file. Messages have a sender, a receiver, a subject, a message, and a time stamp. We can also create a subclass of Messages called *WallMessage* which

is similar to Messages in that it has the same data properties, but it has additional restrictions such as that a *WallMessage* may only be sent to a single individual.

7.3.4 Modeling User/Resource Relationships

Current applications such as Facebook assume that the only relationship between users and resources is the ownership. However, from an access control point of view this is not enough. Let us consider, for example, a photograph that contains both John Smith and Jane Doe. Jane took the picture and posted it on the social network. Traditionally, Jane would be the sole administrator of that resource. Since the photo contains the image of John (we say that John is tagged to the photo), in our model John may have some determination as to which individuals can see the photo.

To model something like a photo album, we can use two classes. The first is a simple *Photo* class that simply has an optional name and caption of the photo and a required path to the location of the file. A photo is then linked to each person that is listed as being in the photo. A *PhotoAlbum* has a name and a description. *PhotoAlbum* and *Photo* are linked using the *containsPhoto* relationship. The individual owner – the person who uploaded the photos – is indicated by the *ownsAlbum* relationship. Similarly, we can represent other relationships between users and resources.

7.3.5 Modeling Actions

In a social network, actions are the basis of user participation. According to the proposed representation an action is defined as object property that relates users, resources, and actions. Moreover, we model hierarchies for actions by means of subproperty. Take, for instance, three generic actions: *Read, Write, Delete*. We define a hierarchy in which *Delete* is a subtype of *Write* which is, itself, a subtype of *Read*. In a non-hierarchical model, if John Smith was able to read, write, and delete a photo, then we would need three authorizations to represent this property. However, as we have defined the hierarchy, with only the authorizations of ¡“John Smith”, Delete, Photo1¿, John Smith has all three properties allowed.

We can also extend traditional access restrictions to take advantage of social networking extensions. For instance, the action *Post* can be defined as a subtype of Write. So, let us say that we define the actions *Write* to mean that an individual can send a private message to another in-

dividual, and that the action *Post* means that an individual can post a message to another's Wall so that any of their friends can see it. Then allowing a user the *Post* action would allow them to see the friends wall, send them a private message, and write on their wall, but she could not delete anything.

7.3.6 Running Example

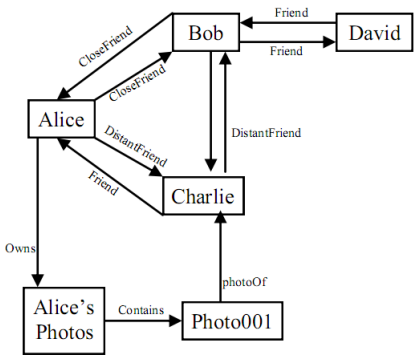


Figure 7.1. A portion of an OSN

In the remainder of this paper, we will use the small network shown in Figure 7.1 to illustrate our access control mechanism. Our running example has four individuals: Alice, Bob, Charlie, and David. Alice, Bob, and Charlie form a clique with different strengths of friendship connecting them. David is a friend only of Bob via the default Friendship type. There is also a *PhotoAlbum* that was uploaded by Alice that contains a single photo that is a picture of Charlie.

7.4 Security policies for OSNs

As evinced by recent work on social network security, protecting resources in social networks requires us to revise traditional access control models and mechanisms. However, the approaches proposed so far have focused only on access control policies, that is, on the problem of regulating the access to OSN resources. We think that this is not enough, in that the complexity of the social network scenario requires the definition of further security policies, besides standard access control policies. In this section, we outline the security policies our framework supports.

Access Control policies. The framework supports access control policies to regulate how resources can be accessed by OSN participants. In particular, the supported access control policies

are defined on the basis of our previous work [12]. Here, authorized users are denoted in terms of the type, depth, and trust level of the relationships existing between nodes in the network. For instance, an access control policy can state that the only OSN participants authorized to access a given resource are those with a direct or indirect friendship relationship with the resource owner, provided that this relationship has a given trust value. However, the access control policies supported by the proposed framework have some notable improvements w.r.t. those presented in [12]. These improvements are mainly due to the fact that our access control policies are defined according to the SNKB described in Section 7.3. This means that the object, subject and privilege of an access control policy are defined exploiting the semantic modeling of resources, users and actions. In particular, as it will be explained in Section 7.5, access control policies are defined as rules over ontologies representing the concepts introduced in Section 7.3. Thus rather than access control policies specified over each single participant and resource of a OSN, we are able to specify access control policies directly on the OSN semantic concepts. Indeed, it is possible to specify a generic access control policy stating that the photos can be accessed only by friends, by simply specifying the *Photo* class as a protected object. As such, the access control policy will be applied to all instances of the *Photo* class, i.e., to all photos, thus greatly simplifying policy administration. Specifying access control policies over semantic concepts has another benefit in that it is possible to exploit the hierarchy defined over the concepts to automatically propagate access control policies. For example, with respect to resources, if *Photo* has been defined with some subclasses, say *PrivatePhoto* and *HolidaysPhoto*, the previous access control policy can be automatically applied to all the instances belonging to any subclass of *Photo*. Access control policies can be also propagated along other dimensions, that is, according to hierarchies specified in the ontologies of other OSN concepts (e.g., ontologies for relationship types and actions). For example, in case the supported relationship ontology defines an hierarchy for the friendship relationship, the previous access control policy is propagated to all OSN participants with which the resource owner has any kind of friendship relationship. A similar propagation arises if the action ontology defines an hierarchy of actions. Note that also in [12], authorized subjects are defined in terms of user relationships rather than by listing specific instances (i.e., person ids). However, in that work policy propagation is not possible, since no hierarchies are defined over resources, relationships and actions. Moreover, the semantic modeling we propose in this paper

makes us able to specify authorized users not only in terms of the relationships they should have with the resource owner (as in [12]), but also in terms of the relationships they should have with the resource. Thus, for example it is possible to specify an access control policy stating that all OSN participants that are tagged to a photo are authorized to access that photo. The only way to specify this access control policy in [12] as well as in all the other existing models for OSNs is to explicitly specify a different access control policy for each OSN participant tagged to the photo.

Filtering policies. In a OSN, users can publish information of very heterogeneous content, ranging from family photos to adult-oriented contents In this sense, the access control issues arising in OSNs are similar to those we have in the web, where the availability of inappropriate information could be harmful for some users (for example, young people). To protect users from inappropriate or unwanted contents, we introduce *filtering policies*, by which it is possible to specify which data has to be filtered out when a given user browses the social network pages. By means of a filtering policy, it is, for example, possible to state that from OSN pages fetched by user Alice, all videos that have not been published by Alice's direct friends have to be removed. Similar to access control policies, filtering policies are defined as rules over ontologies representing the concepts introduced in Section 7.3 (see Section 7.5). This implies that policy propagation is possible also in case of filtering policies. Another relevant aspect of filtering policies is related to the user that specifies the policy (i.e., the grantor). Indeed, in our framework, a filtering policy can be specified in two different ways. According to the first one, a filtering policy is specified by a user to state which information she prefers not to access, i.e., which data has to be filtered out from OSN pages fetched by her. Thus, in this case the grantor and the user to which the policy applies, i.e., the target user, are the same. These policies state user preferences w.r.t. the contents one wants to access and for that reason are called *filtering preferences*. However, we also supports the specification of filtering policies where the target user and the grantor are different. This kind of filtering policies makes the grantor able to specify how the SN pages fetched by target users have to be filtered. By means of these filtering policies, a grantor can *supervise* the content a target user can access. In this case, we refer to the filtering policy as *supervised filtering policy*. This represents an extremely useful feature in open environments like OSNs. For example, a parent can specify a supervised filtering policy stating that her children do not have to access those videos published by users that are not trusted by the parent herself. As it will be more clear later on,

semantic technologies greatly facilitate the specification of this kind of policies.

It is worth noticing that both filtering preferences and supervised filtering policies can not be enforced by simply supporting negative access control policies, that is, policies avoiding access to resources. This is due to the fact that access control policies and filtering policies have totally different semantics. Indeed, an access control policy is specified by the resource owner to state who is authorized or denied to access her resources. Rather, a filtering policy is specified by a supervisor for a target user or by the target user herself, to specify how resources have to be filter out when she fetches an OSN page. Note that, according to the proposed semantics, this filtering takes place even in the case the target user is authorized to access the resource, that is, even if she satisfies the access control policies specified by the resource owner.

Admin policies. Introducing access control and filtering policies in a multi-user environment like OSNs requires to determine who is authorized to specify policies and for which target users and objects. To address this issue we introduce *admin policies*, that make the Security Administrator (SA) of the social network able to state who is authorized to specify access control and filtering policies. Admin policies have to be flexible enough to model some obvious admin strategies that are common to traditional scenarios (e.g., the resource owner is authorized to specify access control policies for her resources) as well as more complex strategies, according to the security and privacy guidelines adopted by the OSN. For instance, the SA could specify an admin policy stating that users tagged to a given resource are authorized to specify access control policies for that resource. Note that, as previously pointed out, the ontology modeling the relationships between users and resources described in Section 7.3 is extremely useful in the specification of such admin policies. Other kinds of admin policies are those related to filtering policies. For instance, by means of an admin policy, a SA could authorize parents to define supervised filtering policies for their young children. This admin policy can be defined by stating that if a user U1 has a relationship of type ParentOf with a user U2, which has age less than 16 (i.e., with the property age less than 16), then U1 can state supervised filtering policies where the target user is U1. The SA could further refine this admin policy to specify that the parents can state supervised filtering policies for their young children only for video resources. This would modify the previous admin policy by limiting the scope of the supervised filtering policy the parents are authorized to specify.

7.5 Security policy specification

A policy language defines security policies according to three main components: a *subject specification* aiming to specify the entity to which a security policy applies (e.g., users, processes), an *object specification* to identify the resources to which the policy refers to (e.g., files, HW resources, relational tables), and an *action specification*, specifying the action (e.g., read, write execute, admin) that subjects can exercise on objects. Moreover, to make easier the task of policy evaluation, policies are enforced through a set of *authorizations*, stating for each subject the rights she has on the protected resources. We encode security policies by means of rules. In general, a rule consists of two formulae and an implication operator, with the obvious meaning that if the first formula, called the antecedent, holds then the second formula, called the consequent, must also hold. Thus, we encode each security policy as a *security rule*, that is, a rule whose antecedent represents the conditions stated in the policy subject and object specifications, and the consequent represents the entailed authorizations. Note that since the framework supports different types of security policies, the security rules could entail different types of authorizations. In particular, if the antecedent of a security rule encodes an access control or admin policy, the consequent denotes the entailed access control or admin authorizations. In contrast, if the rule's antecedent encodes a filtering policy (either a filtering preference or a supervised filtering policy), the consequent entails *prohibitions* rather than authorizations, since this policy limits access to resources.

We adopt SWRL to encode security rules. SWRL has been introduced to extend the axioms provided by OWL to also support rules. In SWRL, the antecedent, called the body, and the consequent, called the head, are defined in terms of OWL classes, properties and individuals. More precisely, they are modeled as positive conjunctions of *atoms*. Atoms can be of the form: (1) C(x), where C is an OWL description or data range; (2) P(x,y), where P is an OWL property and x and y could be variables, OWL individuals or OWL data values; (3) sameAs(x,y); (4) differentFrom(x,y); (5) builtIn(r,x,...), where r is a built-in predicate that takes one or more arguments and evaluates to true if the arguments satisfy the predicate. More precisely, an atom C(x) holds if x is an instance of the class description or data range C, an atom P(x,y) holds if x is related to y by property P, an atom sameAs(x,y) holds if x is interpreted as the same object as y, an atom differentFrom(x,y) holds if x and y are interpreted as different objects, and builtIn(r,x,...) holds if

the built-in relation r holds on the interpretations of the arguments.

Exploiting SWRL to specify security rules implies that authorizations and prohibitions must be represented in some ontology, thus to be encoded as a SWRL head. For this reason, before presenting the encoding of a security policy, we first introduce an ontology to model authorizations and prohibitions. We refer to the knowledge base derived by this ontology as Security Authorization Knowledge Base (SAKB).

7.5.1 Authorizations and Prohibitions

Since the framework supports three different types of security policies, it has to manage three different types of authorizations, namely access control authorizations, admin authorizations, and prohibitions. In the following, we introduce the proposed ontology for their representations. However, it is relevant to notice that this ontology is strictly related to the ontologies supported by the OSN (see Section 7.3), in that it defines authorizations/prohibitions on the basis of the supported actions and resources. As such, the following does not intend to be the standard ontology for SAKBs, rather it is the one that we adopt in our framework, based on the semantic modeling presented in Section 7.3. Thus, the discussion presented here must be read as a guideline for the definition of an ontology of a SAKB.

Access control authorizations. The first kind of authorizations are those entailed by access control policies. In general, an access control authorization can be modeled as a triple (u,p,o) stating that subject u has the right to execute privilege p on object o. Thus, in some way, an access control authorization represents a relationship p between u and o, meaning that u can exercise p on o. Therefore, we decide to encode an access control authorization for privilege p as an instance of an OWL object property, named p, defined between the authorized person and the authorized resource.

To model all possible access control authorizations we have to introduce a different object property for each action supported in the OSN (see Section 7.3). It is interesting to note that by properly defining the object property encoding access control authorizations we can automatically propagate the authorizations on the basis of the classification defined among actions.

Example 7.5.1 *Let us consider, for example, the action Post and assume it has been defined as subclass of action Write.*

In terms of access control, if the post privilege is authorized to a user, then the write privilege is also authorized. In the proposed framework, the access control authorizations can be automatically inferred provided that object property Post has been defined as subproperty of the object property Write. We do note that this hierarchy may be different than in traditional access control systems. When we use SWRL, anything that is defined for a superclass will also be defined for its subclasses. However, the reverse is not true. So, when we allow an individual to Write, it does not automatically confer the Post authority.

Prohibitions. Filtering policies state whether the target user is not authorized to access a certain object, in the case of supervised filtering policies, or she prefers not to access, in the case of filtering preferences. Similarly to access control authorizations, a prohibition specifies a relationship between a user and the resource she is not authorized or she prefers not to access. For this reason, also prohibitions can be expressed as an object property between *Person* and *Resource* classes. More precisely, a prohibition for the Read privilege is defined as the OWL object property *PRead*. An instance of this object property ¡John, URI1¿*:PRead* states that Bob has not to read resource URI1. Similarly, to access control authorizations, it is possible to specify how prohibitions have to be propagated by simply defining subproperty.

Example 7.5.2 *Let us again consider the three basic actions: Read, Write, Delete, and their prohibited versions: PRead, PWrite, PDelete. We again wish to form a hierarchy of actions in a logical order. That is, if an individual is prohibited from Reading a resource, then she should also be prohibited from Writing and Deleting that resource. To do this, we can simply define PRead to be a subtype of PWrite, which is a subtype of PDelete.*

Admin authorizations. Admin authorizations are those authorizations implied by admin policies, which, we recall, have the aim to authorize users to specify access control or filtering policies. Therefore, admin policies entail two types of authorizations: authorizations to specify access control policies, to which we simply refer as *admin authorizations*, and authorizations to specify filtering policies, i.e., *admin prohibitions*. In general, an admin authorization can be represented

as a triple (u,p,o) stating that user u is authorized to specify access control policies for privilege p on object o. Thus, similarly to authorizations and prohibitions also admin authorizations can be expressed as an object property between *Person* and *Resource* classes.

