

# **Network Analysis on Incomplete Structures**

by

**Matthew Burgess**

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Computer Science and Engineering)  
in The University of Michigan  
2016

Doctoral Committee:

Associate Professor Eytan Adar, Co-Chair  
Assistant Professor Michael J. Cafarella, Co-Chair  
Associate Professor Carl Lagoze  
Associate Professor Rada Mihalcea  
Associate Professor Qiaozhu Mei

© Matthew Burgess

---

All Rights Reserved

2016

*To my family and friends*

# Acknowledgments

I would first like to extend my gratitude to my thesis committee, Eytan Adar, Michael Cafarella, Rada Mihalcea, Qiaozhu Mei, and Carl Lagoze for their mentorship and construct feedback on this dissertation. Entering a computer science PhD program as a student without a B.S in computer science was both exciting and daunting. I remember taking advanced databases (EECS 584) with Mike Cafarella without having a real understanding of what a database was or even a firm grasp of basic data structures. Despite my ignorance, Mike showed both compassion and guidance in helping to catch me up so I could pass the class. Fast forwarding six years, as one my thesis advisors Mike has always had my back, letting me decide the kinds of projects I want to work on while also offering guidance when I got stuck and support during the hard times of paper rejections. I am especially grateful for his inspiring research vision and for his encouragement to think big.

I am also grateful for the stewardship of my other thesis advisor, Eytan Adar. I was at first intimidated by Eytan's seemingly endless knowledge. I soon got over this intimidation and felt genuinely to have been advised by someone so creative and generous with their knowledge. I admire Eytan for his research acumen but also for his integrity, work ethic, and kindness. I appreciate that Eytan gave me space to come up with my own research projects (though sometimes struggling) while at the same time inspiring me with his own ideas. Looking back, I see that as a result of working under Eytan's supervision, I have grown as both a researcher and engineer. I am and will always be grateful for the wisdom Eytan has shared with me.

I have been lucky enough during my graduate career to have had some amazing mentors outside of Michigan community. I am especially grateful for the mentorship I received at LinkedIn from Pete Skomoroch and Joe Adler. Thanks for introducing me to to the work outside of academia and paving the way for my current career trajectory. I am also very grateful for having been given the chance to be part of the Data Science for Social Good fellowship program. This fellowship came at one of the darker times in my PhD studies and illuminated the path towards the completion of my degree. I am extremely grateful for having had the privilege of working with the other members of the Sunlight team: Eugenia

Giraudy, Julian Katz-Samuels, Lauren Haynes, and Joe Walsh. I have also greatly benefited from the academic community. I would like to thank all of my fellow graduate students Rahul Jha, Dolan Antenucci, Mike Anderson, Sam Carton, Shirley Chen, Zhe Zao, Yue Wang, Xin Rong, Travis Martin, Yongjoo Park, Elaine Wah, Quong Duong, Jesh Bretman, Bryce Weidenback, Ben Casell, Dave Meisner, Frido Lidner and Collin Johnson.

I would not have even applied to graduate school in computer science if it wasn't for the mentorship I received from the theory group at Northwestern. Being my first research mentor, Jason Hartline was one of the first to spark my interest in algorithms and the pursuit of a Phd. I am especially grateful to my dear friend Nicole Immorlica, who has been a constant source of professional wisdom as well as a trusted "squirrel" friend. I would have never thought that the professor who taught me Discrete Math 10 years ago, would today, be one of my most trusted and beloved friends. I look forward to continuing our friendship and to our future wilderness adventures.

I can say without a doubt that If I didn't have the support of my friends and family I would have never been able to finish this dissertation. I am grateful for the support from Ravi Chopra, Kirstin Weiss, Michelle Rzendzian, Victoria Peebles, Matt Stone (Nice Matt), Aston Gonzalez, Anthony Gutierrez, Matt Aslesen (Tall Matt), Cody Andrews, Jeremy Johnson, Levani Papashvili, Roger Grant, Silvia Lindtner, William Ngo, Adam Bonnington, Jean Hardy, Brad Gorman, Stephen Moldrom, James Hunter, Joe Dematio, and Fred Girodeat. I would like to extend a special acknowledgement to my housemates Andrew Deorio, Halley Crissman, and Scott DeOrio. I have enjoyed all of our fabulous dinners and living room disco parties. You three have been such amazing friends and housemates during my time in Ann Arbor, I am so thankful to have you in my life. Thanks Scott for your integral role in organizing both *homo potluck* and *Gay Supper Club (GSC)*; these two institutions brought together my gay family in Ann Arbor. I am especially grateful for my friend Eli Feinmen, for being a sounding board about dissertation woes and for all of the fun dinner parties. I am equally grateful for my friend, Trevor Hoppe, who I have enjoyed numerous dinners, spa days, and vacations with. Thanks for all of the support you have given me and for making life—one rewards point at a time—that much more fabulous. I would also like to thank my friend Andrea Pellegrini for showing me the ropes of grad school and always being there to support me when things were tough.