Example 7.5.3 *According to this modeling, we can define the Object property AdminRead, whose instances state that a given user is authorized to express access control policies granting the read privilege on a given object. Consider the instance ¡Bob,URI1¿:AdminRead, which states that Bob is authorized to specify access control policies granting the read privilege on the URI1 object.*

Similarly, to access control authorizations, it is possible to specify how admin authorizations have to be propagated by simply defining subproperty.

Example 7.5.4 *Let us declare the previously mentioned property AdminRead and further create the properties AdminWrite and AdminAll. We declare AdminAll to be a subproperty of both AdminWrite and AdminRead. Consider our running example where Alice owned a photo. Let us assume that this grants her the AdminAll authorization on the photo. If Alice attempts to allow Bob to Read the photo, an action which is restricted to individuals with the AdminRead property, then this is allowed via the AdminAll property.*

In contrast, an admin prohibition can be represented as a tuple (s,t,o,p), which implies that user s (supervisor) is authorized to specify filtering policies for the privilege p applying to the target user t and to object o. Differently from previous authorizations, admin prohibitions can not be represented as properties in that they do not represent a binary relationship. For that reason, we decide to model admin prohibitions as an OWL class *Prohibition*. This makes us able to specify all the components of the prohibition as class properties. More precisely, given an admin prohibition (s,t,o,p), we can model the authorized supervisor has an object property *Supervisor* between the *Prohibition* and *Person* classes. Similarly, the target user can be represented as an object property *TargetUser* between the *Prohibition* and *Person* classes, and the target object as an object property *TargetObject* between the *Prohibition* and the *Resource* classes. In contrast, the privilege over which the supervisor is authorized to state filtering policies is not represented as an object property. Indeed, to automatically propagate admin prohibitions we prefer to specify the privilege directly as the class name. Thus, as an example, instances of the *ProhibitionRead* class

	SWRL rule
(1)	Video(?targetObject,) ∧ ParentOf(Bob,?controlled) ⟹ PRead(?controlled,?targetObject)
(2)	Owner(Bob,?targetObject) ∧ Photo(?targetObject) ∧ Friend(Bob,?targetSubject) ⟹ Read(?targetSubject,?targetObject)
(3)	Photo(?targetObject) ∧ photoOf(Alice,?targetObject) ∧ Friend(Alice,?targetSubject) ⟹ Read(?targetSubject,?targetObject)
(4)	Photo(?targetObject) ∧ Owns(?owner, ?targetObject) ∧ Friend(?owner, ?targetSubject1) ∧ Friend(?targetSubject1, ?targetSubject2) ⟹ Read(?targetSubject2, ?targetObject)

Table 7.1. Examples of SWRL security rules

state admin prohibitions authorizing the specification of filtering policies for the read privilege. By properly defining subclasses it is possible to automatically infer new admin prohibitions.

Example 7.5.5 *Suppose we have a generic Prohibition class with the subclasses PRead and PView. We then create another subclass of each of these as PAll. Suppose we have two individuals, John and Jane, and John is Jane's father. In this scenario, John should be allowed to filter what videos his daughter is able to see. That is, we have a prohibition ("John", "Jane", Video, PAll). Now, for any video and any permission, John can disallow those that he wishes.*

7.5.2 Security Rules

The proposed framework translates each security policy as a SWRL security rule where the antecedent encodes the conditions specified in the policy (e.g., conditions denoting the subject and object specifications), whereas the consequent encodes the implied authorizations or prohibitions. In particular, since we model security rules as SWRL rules, the SWRL body states policy conditions over the SNKB, i.e., conditions on ontologies introduced in Section 7.3, whereas the SWRL head entails new instances of the SAKB, i.e., instances of the ontology introduced in Section 7.5.1. As a consequence, the specification of SWRL security rules is strictly bound to the ontologies supported by the OSN to model SN and SA knowledge bases. This implies that it is not possible to provide a formalization of generic SWRL rules, since these can vary based on the considered ontologies. In contrast, in this section, we aim to present some meaningful examples of possible SWRL security rules defined on top of ontologies adopted in our framework.

We start by considering the admin policy stating that the owner of an object is authorized to specify access control policies for that object. The corresponding SWRL rule defined according to the ontologies presented in the previous sections is the following:

```
Owns(?grantor, ?targetObject)⟹ AdminAll(?grantor, ?targetObject)
```

The evaluation of the above rule has the result of generating a different instance of the object property *AdminAll* for each pairs of user and corresponding owned resource. It is relevant to note that this authorization is propagated according to the ontology modeling the SAKB. Thus, since the framework exploits the one introduced in Section 7.5.1, the above authorization is propagated also to *AdminRead* and *AdminWrite*.

Another meaningful admin policy for a social network is the one stating that if a user is tagged to a photo then she is authorized to specify access control policies for the read privilege on that photo. This can be encoded by means of the following SWRL security rule:

```
Photo(?targetObject) ∧ photoOf(?grantor, ?targetObject)
⟹ AdminRead(?grantor,?targetObject)
```

The above rules are interesting examples stressing how in the proposed framework, it is possible to easily specify admin policies whose implementation in a non semantic-based access control mechanism would require complex policy management. Indeed, providing the OSN with ontologies modeling the relationships between users and resources (e.g., modeling ownership or tagging relationships) makes the SA able to specify admin policies by simply posing conditions on the type of the required relationship. In contrast, enforcing these admin policies in a traditional access control mechanism would require implementing complex policy management functionalities, in that it would be required to first determine all possible relationships between users and resources then to specify admin authorizations for all of them. Rather in the proposed framework this task is performed by the reasoner.

Example 7.5.6 *Table 7.1 presents some examples of SWRL security rules. The first security rule encodes a filtering policy stating that Bob's children can not access videos. Once this rule is*

evaluated, an instance of prohibition for each of Bob's children and video resource is created. In contrast, the second security rule corresponds to an access control policy stated by Bob to limit the read access to his photos only to his direct friend, whereas the third encodes an access control policy specifying that photos where Alice is tagged can be accessed by her direct friends. Finally, the fourth rule specifies that if a person has a photo, then friends of their friends (an indirect relationship) can view that photo.

7.6 Security rule enforcement

Our framework acts like a traditional access control mechanism, where a *reference monitor* evaluates a request by looking for an authorization granting or denying the request. Exploiting this principle in the proposed framework implies retrieving the authorizations/prohibitions by querying the SAKB ontology. Thus, for example, to verify whether a user u is authorized to specify access control policies for the read privilege on object o, it is necessary to verify if the instance *AdminRead(u,o)* is in the ontology, i.e., to perform an instance checking. This implies that before any possible requests evaluation all the SWRL rules encoding security policies have to be evaluated, thus to infer all access control/admin authorizations as well as all prohibitions. For this reason, before policy enforcement it is required to execute a preliminary phase, called *policy refinement*. This phase aims to populate the SAKB with the inferred authorizations/prohibitions, by executing all the SWRL rules encoding security policies.

Once authorizations/prohibitions are inferred, security policy enforcement can be carried out. In particular, access control and filtering policies are evaluated upon an *access request* is submitted, whereas admin policies are evaluated when an *admin request* is submitted. In the following, we present both the request evaluation by showing how the corresponding policies are enforced.

7.6.1 Admin request evaluation

An admin request consists of two pieces of information: the name of the *grantor*, i.e., the user that has submitted the admin request, and the access control or filtering policy the grantor would like to specify, encoded as SWRL rule, that is, the *submitted SWRL*. The submitted SWRL has to be inserted in the system only if there exists an admin authorization in the SAKB for the

grantor. For example, if the submitted rule requires to specify an access control policy for the read privilege on targetObject, then there must exist an instance of ¡grantor, targetObject¿:*Read*. Note that information about the privilege and the targetObject can be retrieved directly from the submitted SWRL. Thus, in order to decide whether the request above can be authorized or not, a possible way is to query the SAKB to retrieve the corresponding admin authorization, if any. If there exists an instance, then the submitted SWRL can be evaluated, otherwise the framework denies to the grantor the admin request. An alternative way is to rewrite the submitted SWRL by adding in its body also condition to verify whether there exists an admin authorization in the SAKB authorizing the specification of the rule. The following example will clarify the underlying idea.

Example 7.6.1 *Let us assume that the system receives the following admin request:* $\{Bob, SWRL_1\}$*, where* $SWRL_1$ *is the following:*

$SWRL_1$: `Owns(Bob,?targetObject)` $\wedge$ `Photo(?targetObject)`

$\wedge$ `Friend(Bob,?targetSubject)` $\Longrightarrow$ `Read(?targetSubject,?targetObject)`

In order to determine the result of the admin request, the framework has to verify the existence of ¡Bob,targetObject¿: AdminRead instance in the SAKB. This check can be incorporated in the body of $SWRL_1$ by simply modifying it as follows:

`New_SWRL`$_1$`:` **AdminRead(Bob,?targetObject)** $\wedge$
`Owns(Bob,?targetObject)` $\wedge$ `Photo(?targetObject)` $\wedge$
`Friend(Bob,?targetSubject)`
$\Longrightarrow$ `Read(?targetSubject, ?targetObject)`

Then New_SWRL_1 is evaluated with the consequence that Read access control authorizations will be inserted in SAKB only if Bob is authorized to specify them by an admin policy.

In case of an admin request submitting a filtering policy, to decide whether the grantor is authorized to specify that policy, a search is required in the *Prohibitions* class (i.e., the subclass corresponding to the action the filtering policy requires to prohibit) for an instance having the property *Grantor* equal to the grantor and the properties *Controlled* and *TargetObject* equal to the controlled and *TargetObject* specified in the head of the submitted SWRL rule, respectively. Also in this case, we can adopt an approach based on SWRL rewriting.

Example 7.6.2 *Let us assume that the framework receives the following admin request: {Bob, $SWRL_2$}, where $SWRL_2$ is the following:*

`SWRL`$_2$`:` `Video(?targetObject)` $\wedge$ `ParentOf(Bob,?controlled)` $\Longrightarrow$
`PRead(?controlled,?targetObject)`

Then, the system can modify the submitted SWRL as:

`New_SWRL`$_2$`:` **PRead(?p)** $\wedge$ **Grantor(?p,Bob)** $\wedge$ **Controlled(?p,?controlled)** $\wedge$ **TargetObject(?p,?targetObject)** $\wedge$ `Video(?targetObject)` $\wedge$ `ParentOf(Bob,?control- led)` $\Longrightarrow$
`PRead(?controlled,?targetObject)`

whose evaluation has the effect to insert instances of PRead property (i.e., read prohibitions) only if there exists an Admin Prohibition (Bob,c,o,Read), where c is Bob's children, and o is a video resource. Note that this is valid also if the submitted SWRL explicitly specifies the name of the controlled user (e.g.,.... $\Longrightarrow$ `PRead(Alice,?targetObject)`*).*

7.6.2 Access request evaluation

In general, an access request can be modeled as a triple (u, p, URI), which means that a user u requests to execute the privilege p on the resource located at URI. To evaluate this request the framework has to verify whether there exists an access control authorization granting p on URI to requester r. However, since the proposed system also supports filtering policies, the presence of such an authorization does not necessarily imply that r is authorized to access URI because there could be a prohibition denying access to the resource to the user. Thus, to evaluate whether an access request has to be granted or denied, it is necessary to perform two queries to the SAKB. The first to retrieve authorizations and the second to retrieve prohibitions. More precisely, if u requires the read privilege $Read$, the system has to query the instances of object property *Read* and *PRead*. In particular, both the queries look for instance ¡u,URI¿ (i.e.,¡u,URI¿:*Read* and ¡u,URI¿:*PRead*). Then, the access is granted if the first query returns an instance and the second returns the empty set. It is denied otherwise.

Example 7.6.3 *Consider again Example 7.5.5, with the addition of a person named "Susan". Susan is a friend of Jane and has posted a video which she allows to be seen by all of her friends. However, Jane's father prohibits her from viewing videos. When a request is made by Jane to see Susan's video, the authorization and the prohibition queries are performed. The authorization query returns a Read permission, but the prohibition query returns a PRead. This means that Jane will be unable to view the video.*

7.7 Framework Architecture

In our proposed framework, we plan to build several layers on top of the existing online social network application. We plan to implement our prototype using Java based open source semantic

web application development framework called JENA [3], since it offers an easy to use programmatic environment for implementing the ideas discussed in this paper. Here, we describe each of these layers independently, as well as the motivation behind choosing specific technologies in our framework. While we use specific instances of Facebook as the over-arching application utilizing the lower level semantic layers, any social network application could be modified to use the design we describe here.

7.7.1 RDF Datastore

We assume the use of a general RDF triple-store to hold the underlying data. In this representation, all facts about an entity are recorded as a triple of the form ¡Subject, Predicate, Object¿. So, suppose we have an individual named John Smith, who is assigned a unique identifier 999999, would give us the tuple ¡999999, *foaf:Name*, "John Smith"¿.

We plan to use a similar format in a separate table to store a list of authorizations so that we do not have to re-infer them each time an authorization is requested. For the data storage system, we plan to use MySQL because of its availability and because of its ease of interface with JENA.

We note here that an RDF datastore differs from a relational database in that there is no database method of ensuring that constraints are maintained on the ontology as a whole, such as making sure that a defined *Person* has a name. The database representation of this fact is no different than the non-essential statement that the person lives in Albuquerque. However, we plan to use OWL-DL statements to define these constraints, and then allow the RDF/OWL engine to enforce the constraints as described below.

7.7.2 Reasoner

Any reasoner that supports SWRL rules can be used to perform the inferences described in this paper. However, we chose SweetRules[4] because it interfaces with JENA and has a rule-based inference engine. This means that we can use both forward and backward chaining in order to improve the efficiency of reasoning for enforcing our access control policies. Forward chaining is

[3] http://jena.sourceforge.net/

[4] http://sweetrules.projects.semwebcentral.org/

an inference method where the engine will begin with the data provided and look at the established inference rules in an attempt to derive further information. This can be used when the system needs to infer permissions on a large scale, such as when a resource is added for the first time.

At this point, there will be a large one-time addition of authorizations to the allowed list of users. However, later, after other friends are added, checking to see if a user has access to a limited number of resources can be done through backward chaining.

Basically, in backward chaining, we begin with the desired goal (e.g., goal is to infer whether "John have permission to see the photo album A"), and check whether it is explicitly stated or whether it could be inferred by some other rules in a recursive fashion. Obviously, this will allow a result to be inferred about an individual (e.g., John) without re-checking all other individuals.

In [44], Mika proposes a basic general social network, called Flink, based on a semantic datastore using a similar framework to that we have proposed, but using several different specific semantic technologies. However, he does specify that their implementation, using backwards- and forward-chaining is efficiently scalable to millions of tuples, which provides an evidence of the viability of our proposed scheme.

7.7.3 RDF/OWL engine

For the RDF/OWL interface, we chose to use the JENA API. We use this to translate the data between the application and the underlying data store. JENA has several important features that were considered in its use. First, it is compatible with SweetRules. Secondly, it supports OWL-DL reasoning which we could use to verify that the data is consistent with the OWL restrictions on the ontology. The OWL restrictions are simple cardinality and domain/range constraints such as every person has to have a name and must belong to at least one network. To enforce these constraints, we plan to have the application layer pass the statements to be entered about an individual until all have been collected. We then have JENA insert these statements into the database and then check the new model for consistency. If there are any constraints that have been violated, then we pass this information back to the social network application and have it gather the required information from the user.

7.8 Conclusions

In this paper, we have proposed an extensible fine grained on-line social network access control model based on semantic web tools. In addition, we propose authorization, admin and filtering policies that are modeled using OWL and SWRL. The architecture of a framework in support of this model has also been presented. We intend to extend this work toward several directions. A first direction arises by the fact that supporting flexible admin policies could bring the system to a scenario where several access control policies specified by distinct users can be applied to the same resource. Indeed, in our framework social network's resources could be related to different users according to the supported ontology. For example, a given photo could be connected to the owner, say Bob, as well as to all users with which is tagged with, say Alice and Carl. According to the semantics of admin policies, it could be the case that some of these tagged users are authorized to specify access control policies for that resource, say only Alice. For example, Alice could have specified that the photos tagged to her can be accessed only by her direct friends, whereas Bob could have specified that his photos have to be accessed by his direct friend and colleagues. In order to enforce access control the framework have to decide how the specified access control policies have to be combined together. As such, a first important extension of the proposed framework will be the support of a variety of policy integration strategies. As a further important future work, we plan to implement our framework using the ideas discussed in Section 7.7 and test the efficiency of various ways of combining forward and backward chaining based reasoning for different scenarios.

CHAPTER 8
SOCIAL NETWORK ACCESS CONTROL - IMPLEMENTATION

In the previous chapter, we provided a framework for social network access control using semantic web technologies. Here, we show the results of an initial implementation of our framework on synthetic data.

8.1 Introduction

On-line Social Networks (OSNs) are platforms that allow people to publish details about themselves and to connect to other members of the network through links. Recently, the popularity of OSNs is increasing significantly. For example, Facebook now claims to have more than a hundred million active users.[1] The existence of OSNs that include person-specific information creates both interesting opportunities and challenges. For example, social network data could be used for marketing products to the right customers. At the same time, security and privacy concerns can prevent such efforts in practice [5]. Improving the OSN access control systems appears as the first step toward addressing the existing security and privacy concerns related to on-line social networks. However, most of current OSNs implement very basic access control systems, by simply making a user able to decide which personal information are accessible by other members by marking a given item as public, private, or accessible by their direct contacts. In order to give more flexibility, some online social networks enforce variants of these settings, but the principle is the same. For instance, besides the basic settings, Bebo (`http://bebo.com`), Facebook (`http://facebook.com`), and Multiply (`http://multiply.com`) support the option "selected friends"; Last.fm (`http://last.fm`) the option "neighbors" (i.e., the set of users having musical preferences and tastes similar to mine); Facebook, Friendster (`http://friendster.com`), and Orkut (`http://www.orkut.com`) the option "friends of friends"; Xing (`http://xing.com`) the options "contacts of my contacts" (2nd degree contacts), and

[1] http://www.facebook.com/press/info.php?statistics

"3rd" and "4th degree contacts". It is important to note that all these approaches have the advantage of being easy to be implemented, but they lack flexibility. In fact, the available protection settings do not allow users to easily specify their access control requirements, in that they are either too restrictive or too loose. Furthermore, existing solutions are platform-specific and they are hard to be implemented for various different online social networks.

To address some of these limitations, we propose an extensible, fine-grained OSN access control model based on semantic web technologies. Our main idea is to encode social network-related information by means of an ontology. In particular, we suggest to model the following five important aspects of OSNs using semantic web ontologies: (1) user's profiles, (2) relationships among users (e.g., Bob is Alice's close friend), (3) resources (e.g., online photo albums), (4) relationships between users and resources (e.g., Bob is the owner of the photo album), (5) actions (e.g., post a message on someone's wall). By constructing such an ontology, we model the Social Network Knowledge Base (SNKB). The main advantage for using an ontology for modeling OSN data is that relationships among many different social network concepts can be naturally represented using OWL. Furthermore, by using reasoning, many inferences about such relationships could be done automatically. Our access control enforcement mechanism is then implemented by exploiting this knowledge. In particular, the idea is to define security policies as rules (see Section 8.2), whose antecedents state conditions on SNKB, and consequents specify the authorized actions. In particular, we propose to encode the authorizations implied by security policies by means of an ontology, obtaining the Security Authorization Knowledge Base (SAKB). Thus, security policies have to be translated as rules whose antecedents and consequents are expressed on the ontology. To achieve this goal, we use the Semantic Web Rule Language (SWRL) [33]. As consequence, the access control policies can be enforced by simply querying the authorizations, that is, the SAKB. The query can be easily directly implemented by the ontology reasoner by means of instance checking operations, or can be performed by a SPARQL query, if the ontology is serialized in RDF. In this paper, we focus on how to model such a fine-grained social network access control system using semantic web technologies. We also assume that a centralized reference monitor hosted by the social network manager will enforce the required policies. Since our proposed approach depends on extensible ontologies, it could be easily adapted to various online social networks by modifying the ontologies in our SNKB. Furthermore, as we discuss in details

later in the paper, semantic web tools allow us to define more fine grained access control policies than the ones provided by current OSNs.

The paper is organized as follows. In Section 8.2, we introduce a high level overview of the security policies we support in our framework. In addition to access control policies, we state filtering policies that allow a user (or one of her supervisors) to customize the content she accesses. We also introduce administration policies, stating who is authorized to specify access control and filtering policies. In Section 8.3, we discuss how security policies could be enforced. In Section 8.4, we give an overview of the architecture we have chosen to integrate the semantic components. In Section 8.5, we describe the synthetic data that we use in our experiments, and in Section 8.6 we discuss and provide the results of experiments using our implementation of semantic web-based access control for social networks. Finally, we conclude the paper in Section 8.7.

8.2 Security in Online Social Networks

For a detailed discussion of the use of semantic technologies in online social networks, please refer to our work in [11]. Here, we will constrain our discussions to those specific topics which impact our implementation of an access control mechanism for resources in an online social network.

In the recent past, Facebook has made significant changes to its method of defining the relationships between friends on the network. Previously, if we had two Facebook users who were friends, John and Jane for instance, either John or Jane could select one of the pre-chosen friendship types that Facebook allowed, where the other friend would be required to confirm or reject this label of their friendship. Following this, any friend of either John or Jane could see the definition that was applied to this friendship.

Now, however, instead of defining link types that are visible to others, each individual has the ability to create meaningful lists. Friends can then be added into as many lists as a user chooses. These lists can then be used to control visibility to status updates, wall posts, etc. However, there is no way to define a hierarchy of these lists. For instance, if one was to create a ‘High School Classmates’ and then a ‘College Classmates’ list, there is no way to create a ‘Classmates’ list, without individually adding each individual person to that third list.

We represent a friendship using the n-ary relation pattern, as specified by the W3C [19]. This means that each friendship is an instance of a class, which we call *FriendshipRelation*. This allows us to maintain separate information about each friendship. Specifically, we maintain a *TrustValue* for each friendship. This allows us to determine a specific strength of a friendship, even when compared to those in the same class.

Our implementation supports access control policies to regulate how resources can be accesed by the members of an online social network. In particular, the supported access control policies are defined on the basis of our previous work [12]. Here, authorized users are denoted in terms of the type and/or trust level of the relationships between nodes in the network. For instance, an access control policy can state that the only OSN participants authorized to access a given resource are those with a direct friendship relationship with the resource owner, as long as the relationship also has a certain trust level.

Note, however, that using semantic reasoning can give some improvements over the capabilities discussed in [12]. This benefit comes from our ability to specify access control policies over semantic concepts in the OSN. For example, as a default, Facebook may specify that photos can only be viewed by direct friends. In the absence of policies defined by individual users, reasoning will default to this general policy.

8.2.1 Filtering Policies

In an OSN, users can publish information of very heterogeneous content, ranging from family photos to adult-oriented contents. In this sense, the access control issues arising in OSNs are similar to those we have in the web, where the availability of inappropriate information could be harmful for some users (for example, young people). To protect users from inappropriate or unwanted contents, we introduce *filtering policies*, by which it is possible to specify which data has to be filtered out when a given user browses the social network pages. By means of a filtering policy, it is, for example, possible to state that from OSN pages fetched by user Alice, all videos that have not been published by Alice's direct friends have to be removed.

Similar to access control policies, filtering policies are defined as rules over ontologies. This implies that policy propagation is possible also in case of filtering policies. Another relevant aspect

of filtering policies is related to the user that specifies the policy (i.e., the grantor). We define two methods where a filtering policy may be created. According to the first one, a filtering policy is specified by a user to state which information she prefers not to access, i.e., which data has to be filtered out from OSN pages fetched by her. Thus, in this case the grantor and the user to which the policy applies, i.e., the target user, are the same. These policies state user preferences w.r.t. the contents one wants to access and for that reason are called *filtering preferences*. However, we also support the specification of filtering policies where the target user and the grantor are different. This kind of filtering policies makes the grantor able to specify how the SN pages fetched by target users have to be filtered. By means of these filtering policies, a grantor can *supervise* the content a target user can access. In this case, we refer to the filtering policy as *supervised filtering policy*. This represents an extremely useful feature in open environments like OSNs. For example, a parent can specify a supervised filtering policy stating that her children do not have to access those videos published by users that are not trusted by the parent herself. As it will be more clear later on, semantic technologies greatly facilitate the specification of this kind of policies.

It is worth noticing that both filtering preferences and supervised filtering policies can not be enforced by simply supporting negative access control policies, that is, policies avoiding access to resources. This is due to the fact that access control policies and filtering policies have totally different semantics. Indeed, an access control policy is specified by the resource owner to state who is authorized or denied to access her resources. Rather, a filtering policy is specified by a supervisor for a target user or by the target user herself, to specify how resources have to be filtered out when she fetches an OSN page. Note that, according to the proposed semantics, this filtering takes place even in the case the target user is authorized to access the resource, that is, even if she satisfies the access control policies specified by the resource owner.

8.3 Security rule enforcement

Our framework acts like a traditional access control mechanism, where a *reference monitor* evaluates a request by looking for an authorization granting or denying the request. Exploiting this principle in the proposed framework implies retrieving the authorizations/prohibitions by querying the SAKB ontology. Thus, for example, to verify whether a user u is authorized to specify

access control policies for the read privilege on object o, it is necessary to verify if the instance *AdminRead(u,o)* is in the ontology, i.e., to perform an instance checking. This implies that before any possible requests evaluation all the SWRL rules encoding security policies have to be evaluated, thus to infer all access control/administrative authorizations as well as all prohibitions. For this reason, before policy enforcement it is required to execute a preliminary phase, called *policy refinement*. This phase aims to populate the SAKB with the inferred authorizations/prohibitions, by executing all the SWRL rules encoding security policies.

Once authorizations/prohibitions are inferred, security policy enforcement can be carried out. In particular, access control and filtering policies are evaluated upon an *access request* being submitted, whereas administrative policies are evaluated when an *administration request* is submitted. We do this because, in general, the number of people given administrative access to an object is far less than the number of individuals who may have access to that same object. In the following, we present both the request evaluation by showing how the corresponding policies are enforced.

8.3.1 Administration request evaluation

An administrative request consists of two pieces of information: the name of the *grantor*, i.e., the user that has submitted the administrative request, and the access control or filtering policy the grantor would like to specify, encoded as SWRL rule, that is, the *submitted SWRL*. The submitted SWRL has to be inserted in the system only if there exists an administrative authorization in the SAKB for the grantor. For example, if the submitted rule requires to specify an access control policy for the read privilege on targetObject, then there must exist an instance of <grantor, targetObject>:*Read*. Note that information about the privilege and the targetObject can be retrieved directly from the submitted SWRL. Thus, in order to decide whether the request above can be authorized or not, a possible way is to query the SAKB to retrieve the corresponding administrative authorization, if any. If there exists an instance, then the submitted SWRL can be evaluated, otherwise the framework denies to the grantor the administrative request. An alternative way is to rewrite the submitted SWRL by adding in its body also condition to verify whether there exists an administrative authorization in the SAKB authorizing the specification of the rule. The following example will clarify the underlying idea.

Example 8.3.1 *Let us assume that the system receives the following administrative request: {Bob, Tag}, where Tag is the following:*

```
Tag:  Owns(Bob,?targetObject) ∧ Photo(?targetObject) ⟹
Tag(?targetSubject,?targetObject)
```

That is, that if Bob is the owner of a specific photo (represented by ?targetobject), then the ?targetSubject should be added to the list of users who is allowed to tag new individuals in the photo. In order to determine the result of the administrative request, the framework has to verify the existence of <Bob,targetObject> : AdminTag instance in the SAKB. That is, a check to ensure that Bob is allowed to convey the Tag privilege on the photograph.

This revised check can be incorporated in the body of Tag by simply modifying it as follows:

```
New_Tag:  AdminRead(Bob,?targetObject) ∧ Photo(?targetObject) ⟹
Tag(?targetSubject, ?targetObject)
```

Then New_Tag is evaluated with the consequence that the individual will have the ability to tag people in the photo only if Bob is authorized to specify them by an administration policy.

8.3.2 Access request evaluation

In general, an access request can be modeled as a triple (u, p, URI), which means that a user u requests to execute the privilege p on the resource located at URI. To evaluate this request the framework has to verify whether there exists an access control authorization granting p on URI to requester r. However, since the proposed system also supports filtering policies, the presence of such an authorization does not necessarily imply that r is authorized to access URI because there could be a prohibition denying access to the resource to the user. Thus, to evaluate whether an access request has to be granted or denied, it is necessary to perform two queries to the SAKB. The first to retrieve authorizations and the second to retrieve prohibitions. More precisely, if u requires the read privilege $Read$, the system has to query the instances of object property *Read*

and *PRead*. In particular, both the queries look for instance $< u, URI >$ (i.e.,$< u, URI >$:*Read* and $< u, URI >$:*PRead*). Then, the access is granted if the first query returns an instance and the second returns the empty set. It is denied otherwise.

8.4 Architecture

In our system, we built several layers on top of a reduced online social network application. We considered the actions of a social network (messages, wall posts, viewing profiles, viewing images, etc.) and examined those that involved the most access requests. For example, if a user, John, was to go to Jane's profile, then in the best case, there is a single check (are John and Jane friends of an appropriate level) on permissions.

However, when you consider an image, which can easily have a dozen people tagged in it, and each of those individuals may specify their own additional constraints to the viewership of that image, then it is easy to see that this will be the more complicated example of permissions inference in a system. In effect, we built our system to test a consistent series of worst-case scenarios to test its ability to handle a testing load.

We implement our prototype using the Java based open source semantic web application development framework called JENA [2], since it offers an easy to use programmatic environment for implementing the ideas discussed in this paper, as well as generally being a framework that is supported by most semantic products currently available. Here, we describe each of these layers independently, as well as the motivation behind choosing specific technologies in our framework. While we use specific instances of Facebook as the over-arching application utilizing the lower level semantic layers, any social network application could potentially be modified to use the design we describe here.

8.4.1 RDF Datastore

We use a general RDF triple-store to hold the underlying data. In this representation, all facts about an entity are recorded as a triple of the form <Subject, Predicate, Object>. So, suppose we

[2]http://jena.sourceforge.net/

have an individual named John Smith, who is assigned a unique identifier 999999999, would give us the tuple <999999999, *foaf:Name*, "John Smith">.

Due to the size of the data set we used for experimentation, we do maintain a separate datastore for each of the following: list of users, security policy for each user, resource descriptions, and each generic class of friend (see 8.5 for specifics on generic classes). While not strictly the desired method of maintaining RDF data, we determined that because of the large size of the data we were using, it was not feasible to store all of this in fewer datastores, due to the computational time that would have been required for even the simplest of queries. Even through the use of database indices, there are still far too many tuples in a single table to be efficient for use. Due to its ease of interface and its availability, we used MySQL as the database engine.

We note here that an RDF datastore differs from a relational database in that there is no database method of ensuring that constraints are maintained on the ontology as a whole, such as making sure that a defined *Person* has a name. Because we condense the actions of a social network, we assume that this is handled programmatically at a higher layer.

8.4.2 Reasoner

Any reasoner that supports SWRL rules can be used to perform the inferences described in this paper. However, we initially chose SweetRules[3] because it interfaces with JENA and has a rule-based inference engine, which meant that we could use both forward and backward chaining in order to improve the efficiency of reasoning for enforcing our access control policies. However, during implementation of our system, difficulties arose through the use of SweetRules (described later in Section 8.6) and, due to its success in other projects, chose to use Pellet[4].

8.4.3 RDF/OWL engine

For the RDF/OWL interface, we chose to use the JENA API. We use this to translate the data between the application and the underlying data store. JENA has several important features that were considered in its use. First, it is compatible with Pellet. Secondly, it supports OWL-DL

[3]http://sweetrules.projects.semwebcentral.org/
[4]http://clarkparsia.com/pellet

reasoning which we could use to verify that the data is consistent with the OWL restrictions on the ontology. The OWL restrictions are simple cardinality and domain/range constraints such as every person has to have a name and must belong to at least one network. To enforce these constraints, we plan to have the application layer pass the statements to be entered about an individual until all have been collected. We then have JENA insert these statements into the database and then check the new model for consistency. If there are any constraints that have been violated, then we pass this information back to the social network application and have it gather the required information from the user.

8.5 Data Generation

As we began our implementation, it was apparent we would need to be able to measure the performance of our implementation on large data sets. Because the size of Facebook (at the time we started the implementation) was approximately 300 million users, we established 350 million as the required number of nodes in our data set, to ensure that our implementation could scale to match the numbers Facebook would soon reach.[5] Unfortunately, there are no publicly available data sets of this size that we could use. Instead, we generated our own data set.

That is, we created a data generator that generates n nodes. We began with creating a group of 50 nodes, and generated 50 edges that were distributed randomly across the 50 nodes in the graph. We then chose a random node, n_i with at least one friend and performed Dijkstra's Algorithm to determine if the subgraph was connected. If the subgraph was not connected, then there were j nodes which were not reachable from n_i. We then chose $0 < r_i \leq j$ to be a number of new edges to create. For each edge, we randomly chose a node from the connected subgraph containing n_i and chose a destination node in the disconnected portion of the subgraph. As long as there was a disconnected portion of the subgraph, we continued generating edges. By performing subgraph-joins in this manner, we are able to generate a social network-like graph while still maintaining randomness and not hand-picking which edges would have links between them.

It is important to note at this point that our datastore records each linked twice. Even though the graph is undirected, there is a link type (used in inference) that is directed. For instance,

[5]It is important to note that at this time, Facebook reports that they have in excess of 450 million users. http://www.facebook.com/press/info.php?statics

while the generic 'Friend' link type is bi-directional, the specific 'BestFriend' link type is not necessarily. To maintain this, we recorded each direction of the link with its associated link type. Once the initial subgraph was complete, we iterated through the nodes to assign each edge a link type. We established three generic link types: Friend, Family, and Co-Worker, and recorded them for each direction of the edge. That is, $(n_i, n_j) \in \mathcal{E}$, if assigned the Friend link type, would have generated the tuples $\{n_i, n_j$, Friend$\}$, and $\{n_j, n_i$,Friend$\}$.

We uniformly chose a generic type for each edge that a node is a member of. After this, we then assigned specific subtypes. For 10% of Friend generic link types, we assigned the specific (uni-directional) link type of bestFriend. That is, in the above example, if n_i declared n_j to be a bestFriend, then the tuple $\{n_i, n_j$, Friend$\}$ would have become $\{n_i, n_j$, bestFriend$\}$. Note that the second tuple would have remained unchanged at this time.

Next, we used a Pareto distribution over the number of defined Family Members to determine how many relationships would be defined as 'ParentOf'. We use a Pareto distribution because while having one or two parents listed is what we may generally think of as reasonable for a child, in today's mixed families, it would not be outside the realm of the believable to have more parents listed for a child. It is also important to note that when a link was defined as ParentOf, its partner tuple was automatically assigned the inverse relationship of ChildOf. That is, suppose that our earlier example was instead, $\{n_i, n_j$, Family$\}$ and $\{n_j, n_i$, Family$\}$. If n_i was determined to be the parent, in a single step, we would have the amended tuples $\{n_i, n_j$,ParentOf$\}$ and $\{n_j, n_i$,ChildOf$\}$.

For the Co-worker generic link type, we did not further define a specific link type.

For each node, we also defined a security policy. For clarity, we defined three security policies, which are chosen uniformly at random:

1. **Strict** – Only BestFriends and Family can view photos of self and any children; their children may not view any videos

2. **Casual** – Anyone can see photos; no restriction on children

3. **ParentStrict** – Anyone can see photos of the parent, only family can see photos of their children; Child cannot see any videos.

We then generated m resources, where m is a random number less than 4.5 million and more than 750,000, which should allow us to model both more active and more passive social networks. A resource could be either a photo or a video. We weighted the probability of a resource being a photo to 75%. We then drew from a uniform distribution between 1 and 25, inclusive, to represent the number of people to 'tag' in a photo. This 'tag' indicated a person appearing in the photo, and we viewed this as having a 'stake' in the individuals in the network who could see the photo.

It is important here to note several things. The first is that while our uniform distribution may not reflect a true probabilistic model of the realities of photos on a social networking site such as Facebook, the inclusion of larger groups having a stake in photos represents a more difficult inference problem for such a site. The second is that we do not restrict the individuals being tagged in a photo to only those friends of the photo owner. Because this functionality exists in current online social networks, we needed to support this type of tagging. This ability can support photos of events such as weddings, where a person taking a photo may only know a subset of individuals in the photo, but people who view the photo can supply more details as to other individuals who were photographed. We also note at this point that our method generates a graph where the average number of friends that a person has is 102 users[6].

8.5.1 Event Generation

We next generated a series of events that will be processed in order to examine the effect of using a semantic web-based reasoner for social network access control. We condensed the full set of actions that can be performed in a social network (such as games, posting on walls, changing one's status, etc.) to those that would most strongly affect the ability of a reasoner: Adding friends, Accessing existing resources, creating new resources, and changing permissions. When creating users, adding friends, or creating new resources we performed the task as described previously. However, when we accessed an existing resource, a randomly chosen user attempted to access a randomly selected resource.

[6] As of April 1, 2010, Facebook reports an average of 130 friends

8.6 Experiments

We performed two independent implementations of reasoners. Our first implementation relied upon the SweetRules inference engine. We attempted to perform inference on the entire dataset. Performing inference in this way took 17 hours to load the initial model into memory, and then several seconds to perform each specific reasoning request. However, we noticed that when our model needed to be updated (through new resources, friends, or a change in security policy) these changes were not reflected in our in-memory model. This caused inference done later to become more and more incorrect.

Because of this, we implemented a reasoning solution using Pellet, which has become a very popular reasoner. Initially, we simply changed the reasoner that we used to Pellet. However, in the initial loading step with Pellet, we received out of memory errors and were unable to procede to the reasoning segment.

We then decided that we needed to implement some type of partitioning scheme for the social network. A naïve approach of partitioning would have resulted in some friendship links that spanned partitions. These cross-partition edges would have resulted in one of two things:

1. Reconstructing partitions – Suppose that a partition, P_i, has an user u_a with n friends, u_1, ..., u_n who are stored in partitions $P_1, ...P_n$, respectively. Remember that we can determine who the friends of u_a are from our *Friends* datastore. This ability, however, does not assist us in determining which specific partition their friends are in without an additional index. We must then devote considerable resources to recombining specific partitions in order to be able to effectively infer access permissions.

2. Ignoring links – The other option is that we can simply ignore any friendship links that lead to another partition. This is clearly not a viable option, because it will obviously result in far too many invalid access requests.

We then realized that for any individual access request, there are only groups of users whose security policies and friends are important to determining the success or failure of the request: 1) the resource owner, 2) the person making the access request, and 3) those individuals tagged in the resource. So, we adopt an amended partitioning scheme. For any access request, we generate

a temporary table that contains only the members of the three groups mentioned above, all links involving those individuals, the requestor, and the security policies of all these people.

We then use three methods of measuring trust values and perform experiments to determine how each type affects the time required to perform inference:

- Link Type Only (LTO) is the method described above, where there is only a link type for each edge.
- Trust Value Only (TVO) is a method in which we use only a trust value in place of most link types. Note, however, that because of the unique constraints held by a 'ParentOf'/'ChildOf' relationship, we do still maintain only this link type. We do not maintain other generic or specific link types here, however. We assume that the trust value (which would be assigned by a user) are more specific measures than a defined link type.
- Value/Trust Hybrid (VTH) is an approach where we retain all generic link type declarations, the 'ParentOf' specific type, and add on top of this a Trust Value. This provides for a finer granularity in a security policy. For instance, instead of just being a BestFriend, a specific user can define various security policies, such as only accessible to 'Friend's with a trust value greater than 7, where they may declare a BestFriend to be a trust value of 6 or higher. This allows the user to restrict even among best friends or family.

The results of our experiments are provided in Table 8.6. This table shows the average amount of time that it takes to perform inference tasks, as well as the longest time taken and the shortest time taken. Note that this includes the time required to generate the temporary table that is required for the request to be evaluated.

Additionally, we ran an additional series of tests to determine the effect of the number of friends in a graph, as shown in Figure 8.1. For these tests, we generated a subgraph where each person had a specific number of friends, ranging between 50 and 150. We repeated tests using our three types of trust measurement (LTO, TVO, and VTH) and report the average time taken for inference. Again, we see that LTO clearly takes the least time for inference. However, additionally, we can see that there is only a slight increase in the time taken for inference at each additional group of friends. Further, there is only a slight decrease at the point where all members have

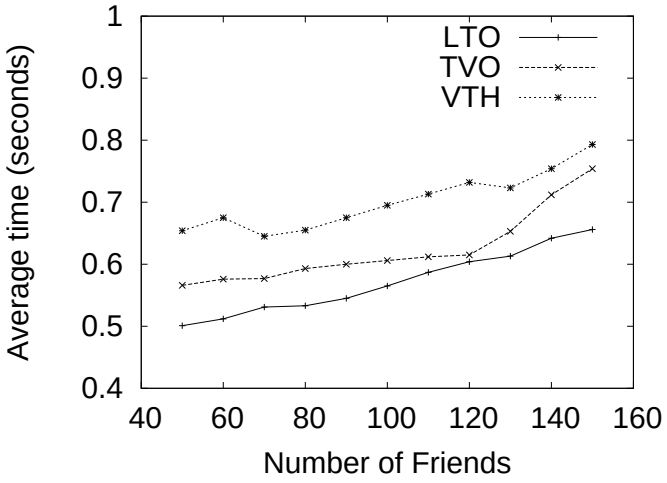


Figure 8.1. Average time for inference by number of friends

50 friends, which indicates that most of the time for our inference operation is consumed by the overhead of the inference engine, including our dynamic partitioning method.

	Average	Low	High
LTO	0.585	0.562	0.611
TVO	0.612	0.534	0.598
VTH	0.731	0.643	0.811

Table 8.1. Time to conduct inference (in seconds)

8.7 Conclusions

In this paper, we have proposed an extensible fine grained on-line social network access control model based on semantic web tools. In addition, we propose authorization, administration and filtering policies that are modeled using OWL and SWRL. The architecture of a framework in support of this model has also been presented. Further, we have implemented a version of this framework and presented experimental results for the length of time access control can be evaluated using this scheme. Further work could be conducted in the area of determining a minimal set of access policies that could be used in evaluating access requests in a further attempt to increase the efficiency of these requests.

Additionally, we have shown that existing social networks need some form of reasonable data partitioning in order for semantic inference of their access control to be reasonable in its speed

and memory requirements, due to constraints on the memory available to perform inference. Additionally, further work can be used in determining the best method of representing the individual information of a person in a social network to determine if a hybrid semantic/relational approach or a pure approach offers the best overall system.

8.8 Acknowledgments

This work was partially supported by National Science Foundation Grants Career-0845803, CNS-0716424, and 0742477 as well as Air Force Office of Scientific Research MURI Grant FA9550-08-1-0265. Additionally, work reported in this paper is partially funded by the Italian Ministry of University, Education, and Research under the ANONIMO project (PRIN-2007F9437X 004).

CHAPTER 9

SANITIZATION OF SOCIAL NETWORK DATA FOR RELEASE TO SEMI-TRUSTED THIRD PARTIES

9.1 Learning Methods on Social Networks

9.1.1 Naïve Bayes on Friendship Links

Consider the problem of determining the class detail value of person n_i given their friendship links using a Naïve Bayes model. That is, of calculating $P(C_x|\mathcal{N}_i)$. Because there are relatively few people in the training set that have a friendship link to n_i, the calculations for $P(C_x|F_{i,j})$ become extremely inaccurate. Instead, we choose to decompose this relationship. Rather than having a link from person n_i to n_j, we instead consider the probability of having a link from n_i to someone with n_j's details. Thus,

$$\begin{aligned} P(C_x|F_{i,j}) &\approx P(C_x|L_1, L_2, \ldots, L_m) \\ &\approx \frac{P(C_x) \times P(L_1|C_x) \times \ldots \times P(L_m|C_x)}{P(L_1, \ldots, L_m)} \end{aligned} \tag{9.1}$$

where L_n represents a link to someone with detail ID_n.

9.1.2 Weighing Friendships

There is one last step to calculating $P(C_x|\mathcal{N}_i)$. In the specific case of social networks, two friends can be anything from acquaintances to close friends or family members. While there are many ways to weigh friendship links, the method we used is very easy to calculate and is based on the assumption that the more public details two people share, the more private details they are likely to share. This gives the following formula for $W_{i,j}$, which represents the weight of a friendship link from n_i to node n_j:

$$W_{i,j} = \frac{|(D_i^1, \ldots, D_i^n) \cap (D_j^1, \ldots, D_j^m)|}{|D_i|} \tag{9.2}$$

Equation 9.2 calculates the total number of details n_i and n_j share divided by the number of details of n_i.

Note that the weight of a friendship link is not the same for both people on each side of a friendship link. In other words, $W_{j,i} \neq W_{i,j}$. The final formula for person i becomes the following, where Z represents a normalization constant and $P(C_x|F_{i,j})$ is calculated by equation 9.1.

$$\rho(C_x, \mathcal{N}_i) = \frac{1}{Z} \sum_{n_j \in \mathcal{N}_i} [P(C_x|F_{i,j}) \times W_{i,j}] \tag{9.3}$$

The value $\rho(C_x, \mathcal{N}_i)$ is used as our approximation to $P(C_x|\mathcal{N}_i)$

9.1.3 Network Classification

Collective Inference is a method of classifying social network data using a combination of node details and connecting links in the social graph. Each of these classifiers consists of three components: a Local classifier, a relational classifier, and a Collective Inference algorithm.

Local classifiers

Local classifiers are a type of learning method that are applied in the initial step of collective inference. Typically, it is a classification technique that examines details of a node and constructs a classification scheme based on the details that it finds there. For instance, the Naïve Bayes classifier we discussed previously is a standard example of Bayes classification. This classifier builds a model based on the details of nodes in the training set. It then applies this model to nodes in the testing set to classify them.

Relational classifiers

The relational classifier is a separate type of learning algorithm that looks at the link structure of the graph, and uses the labels of nodes in the training set to develop a model which it uses to classify the nodes in the test set. Specifically, in [42], Macskassy and Provost examine four relational classifiers: Class-Distribution Relational Neighbor (cdRN), Weighted-Vote Relational

Neighbor (wvRN), Network-only Bayes Classifier(nBC), and Network-only Link-based Classification (nLB).

The **cdRN** classifier begins by determining a reference vector for each class. That is, for each class, C_x, cdRN develops a vector RV_x which is a description of what a node that is of type C_x tends to connect to. Specifically, $RV_x(a)$ is an average value for how often a node of class C_x has a link to a node of class C_a. To classify node n_i, the algorithm builds a class vector, CV_i, where $CV_i(a)$ is a count of how often n_i has a link to a node of class C_a. The class probabilities are calculated by comparing CV_i to RV_x for all classes C_x.

The **nBC** classifier uses Bayes Theorem to classify based only on the link structure of a node. That is, it defines

$$\begin{aligned} P(n_i = C_x|\mathcal{N}_i) &= \frac{P(\mathcal{N}_i|n_i = C_x) \times P(n_i = C_x)}{P(\mathcal{N}_i)} \\ &= \prod_{n_j \in \mathcal{N}_i} \frac{P(n_j = C_a|n_i = C_x) \times P(n_i = C_x)}{P(n_j)} \end{aligned}$$

where $\mathcal{N}_i$ are the neighbors of n_i, and then uses these probabilities to classify n_i.

The **nLB** classifier collects the labels of the neighboring nodes and by means of logistic regression, uses these vectors to build a model.

In the **wvRN** relational classifier, to classify a node n_i, each of its neighbors, n_j, is given a weight. The probability of n_i being in class C_x is the weighted mean of the class probabilities of n_i's neighbors. That is,

$$P(n_i = C_x|\mathcal{N}_i) = \frac{1}{Z} \sum_{n_j \in \mathcal{N}_i} [w_{i,j} \times P(n_j = C_x)]$$

where $\mathcal{N}_i$ is the set of neighbors of n_i and $w_{i,j}$ is a link weight parameter given to the wvRN classifier. For our experiments, we assume that all link weights are 1.

Collective Inference methods

Unfortunately, there are issues with each of the methods described above. Local classifiers consider only the details of the node it is classifying. Conversely, relational classifiers consider only the link structure of a node. Specifically, a major problem with relational classifiers is that while

we may cleverly divide fully labeled test sets so that we ensure every node is connected to at least one node in the training set, real-world data may not satisfy this strict requirement. If this requirement is not met, then relational classification will be unable to classify nodes which have no neighbors in the training set. Collective Inference attempts to make up for these deficiencies by using both local and relational classifiers in a precise manner to attempt to increase the classification accuracy of nodes in the network. By using a local classifier in the first iteration, collective inference ensures that every node will have an initial probabilistic classification, referred to as a *prior*. The algorithm then uses a relational classifier to re-classify nodes. At each of these steps $i > 2$, the relational classifier uses the fully-labeled graph from step $i - 1$ to classify each node in the graph.

The Collective Inference method also controls the length of time the algorithm runs. Some algorithms specify a number of iterations to run, while others converge after a general length of time. We choose to use Relaxation Labeling as described in [42]: a method which retains the uncertainty of our classified labels. That is, at each step i, the algorithm uses the probability estimates, not a single classified label, from step $i - 1$ to calculate new probability estimates. Further, to account for the possibility that there may not be a convergence, there is a decay rate, called α set to 0.99 that discounts the weight of each subsequent iteration compared to the previous iterations. We chose to use Relaxation labeling because in the experiments conducted by Macskassy and Provost [42], Relaxation Labeling tended to be the best of the three collective inference methods.

Each of these classifiers, including a Relaxation Labeling implementation, is included in NetKit-SRL[1]. As such, after we perform our sanitization techniques, we allow NetKit to classify the nodes to examine the effectiveness of our approaches.

9.2 Hiding Private Information

In this section, we first give the operating definition of privacy for a social network. Next, we discuss how to preserve this privacy through manipulation of both details and links.

[1] Available at: http://netkit-srl.sourceforge.net/

9.2.1 Formal Privacy Definition

Problem 1 *Given a graph, $\mathcal{G}$, from a social network, where $\mathcal{I}$ is a subset of $\mathcal{H}$, and $|\mathcal{I}| \geq 1$, is it possible to minimize the classification accuracy on $\mathcal{I}$ when using some set of classifiers $\mathcal{C}$ while preserving the utility of $\mathcal{H}$ - $\mathcal{I}$?*

Definition 12 *Background knowledge, $\mathcal{K}$, is some data that is not necessarily directly related to the social network, but that can be obtained through various means by an attacker.*

Examples of background knowledge include voter registration, election results, phone book results, etc.

Definition 13 *Utility of a graph,*

$$\mathcal{U}(\mathcal{G}) = \sum_{ID_x \in (\mathcal{H}-\mathcal{I})} \left[\max_{c \in C} \left(\mathcal{P}_{c(\mathcal{G})}(ID_x) \right) \right] \tag{9.4}$$

where $\mathcal{H}$ - $\mathcal{I}$ are all non-sensitive detail types, $\mathcal{P}_{c(\mathcal{G})}$ is the accuracy of a classification task using a given classifier c on all arbitrary detail values for a given detail type.

Definition 14 *A graph is $(\Delta, \mathcal{C}, \mathcal{G}, \mathcal{K})$-private if, for a given set of classifiers $\mathcal{C}$,*

$$\max \left[\left(\max_{c \in \mathcal{C}} \left(\mathcal{P}_{c(\mathcal{G},\mathcal{K})} \right) - \max_{c' \in \mathcal{C}} \left(\mathcal{P}_{c'(\mathcal{K})} \right) \right), 0 \right] = \Delta$$

That is, if we have any set of given classifiers, $\mathcal{C}$, then the classification accuracy of any arbitrary classifier $c' \in \mathcal{C}$ when trained on $\mathcal{K}$ and used to classify $\mathcal{G}$ to predict sensitive hidden data is denoted by $\mathcal{P}_{c'(\mathcal{K})}$. Similarly, $\mathcal{P}_{c(\mathcal{G},\mathcal{K})}$ denotes the prediction accuracy of the classifier that is trained on both $\mathcal{G}$and $\mathcal{K}$. Here Δ denotes the additional accuracy gained by the attacker using $\mathcal{G}$. Ideally, if $\Delta = 0$, this means that the attacker does not gain additional accuracy in predicting sensitive hidden data.

9.2.2 Manipulating details

Clearly, details can be manipulated in three ways: adding details to nodes, modifying existing details and removing details from nodes. However, we can broadly classify these three methods

into two categories: perturbation and anonymization. Adding and modifying details can both be considered methods of perturbation – that is, introducing various types of "noise" into $\mathcal{D}$ in order to decrease classification accuracies. Removing nodes, however, can be considered an anonymization method.

Consider, for instance, the difference in two graphs, $\mathcal{G}$' and $\mathcal{G}$", which are sanitized versions of $\mathcal{G}$ by perturbation and anonymization methods, respectively. In $\mathcal{G}$', there are artificial details within $\mathcal{D}$'. That is, suppose that there is a node $n_i \in \mathcal{G}$, $\mathcal{G}$', $\mathcal{G}$" which has a listed detail of (favorite activities, sports) in our two sanitized data sets. When we consider this instance in $\mathcal{G}$', we are uncertain about its authenticity. Depending on the perturbation method used, the original node could have had no favorite activities, or had an entry of (favorite activities, dallas cowboys) which was altered to contain the aforementioned detail.

However, in $\mathcal{G}$", if we see the detail (favorite activities, sports) then we know the detail was in the original data set by the same node. If the detail does not occur in the sanitized data set, then we have no information about the user's preference for sports. All we know is that it was not listed.

Due to these considerations, our first solution is to attempt to remove details in order to decrease classification accuracy on sensitive attributes. We follow this with an examination of perturbation methods.

Problem 2 *Given $\mathcal{G}$ and a non-zero set of sensitive details $\mathcal{I}$, determine the set of details $D' \subset \mathcal{D}$, where $\mathcal{G}$' = $\{\mathcal{V}, \mathcal{E}, \mathcal{D}$ - $D'\}$ has the most reduction in classification accuracy for some set of classifiers $\mathcal{C}$ on the sensitive attributes $\mathcal{I}$ for the given number of removals m.*

Assume a person, n_i, has the class value C_2 out of the set of classes C, and this person has public details D_i.

$$argmax_y[P(C_y) * P(D_i^1|C_y) * ... * P(D_i^m|C_y)] \tag{9.5}$$

Equation 9.5 identifies the learned class. Because we globally remove the most representative details, we are able to find this based off of the equation

$$argmax_y[\forall C_x \in C : P(D_y|C_x)] \tag{9.6}$$

This allows us to find the single detail that is the most highly indicative of a class and remove it. Experimentally, we later show that this method of determining which details to remove provides a good method of detail selection.

9.2.3 Manipulating Link Information

The other option for anonymizing social networks is altering links. Unlike details, there are only two methods of altering the link structure: adding or removing links.

For the same reasons given in section 9.2.2, we choose to evaluate the effects of privacy on removing friendship links instead of adding fake links.

Consider equation 9.3 for determining detail type using friendship links. Also assume that there are two possible classes for a node, and the true class is C_1. We want to remove links that increase the likelihood of the node being in class C_1. Please note that we define a node to be in class C_2 if formula 9.7 is positive.

$$\beta(i) = \rho(C_2, \mathcal{N}_i) - \rho(C_1, \mathcal{N}_i) \tag{9.7}$$

Therefore, we would like to maximize the value of $\beta(i)$ as much as possible by removing links.

Define $\beta_j(i)$ as the new value for formula 9.7 if we remove friendship link $F_{i,j}$. We can compute $\beta_j(i)$ as

$$\begin{aligned}\beta_j(i) &= \left(\rho(C_2, \mathcal{N}_i) - \frac{P(C_2|F_{i,j}) * W_{j,i}}{Z}\right) \\ &\quad - \left(\rho(C_1, \mathcal{N}_i) - \frac{P(C_1|F_{i,j}) * W_{j,i}}{Z}\right) \\ &= \beta(i) + \frac{(P(C_1|F_{i,j,)} - P(C_2|F_{i,j})) * W_{j,i}}{Z}\end{aligned} \tag{9.8}$$

Because $\beta(i)$ and Z are constants for all $\beta_j(i)$, the best choice for i that maximizes $\beta_j(i)$ becomes one that maximizes $M_j = (P(C_1|F_{i,j}) - P(C_2|F_{i,j})) * W_{j,i}$.

In our experiments, we order the links for each node based on the M_j values. When we remove links, we remove those with the greatest M_j values.

9.2.4 Detail Anonymization

In order to combat this type of attack on privacy, we attempt to provide detail anonymization for social networks. By doing this, we believe that we will be able to reduce the value of $\Delta_{\mathcal{G},\mathcal{K}}(C)$ to an acceptable threshhold value that matches the desired utility/privacy tradeoff for a release of data.

Definition 15 *A detail generalization hierarchy (DGH) is an anonymization technique that generates a hierarchical ordering of the details expressed within a given category. The resulting hierarchy is structured as a tree, but the generalization scheme guarantees that all values substituted will be an ancestor, and thus at a maximum may be only as specific as the detail the user initially defined.*

To clarify, this means that if a user inputs a favorite activity as the Boston Celtics, we could have, as an example, the following DGH:

$$\text{Boston Celtics} \rightarrow \text{NBA} \rightarrow \text{Basketball}$$

This means that to completely anonymize the entry of "Boston Celtics" in a user's details, we replace it with "Basketball." However, notice that we also have the option of maintaining a bit more specificity (and therefore utility) by replacing it instead with "NBA." This heirarchical nature will allow us to programmatically determine a more efficient release anonymization, which hopefully ensures that we have a generalized network that is as near-optimal as possible. Our scheme's guarantee, however, ensures that at no time will the value "Boston Celtics" be replaced with the value "Los Angeles Lakers."

Alternately, we have some details, such as "Favorite Music" which do not easily allow themselves to be placed in a hierarchy. Instead, we perform detail value decomposition on these details.

Definition 16 *Detail Value Decomposition (DVD) is a process by which an attribute is divided into a series of representative tags. These tags do not necessarily reassemble into a unique match to the original attribute.*

Thus, we can decompose a group such as "Enya" into {ambient, alternative, irish, new age, celtic} to describe the group.

To generate the DGH and the DVD, we use subject authorities which, through having large amounts of data, are able to provide either a DGH or a DVD for some particular (set of) detail types.

9.3 Experiments

9.3.1 Experimental Setup

In our experiments, we implemented four algorithms to predict the political affiliation of each user. The first algorithm is called "Details Only." This algorithm uses Equation 3.8 to predict political affiliation and ignores friendship links. The second algorithm is called "Links Only." This algorithm uses Equation 9.3 to predict political affiliation using friendship links and does not consider the details of a person. The third algorithm is called "Average." The Average algorithm predicts a node's class value based on the following equation:

$$P_A(N_i = C_a) = 0.5 * P_D(n_i = C_a) + 0.5 * P_L(n_i = C_a)$$

where P_D and P_L are the numerical probabilities assigned by the Details Only and Links Only algorithms, respectively. The final algorithm is a traditional Naïve Bayes classifier, which we used as a basis of comparison for our proposed algorithms.

We define two classification tasks. The first is that we wish to determine whether an individual is politically 'conservative' or 'liberal'. The second classification task is to determine whether an individual is 'heterosexual' or 'homosexual'. It is important to note that we consider individuals who would also be considered 'bisexual' as 'homosexual' for this experiment. We begin by pruning the total graph of 160,000 nodes down to only those nodes for which we have a recorded political affiliation or sexual organization in order to have reasonable tests for the accuracy of our classifiers and the impact of our sanitization. This reduces our overall set size to 35,000 nodes for our political affiliation tests and to 69,000 nodes for our sexual orientation tests. We then conduct a series of experiments where we remove a number of details and a separate series of

Detail Name	Detail Value	Weight
group member	legalize same sex marriage	46.160667
group member	every time i find out a cute boy is conservative a little part of me dies	39.685994
group member	equal rights for gays	33.837868
favorite music	ani difranco	17.36825
favorite movies	sicko	17.280959

Table 9.1. A sample of the most liberal detail values

Detail Name	Detail Value	Weight
group member	george w bush is my homeboy	45.88831329
group member	bears for bush	30.86484689
group member	kerry is a fairy	28.50250433
favorite movies	end of the spear	14.53703765

Table 9.2. A sample of the most conservative detail values

experiments where we remove a number of links. We conduct these removing up to 20 details and links, respectively. The results of these initial tests are shown in Figures 9.1 and 9.2, respectively.

9.3.2 Local Classification Results

Tables 9.1 and 9.2 list the most liberal or conservative details. For example the most liberal detail, as shown in table 9.1, is being a member of the group "legalize same sex marriage". Tables 9.3 and 9.4 show the most liberal and conservative details for each detail type.

In Table 9.5, we show the details that most indicate the 'homosexual' classification. In contrast to political affiliation, there are no single details which are very highly correlated with that classification. For example, the three details we've selected here are more highly indicative of being 'Liberal' than of being 'homosexual'. Conversely, we see in Table 9.6 that there are a few categories that are very highly representative of the 'heterosexual' classification.

As can be seen from the results, our methods are generally successful at reducing the accuracy of classification tasks. Figure 9.1 show that removing the details most highly connected with a class is accurate across the Details and Average classifiers. Counter-intuitively, perhaps, is that the accuracy of our Links classifier is also decreased as we remove details. However, as discussed

Detail Name	Detail Value	Weight
activities	college republicans	5.846955271
favorite books	redeeming love	6.348153362
favorite movies	end of the spear	14.53703765
favorite music	delirious	18.85227471
favorite tv shows	fox news	7.753312932
grad school	sw seminary	2.749648395
group member	george w bush is my homeboy	45.88831329
interests	hunting and fishing	7.614995442
relationship status	married	1.667495517
religious views	christian	2.441063037
sex	male	1.087798286

Table 9.3. Most conservative detail value for each detail

Detail Name	Detail Value	Weight
activities	amnesty international	4.659100601
favorite books	middlesex	4.841749269
favorite movies	hedwig and the angry inc	24.80050378
favorite music	deerhoof	22.94603913
favorite tv shows	queer as folk	9.762900035
grad school	computer science	1.698146579
group member	legalize same sex marriage	46.16066789
interests	vegetarianism	11.76878725
looking for	whatever i can get	1.703651985
relationship status	in an open relationship	1.617950632
religious views	agnostic	3.15756412
sex	female	1.103484182

Table 9.4. Most liberal detail values for each detail

Detail Name	Detail Value	Weight
group member	legalize same sex marriage	1.004393825
group member	equal rights for gays	1.000573463
relationship status	it's complicated	1.005384899

Table 9.5. Most homoexual detail value for each detail

Detail Name	Detail Value	Weight
favorite books	the bible	24.58371923
group member	one man, one woman	45.3918239
relationship status	married	53.84381923

Table 9.6. Most heterosexual detail value for each detail

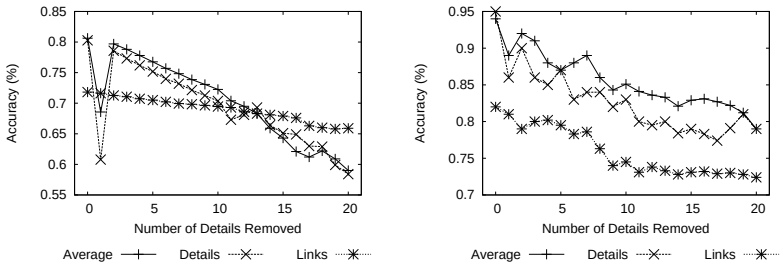
in Section 9.2.3, the details of two nodes are compared to find a similarity. As we remove details from the network, the set of 'similar' nodes to any given node will also change. This can account for the decrease in accuracy of the Links classifier.

Additionally, we see that in Figure 9.1(a) there is a severe drop in the classification accuracy after the removal of a single detail. However, when looking at the data, this can be explained by the removal of a detail that is very indicative of the 'Conservative' class value. When we remove this detail, the probability of being 'Conservative' drastically decreases, which leads to a higher number of incorrect classifications. When we remove the second detail, which has a similar likelihood for the 'Liberal' classification, then the class value probabilities begin to trend downward at a much smoother rate.

While we do not see this behavior in 9.1(b), we do see a much more volatile classification accuracy. This appears to be as a result of the wider class size disparity in the underlying data. Because approximately 95% of the available nodes are 'heterosexual' and there are not details that are as highly indicative of sexual orientation as there are of political affiliation, even minor changes can affect the classification accuracy in unpredictable ways. For instance, when we remove five details, we have lowered the classification accuracy, but for the sixth and seventh details, we see an increase in classification accuracy. Then, we again see another decrease in accuracy when we remove the eighth detail.

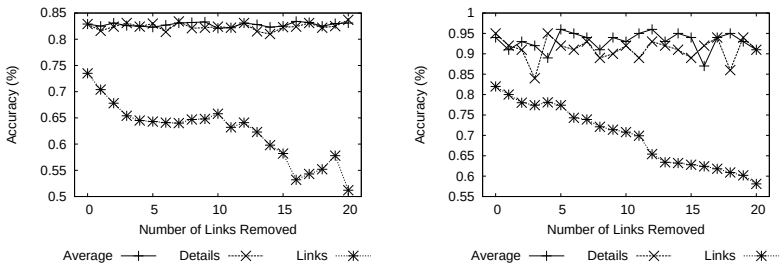
When we remove links, we have a generally more stable downward trend, with only a few exceptions in the 'Political Affiliation' experiments.

While each measure provides a decrease in classification accuracy, we also test what happens in our data set if we remove both details and links. To do this, we conduct further experiments where we test classification accuracy after removing 0 details and 0 links (the baseline accuracy), 0 details and 10 links, 10 details and 0 links, and 10 details and 10 links. We choose these numbers because after removing 12 links, we found that we were beginning to create a number of isolated groups of few nodes or single, disconnected nodes. Additionally, when we removed 13 details, 44% of our 'Political affiliation' data set and 33% of our 'Sexual Orientation' data set had fewer than four details remaining. Since part of our goal was to maintain utility after a potential data release, we chose to remove fewer details and links to support this.



(a) Political Affiliation (b) Sexual Orientation

Figure 9.1. Local classification accuracy by number of details removed



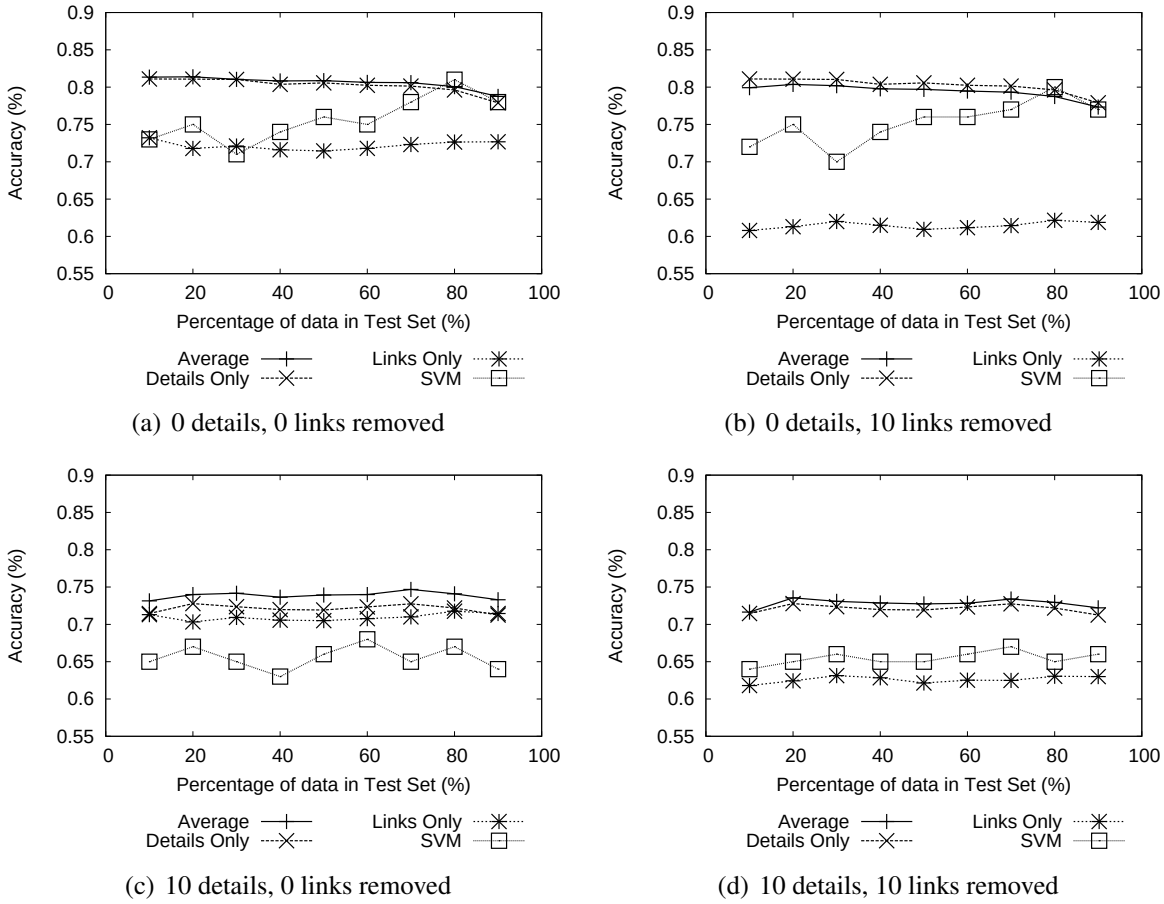
(a) Political Affiliation (b) Sexual Orientation

Figure 9.2. Local classification accuracy by number of links removed

We refer to these sets as 0 details, 0 links; 10 details, 0 links; 0 details, 10 links; 10 details, 10 links removed, respectively. Following this, we want to gauge the accuracy of the classifiers for various ratios of labeled vs. unlabeled graphs. To do this, we collect a list of all of the available nodes, as discussed above. We then obtain a random permutation of this list using the Java function built-in to the *Collections* class. Next, we divide the list into a test set and a training set, based on the desired ratio.

We focus on multiples of 10 for the accuracy percentages, so we generate sets of 10/90, 20/80, . . . , 90/10. Additionally, when creating training sets for our 'Sexual Orientation' data set, because of the wide difference in the group size for 'heterosexual' and 'homosexual', we make sure that we separate out the chosen percentage from the known 'heterosexual' and 'homosexual' groups independently in order to make sure that we have a diversity in both our training and test sets. For example, in a test where we will have only 10% labeled data, we select 10% of heterosexual individuals and 10% of homosexual individuals independently to be in our training set.

We refer to each set by the percentage of data in the test set. We generate five test sets of each ratio, and run each experiment independently. We then take the average of each of these runs.



(a) 0 details, 0 links removed

(b) 0 details, 10 links removed

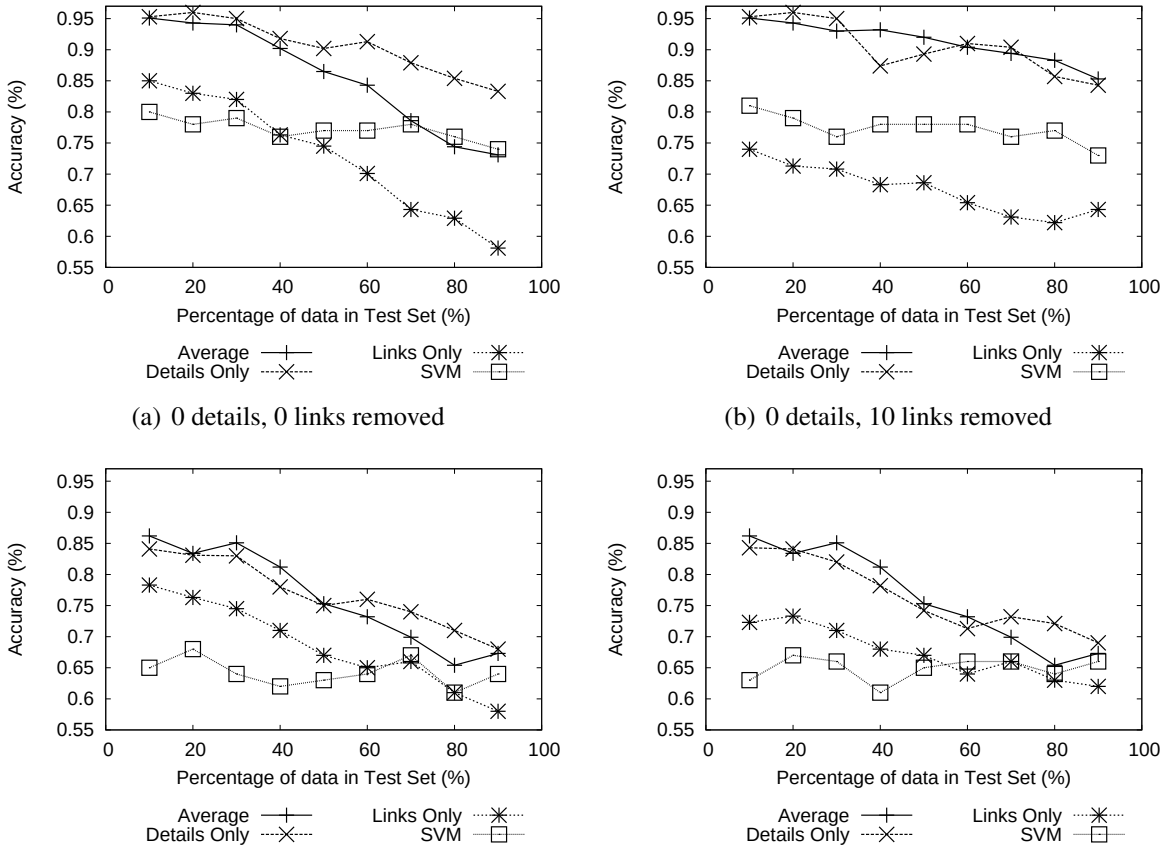
(c) 10 details, 0 links removed

(d) 10 details, 10 links removed

Figure 9.3. Local classifier prediction accuracies by percentage of nodes in test set for Political Affiliation

Our results, as shown in Figure 9.3, indicate that the Average Only algorithm substantially outperformed traditional Naïve Bayes and the Links Only algorithm. Additionally, the Average Only algorithm generally performed better than the Details Only algorithm with the exception of the (0 details, 10 links) experiments.

Also, as a verification of expected results, the Details Only classification accuracy only decreased significantly when we removed details from nodes, while the (0 details, *) accuracies are approximately equivalent. Similarly, the Link Only accuracies were mostly affected by the removal of links between nodes, while the (*, 0 links) points of interest are approximately equal. The difference in accuracy between (0 details, 0 links) and (10 details, 0 links) can be accounted for by the weighting portion of the Links Only calculations, which depends on the similarity between two nodes.



(a) 0 details, 0 links removed

(b) 0 details, 10 links removed

(c) 10 details, 0 links removed

(d) 10 details, 10 links removed

Figure 9.4. Local classifier prediction accuracies by percentage of nodes in test set for Sexual Orientation

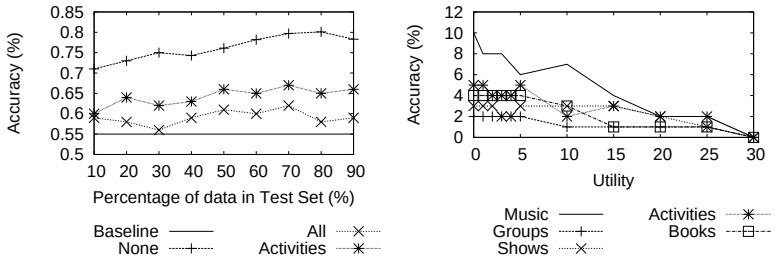
Figures 9.3 and 9.4 show the results of our classification methods for various labeled node ratios. These results indicate that the Average and Details classifiers generally perform at approximately the same accuracy level. The Links Only classifier, however, generally performs significantly worse except in the case where 10 details and no links are removed. In this situation, all three classifiers perform similarly. We see that the greatest variance occurs when we remove details alone. It may be unexpected that the Links Only classifier has such varied accuracies as a result of removing details, but since our calculation of probabilities for that classifier uses a measure of similarity between people, the removal of details may affect that classifier.

Additionally, Figures 9.3 and 9.4 also show the result of using SVMs as a classification technique. We see here that when we remove no details, the classification accuracy of the SVM has a classification accuracy between our Links Only and Average/Details Only classifiers, with the exception of sets where the graph has a large percentage of unknowns (80 and 90% of the graph is unknown) where the SVM classifier can actually outperform the Details Only/Average classifier. However, once we remove details (Figures 9.3(d) and 9.4(d)), the classification accuracy of the SVM drops much further than the Average/Details Only classifier, and even performs worse than the Links Only classification method.

Next, we examine the effects of removing the links. We remove K links from each node, where $K \in [0, 10]$, and again partition the nodes into a test set and training set of equal size. We then test the accuracy of the local classifier on this test set. We repeat this five times and then take the average of each accuracy for the overall accuracy of each classifier after K links are removed. For $K \in [1, 6]$, each link removal steadily decreases the accuracy of the classifier. Removing the seventh link has no noticeable effect, and subsequent removals only slightly increase the accuracy of the Links Only classifier. Also, due to space limitations, for the remainder of experiments we show only the results of the Average classifier, as it is generally the best of the three classifiers.

When we again examine the performance of the SVM, we see similar results to what was seen with Details Only and Average. Since the SVM does not include the link structure in its classification, there is no real affect from removing links on this classification method.

It is important to note that, as we can see from Figure 9.4, the Sexual Orientation classifier seems to be more susceptible to problems of incomplete knowledge. We can see in each subfigure, to a far greater degree than in Figure 9.3, that as we decrease the amount of information available



(a) Effect of generalization on utility (b) Determining Anonymization

Figure 9.5. Generalization results

to the training method the Sexual Orientation classifier accuracy decreases considerably. Once again, we believe that this may be explained simply by the fact that there is far less support of the 'homosexual' classification, and as such, is considerably harder to classify on without adequate data. Specifically, since there are so few instances of the 'homosexual' classification in our data set, when you combine this with the fact that there are no absolute predictors of homosexuality and that the indicators for homosexuality have a very low increased likelihood, if most of the examples of homosexuals are unknown, then classifiers are going to be unable to create an accurate model for prediction.

9.3.3 Generalization Experiments

Each of those details fell into one of several categories: Religion, Political Affiliation, Activities, Books, Music, Quotations, Shows/Movies, and Groups. Due to the lack of a reliable subject authority, Quotations were discarded from all experiments. To generate the DGH for each Activity, Book, and Show/Movie, we used Google Directories. To generate the DVD for Music, we used the Last.fm tagging system. To generate the hierarchy for Groups, we used the classification criteria from the Facebook page of that group.

To account for the free-form tagging that Last.fm allows, we also store the popularity for each tag that a particular detail has. This font size is representative of how many users across the system have defined that particular tag for the music type. We then keep a list of tag recurrence (weighted by strength) for each user. For Music anonymization, we eliminate the lowest-scoring tags.

In our experiments, we assume that the trait "political affiliation" is a sensitive attribute and $\mathcal{C}$ includes a naive Bayes classifier and the implementation of SVM from Weka.

In Figure 9.5(a), we present some initial findings from our domain generalization. We present a comparison of simply using $\mathcal{K}$ to guess the most populated class from background knowledge, the result of generalizing all trait types, generalizing no trait types, and when we generalize the best single performing trait type (activities).

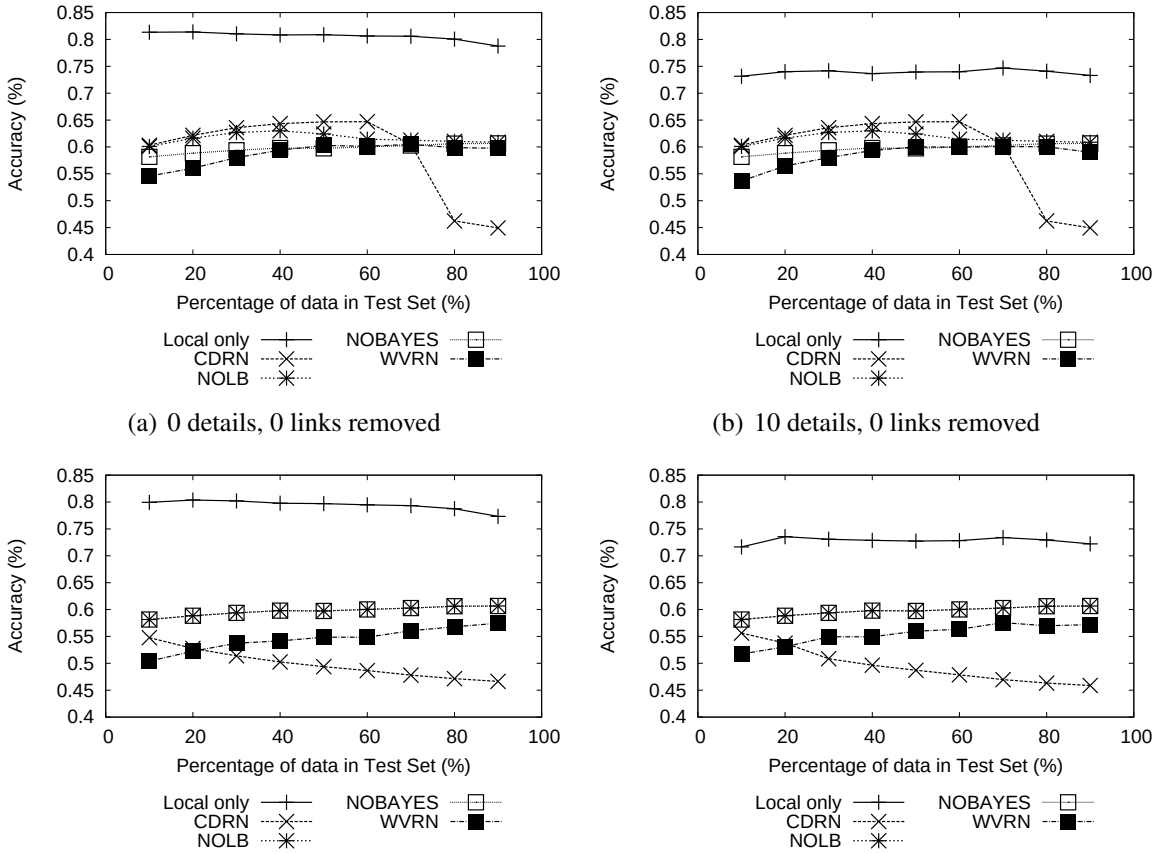
We see here that our method of generalization (seen through the All and Activities lines) do indeed decrease the accuracy of classification on the data set. Additionally, our ability to classify on **non-sensitive attributes (such as gender) are affected by only 2-3%** over the course of the experiments (figure omitted for space). Interestingly, while previous work [39] indicates that Group membership is the dominant detail in classification, we see the most benefit here from generalizing only the Activities detail.

Next, we show that given a desired utility increase, we are able to determine what level to anonymize the dataset to. We see from Figure 9.5(b) that as we allow more utility gain from our anonymized graph, fewer categories are generalized to any degree. We also see that Groups is most consistently anonymized completely until the allowed utility gain is 20 percent. Further, we see that the most variable entry is music.

This may be because the nature of the music detail is that it allows us more easily to include or remove details in order to fit a required utility value. Rather than, say, the activities detail type, which has a fixed hierarchy, music has a loosely-collected group of tags, which we can more flexibly include.

9.3.4 Collective Inference Results

We note that in the Facebook data, there are a limited number of 'groups' that are highly indicative of an individual's political affiliation. When removing details, these are the first that are removed. We assume that conducting the collective inference classifiers after removing only one detail may generate results that are specific for the particular detail we classify for. For that reason, we continue to consider only the removal of 0 details and 10 details, the other lowest point on the classification accuracy. We also continue to consider the removal of 0 links and 10 links due to the marginal difference between the $[6, 7]$ region and removing 10 links.



(a) 0 details, 0 links removed

(b) 10 details, 0 links removed

(c) 0 details, 10 links removed

(d) 10 details, 10 links removed

Figure 9.6. Prediction accuracy of Relaxation Labeling using the Average local classifier (political affiliation)

For the experiments using relaxation labeling, we took the same varied ratio sets generated for the local classifiers. For each, we store the predictions made by the Details Only, Links Only, and Average classifiers and use those as the priors for the NetKit toolkit. For each of those priors, we test the final accuracy of the cdRN, wvRN, nLB, and nBC classifiers. We do this for each of the five sets generated for each of the four points of interest. We then take the average of their accuracies for the final accuracy.

Figure 9.6 shows the results of our experiments using Relaxation Labeling. [42] study the effects of collective inference on four real-world datasets: IMDB, CORA, WebKB, and SEC filings. While they do not discuss the difference in the local classifier and iterative classification steps of their experiments, their experiments indicate that Relaxation Labeling almost always performs better than merely predicting the most frequent class. Generally, it performs at near

80% accuracy, which is an increase of approximately 30% in their datasets. However, in our experiments, Relaxation Labeling typically performed no more than approximately 5% better than predicting the majority class for political affiliation. This is also substantially less accurate than using only our local classifier. We believe that this performance is at least partially because our data set is not densely connected.

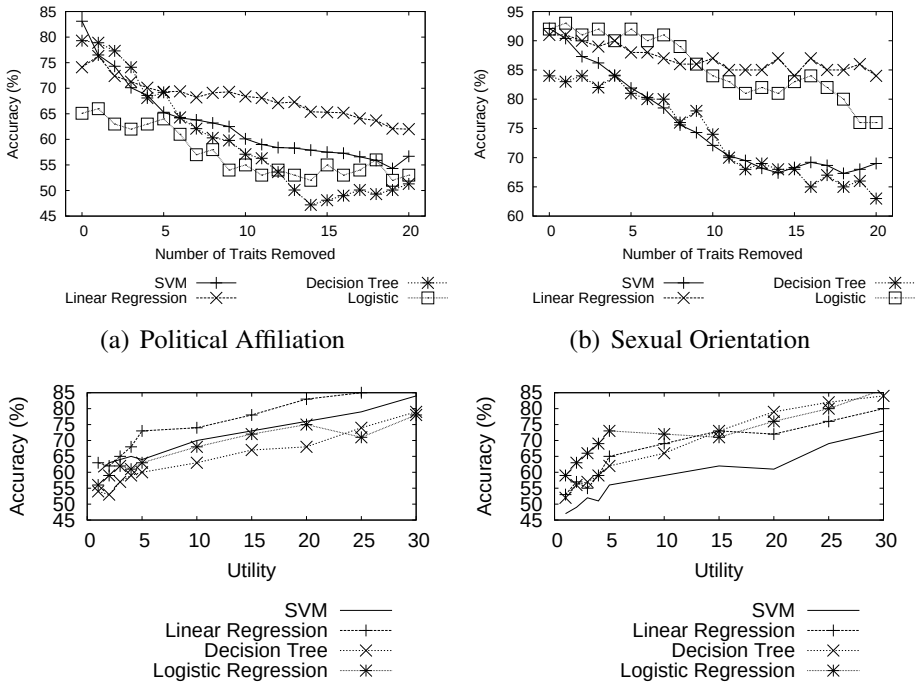
Our results in Figure 9.6 indicate that there is very little significant difference in the collective inference classifiers except for cdRN, which performs significantly worse on data sets where there is a small training set. These results also indicate that our Average classifier consistently outperforms relaxation labeling on the pre- and post-anonymized data sets.

Additionally, if we compare Figures 9.6(a) and 9.6(b) and Figures 9.6(c) and 9.6(d), we see that while the local classifier's accuracy is directly affected by the removal of details and/or links, this relationship is not shown by using relaxation labeling with the local classifiers as a prior. For each pair of the figures mentioned, the relational classifier portion of the graph remains constant, only the local classifier accuracy changes. From these, we see that the most 'anonymous' graph, meaning the graph structure that has the lowest predictive accuracy, is achieved when we remove both details and links from the graph.

9.3.5 Effect of Sanitization on Other Attack Techniques

We further test the removal of details as an anonymization technique by using a variety of different classification algorithms to test the effectiveness of our method. For each number of details removed, we began by removing the indicated number of details in accordance with the method as described in Section 9.2. We then performed 10-fold cross validation on this set 100 times, and conduct this for 0 – 20 details removed. The results of these tests are shown in Figures 9.7(a) and 9.7(b). As can be seen from these figures, our technique is effective at reducing the classification of networks for those details which we have classified as sensitive.

While the specific accuracy reduction is varied by the number of details removed and by the specific algorithm used for classification, we see that we do in fact reduce the accuracy across a broad range of classifiers. We see that Linear regression is affected the least, with approximately



(a) Political Affiliation (b) Sexual Orientation

(c) Political Affiliation (generalization) (d) Sexual Orientation (generalization)

Figure 9.7. Classification accuracy using non-Bayesian methods

a 10% reduction in accuracy. Also that Decision Trees are affected the most, with a roughly 35% reduction in classification accuracy.

This indicates that through the use of a Bayesian classifier, which makes it easier to identify the individual details that make a class label more likely, we can decrease the accuracy of a far larger set of classifiers.

We also see similar results with our generalization method in Figures 9.7(c) and 9.7(d). While the specific value of utility which was defined for Naive Bayes does not exactly hold, we still see that by performing generalization, we are able to decrease classification accuracy across multiple types of classifier.

9.3.6 Effect of Sanitization on Utility

Of course, if the data is to be used by a company, then there must still be some value in the post-sanitization graph. To gauge the utility of the anonymized data set, in Table 9.7 we show the results of various inference tests performed on non-sensitive details. For these tests we used an

SVM and our Bayesian classifiers, as discussed earlier, to run inference and collective inference tasks on each selected detail. We performed each test with 50% of the data in a training set and 50% in a test set. Here, we report the accuracy from running the experiments on random test sets 50 times each from the best-performing classifiers, which were the Average Bayesian with nLB. The accuracies were averaged for display here. The Post-sanitization figures were performed after removing ten details and ten links. Please note that this table represents the accuracy of a classifier on these details, **not** what percentage of the graph has this detail. We tested a selection of details with multiple attributes. For example, the "like video games" detail value is specified specifically in the data set (in Favorite Activities). College-educated was specified as a level of education (scanned for type of degree and school). "Like to read" and "like sports" were inferred from the existence of books in the "favorite book" category or the existence of a sports team in the "favorite activities" category.

It is important to note, obviously, that when we perform inference on details such as "likes to read" we do not consider any detail of the type "favorite book = *". These are discarded for the tests on that type of classification task.

Further, the test sets had a wide variety of representative sizes. "Like to read" had 148,000 profiles, while "like video games" had only 30,000. As we can see from these results, the sanitization has minimal impact on the accuracy of a classifier on non-sensitive details. In fact, for the "College Educated" and "Like video games" details, the sanitization method improved classification accuracies by a small percentage. The apparent reason for this is that the details that are representative of non-sensitive attributes and those that are representative of our sensitive attributes are very disjoint. Recall from Table 9.1 that the group "legalize same sex marriage" is highly indicative that a member is liberal. However, this doesn't translate to any of the tested details. Instead, groups like "1 pwn j00 n h4l0" are indicative of video game players, "i'm taking up money to buy SEC refs glasses" is indicative of sports fans, etc.

It should be noted, however, that the attribute "favorite book = the bible" was removed from this test set, as it is highly indicative of one being a conservative.

Task	Pre-sanitization	Post-sanitization
Like to read	84.3	82.9
Like sports	71.7	70.3
College-educated	73.1	75.2
Like video games	86.1	88.9

Table 9.7. Post-release utility of $\mathcal{G}$ (Classification accuracy)

9.4 Conclusion and Future Work

We addressed various issues related to private information leakage in social networks. For unmodified social network graphs, we show that using details alone, one can predict class values more accurately than using friendship links alone. We further show that using both friendship links and details together gives better predictability than details alone. In addition, we explored the effect of removing details and links in preventing sensitive information leakage. In the process, we discovered situations in which collective inferencing does not improve on using a simple local classification method to identify nodes. When we combine the results from the collective inference implications with the individual results, we begin to see that removing details and friendship links together is the best way to reduce classifier accuracy. This is probably infeasible in maintaining the use of social networks. However, we also show that by removing only details, we greatly reduce the accuracy of local classifiers, which give us the maximum accuracy that we were able to achieve through any combination of classifiers.

Further, effort should be given to converting this to a method by which a social network provider can protect against multiple attribute leakage, rather than a single attribute. We also assumed full use of the graph information when deciding which details to hide. Useful research could be done on how individuals with limited access to the network could pick which details to hide. Similarly, future work could be conducted in identifying key nodes of the graph structure to see if removing or altering these nodes can decrease information leakage.

CHAPTER 10
CONCLUSION

This dissertation has provided the first full-scale analysis of social networks with a focus on privacy. We have provided several major insights that contribute to the overall concept. First, it is possible to leverage even minor data within a network in order to determine hidden information, even with a low initial support. Second, this finding indicates that it is necessary to provide increased control over the data that is contained within one of these networks. There are two considerations for this control: access to the data when using the network as it's intended, and determining hidden information from a potential data release.

Initially, we show that based upon simple, pre-existing relational classifiers, we are able to use specific information about link types that had previously been ignored and use this to increase our ability to determine hidden information. By altering these classifiers, and discovering how to improve their classification ability through the use of this data, we can determine more hidden information than previously expected. Next, we show that through the use of indirect links, we are able to find members of the social network who have a high relative importance to their subregion, and because we propagate this importance to other members of their region, we increase the overall classification accuracy of that network. Finally, we show that by selectively partitioning the network, we can reduce the size of the model that a local classifier is built on; however, through this process, we are able to eliminate excess data within that group and increase the overall classification accuracy of the graph.

Next, we provide a framework for an access control mechanism based upon semantic inference of general, but granular, rules. The motivation for this is that it 1) provides greater control over what rights an individual has to information of theirs appears to another user; 2) control over what right an individual has to representations of their likeness; and 3) greater control over what a minor child is able to do on a social network. Following this, we provide, using synthetic data on a scale of Facebook's current size, a working example using available semantic technologies.

We also provide an analysis of difficulties a provider would face when making these decisions and implementations, yet show that our framework provides a workable model for access control.

Finally, we propose two new methods of preserving privacy in released social network data. We show that each of these methods provides a measure of privacy guarantee for users within the network, but can also be used by third parties for classification on non-sensitive attributes.

REFERENCES

[1] *Naive Bayesian classification of structured data*, volume 57 of *Machine Learning*, pages 233–269. 2004.

[2] Bader Ali, Wilfred Villegas, and Muthucumaru Maheswaran. A trust based approach for protecting user data in social networks. In *2007 Conference of the Center for Advanced Studies on Collaborative research (CASCON'07)*, pages 288–293, 2007.

[3] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*, pages 181–190. ACM, 2007.

[4] Peter S. Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, 110(1):44–91, July 2004.

[5] Stefan Berteau. Facebook's misrepresentation of Beacon's threat to privacy: Tracking users who opt out or are not logged in. CA Security Advisor Research Blog, March 2007. Available at: `http://community.ca.com/blogs/securityadvisor/archive/2007/11/29/facebook-s-misrepresentation-of-beacon-s-threat-to-privacy-tracking-users-who-opt-out-or-are-not-logged-in.aspx`.

[6] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143, New York, NY, USA, 2006. ACM.

[7] Dan Brickley and Libby Miller. FOAF vocabulary specification 0.91. RDF Vocabulary Specification, November 2007. Available at http://xmlns.com/foaf/0.1.

[8] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. Mining hidden community in heterogeneous social networks. In *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, pages 58–65, New York, NY, USA, 2005. ACM.

[9] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. Semantic web-based social network access control. *Computers & Security*, 2010.

[10] B. Carminati, E. Ferrari, and A. Perego. *Security and Privacy in Social Networks*, volume VII of *Encyclopedia of Information Science and Technology*, pages 3369–3376. IGI Publishing, 2 edition, 2008.

[11] Barbara Carminati, Elena Ferrari, Raymond Heatherly, Murat Kantarcioglu, and Bhavani M. Thuraisingham. A semantic web based framework for social network access control. In Barbara Carminati and James Joshi, editors, *SACMAT*, pages 177–186. ACM, 2009.

[12] Barbara Carminati, Elena Ferrari, and Andrea Perego. Enforcing access control in web-based social networks. *ACM Transactions on Information Systems Security*, 13(1), 2009.

[13] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *ACM SIGMOD Record*, volume 27, pages 307–318. ACM, 1998.

[14] D.H. Chau and C. Faloutsos. Fraud detection in electronic auction. In *European Web Mining Forum (EWMF 2005)*, page 87. Citeseer.

[15] Hee-Chul Choi, Sebastian Ryszard Kruk, Sławomir Grzonkowski, Katarzyna Stankiewicz, Brian Davids, and John G. Breslin. Trust models for community-aware identity management. In *Identity, Reference, and the Web Workshop (IRW 2006)*, May 2006. Available at: `http://www.ibiblio.org/hhalpin/irw2006/skruk.pdf`.

[16] NA Christakis and JH Fowler. The spread of obesity in a large social network over 32 years. *New England Journal of Medicine*, 357(4):370–379, July 2007.

[17] NA Christakis and JH Fowler. The collective dynamics of smoking in a large social network. *New England Journal of Medicine*, 358(21):2249–58, May 2008.

[18] Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 281–288. MIT Press, 2006.

[19] World Wide Web Consortium. Status for resource description framework (rdf) model and syntax specification., 1999. Available at: http://www.w3.org/1999/.status/PR-rdf-syntax-19990105/status.

[20] World Wide Web Consortium. Defining n-ary relations on the semantic web, 2006. Available at: http://www.w3.org/TR/swbp-n-aryRelations/.

[21] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. *Internet Computing, IEEE*, 10(4):18–26, 2006.

[22] D. Delen, R. Sharda, and P. Kumar. Movie forecast guru: A web-based dss for hollywood managers. *Decision Support Systems*, 43(4):1151–1170, 2007.

[23] N. Elahi, M.M.R. Chowdhury, and J. Noll. Semantic access control in web based communities. In *Computing in the Global Information Technology, 2008. ICCGI'08. The Third International Multi-Conference on*, pages 131–136, 2008.

[24] Timothy W. Finin, Anupam Joshi, Lalana Kagal, Jianwei Niu, Ravi S. Sandhu, William H. Winsborough, and Bhavani M. Thuraisingham. R OWL BAC: representing role based access control in OWL. In Indrakshi Ray and Ninghui Li, editors, *SACMAT*, pages 73–82. ACM, 2008.

[25] Lise Getoor and Christopher P. Diehl. Link mining: a survey. *SIGKDD Explor. Newsl.*, 7(2):3–12, 2005.

[26] Ralph Gross, Alessandro Acquisti, and John H. Heinz. Information revelation and privacy in online social networks. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80, New York, NY, USA, 2005. ACM Press.

[27] Michael Hay, Gerome Miklau, David Jensen, Philipp Weis, and Siddharth Srivastava. Anonymizing social networks. Technical Report 07-19, University of Massachusetts Amherst, Computer Science Department, March 2007.

[28] Jianming He, Wesley Chu, and Victor Liu. Inferring privacy information from social networks. In Mehrotra, editor, *Proceedings of Intelligence and Security Informatics*, volume LNCS 3975, 2006.

[29] R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. Social network classification incorporating link type values. In *Intelligence and Security Informatics, 2009. ISI'09. IEEE International Conference on*, pages 19–24. IEEE, 2009.

[30] Raymond Heatherly and Murat Kantarcioglu. Extending the classification of social networks. In *Proceedings of the 2011 IEEE international conference on Intelligence and security informatics*, 2011.

[31] Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. Social network classification incorporating link type values. In *Proceedings of the 2009 IEEE international conference on Intelligence and security informatics*, ISI'09, pages 19–24, Piscataway, NJ, USA, 2009. IEEE Press.

[32] Raymond Heatherly, Murat Kantarcioglu, Bhavani Thuraisingham, and Jack Lindamood. Preventing private information inference attacks on social networks. Technical Report UTDCS-03-09, Department of Computer Science, The University of Texas at Dallas.

[33] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium, May 2004. Available at: `http://www.w3.org/Submission/SWRL`.

[34] Matthew O. Jackson. *Social and Economic Networks*, pages 34–43. Princeton University Press, 2008.

[35] D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 593–598. ACM, 2004.

[36] Harvey Jones and Jose Hiram Soltren. Facebook: Threats to privacy. Technical report, Massachusetts Institute of Technology, 2005.

[37] V.E. Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[38] Sebastian Ryszard Kruk, Slawomir Grzonkowski, Adam Gzella, Tomasz Woroniecki, and Hee-Chul Choi. D-FOAF: Distributed identity management with access rights delegation. In Riichiro Mizoguchi, Zhongzhi Shi, and Fausto Giunchiglia, editors, *ASWC*, volume 4185 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2006.

[39] Jack Lindamood, Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. Inferring private information using social network data. In *WWW Poster*, 2009.

[40] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 93–106, New York, NY, USA, 2008. ACM.

[41] Qing Lu and Lise Getoor. Link-based classification. In *20th International Conference on Machine Learning*, Washington, DC, August 2003.

[42] Sofus A. Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8:935–983, 2007.

[43] Aditya Krishna Menon and Charles Elkan. Predicting labels for dyadic data. *Data Min. Knowl. Discov.*, 21:327–343, September 2010.

[44] Peter Mika. Social networks and the semantic web.

[45] Jennifer Neville and David Jensen. Iterative classification in relational data. 2000.

[46] Jennifer Neville and David Jensen. Data mining in social networks. In *National Academy of Sciences Symposium on Dynamic Social Network Analysis*, 2002.

[47] Spiros Papadimitriou and Jimeng Sun. Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *ICDM'08. Eighth IEEE International Conference on Data Mining*, pages 512–521.

[48] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[49] JN Rosenquist, J Murabito, JH Fowler, and NA Christakis. The spread of alcohol consumption behavior in a large social network,. *Annals of Internal Medicine*, 152(7):426–433, April 2010.

[50] Prithviraj Sen and Lise Getoor. Link-based classification. Technical Report CS-TR-4858, University of Maryland, February 2007.

[51] L. Sweeney. Privacy-enhanced linking. *ACM SIGKDD Explorations Newsletter*, 7(2):72–75, 2005.

[52] Ben Tasker, Pieter Abbeel, and Koller Daphne. Discriminative probabilistic models for relational data. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 485–492, San Francisco, CA, 2002. Morgan Kaufmann Publishers.

[53] E. Ted. Link mining applications: Progress and challenges. *ACM SIGKDD Explorations Newsletter*, 7(2):76–83, 2005.

[54] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, rei, and ponder. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 419–437. Springer, 2003.

[55] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. New models in probabilistic information retrieval. Technical Report 5587, British Library, 1980.

[56] Duncan J. Watts and Steven H. Strogatz. Collettive dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.

[57] W. Xu, X. Zhou, and L. Li. Inferring privacy information via social relations. In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 525–530. IEEE, 2008.

[58] Yague, Gallardo, and Mana. Semantic access control model: A formal specification. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 2005.

[59] J.S. Yedidia, W.T. Freeman, and Y. Weiss. *Exploring Artificial Intelligence in the New Millennium*. Science & Technology Books, 2003.

[60] Elena Zheleva and Lise Getoor. Preserving the privacy of sensitive relationships in graph data. In *1st ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD 2007)*, 2007.

[61] Elena Zheleva and Lise Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 531–540, New York, NY, USA, 2009. ACM.

VITA

Raymond Heatherly was born at Langley Air Force Base in Virginia and grew up in Pearl, Mississippi. Following graduation from Pearl High School, Raymond received his Bachelor's of Science with a major in Computer Science and Master of Business Administration from Millsaps College in 2004 and 2005, respectively. In 2006, he entered the Graduate School of the University of Texas at Dallas.