I am especially grateful for the people that have been in my life from the beginning. Thanks to my Mom and Dad for the constant patience and guidance you have shown me throughout the years. You have always supported and empowered me to make my own decisions, even if they were not decisions you would make yourself. Thanks for teaching me that my "best" is always good enough. My brother Nick—the first doctor in the family— has

always been there setting an example for me. I am so lucky to have a such a trust worthy and loving brother, who has given me so much guidance and always been there when I needed him most. I would also like to thank my other brother (from another mother) Scott. You are one of the best listeners I know, thanks for lending an ear. I would like to thank Rick for all of the love and encouragement he has given me over the past 6 years. My fondest memories of my time in Michigan are of the many trips we took to the great lakes. Thanks for always believing in me, even when I didn't believe in myself. Last, but certainly not least, I would like to thank Oscar the cat, for all of the snuggles and sass.

# Table of Contents

<b>Dedication</b> . . . . .	ii
<b>Acknowledgments</b> . . . . .	iii
<b>List of Figures</b> . . . . .	ix
<b>List of Tables</b> . . . . .	xiii
<b>Chapter 1 Introduction</b> . . . . .	1
1.1 Network Representation of Data . . . . .	2
1.2 Incomplete Networks . . . . .	4
1.3 Dissertation Overview . . . . .	7
1.4 Disseration Organization . . . . .	8
<b>Chapter 2 Literature Review</b> . . . . .	10
2.1 Link Prediction . . . . .	10
2.1.1 Unsupervised Link Prediction . . . . .	11
2.1.2 Supervised Link Prediction . . . . .	13
2.2 Link Prediction for Improving the Network Analysis Pipeline . . . . .	14
2.3 Community Detection Overview . . . . .	16
<b>Chapter 3 Community Detection on Incomplete Networks</b> . . . . .	18
3.1 Problem Overview . . . . .	18
3.2 Detecting Communities in Incomplete Networks . . . . .	20
3.3 Link Prediction for Enhancing Community Detection . . . . .	23
3.4 Link Prediction Enhanced Conensus Clustering . . . . .	26
3.4.1 Network Imputation . . . . .	27
3.4.2 Partition Aggregation . . . . .	28
3.5 Evaluation . . . . .	31
3.5.1 Comparing EDGEBOOST with Different Community Detection Methods . . . . .	31
3.5.2 Varying the Parameters of EDGEBOOST . . . . .	37
3.5.3 Runtime Analysis . . . . .	39
3.6 Applying EDGEBOOST to Survey Generation . . . . .	41

3.6.1	Graph-based Summarization Algorithms . . . . .	43
3.6.2	Combining EDGEBOOST with C-LexRank . . . . .	44
3.6.3	Survey Generation Experiment . . . . .	44
3.7	Discussion and Future Work . . . . .	46
3.8	Related Work . . . . .	48
3.9	Conclusions . . . . .	50
<b>Chapter 4</b>	<b>Using Topically Coherent Communities to Rank Social Feeds . . . . .</b>	<b>51</b>
4.1	Chapter Overview . . . . .	51
4.2	User Scenario . . . . .	53
4.3	System Architecture . . . . .	54
4.4	Algorithms . . . . .	57
4.4.1	List Generation . . . . .	57
4.4.2	Topic Labeling . . . . .	58
4.4.3	Topic Ranking . . . . .	59
4.5	Evaluation . . . . .	60
4.5.1	Dataset Description . . . . .	60
4.5.2	List Generation . . . . .	61
4.5.3	Topic Labeling . . . . .	64
4.5.4	Topic Ranking . . . . .	65
4.5.5	End-to-end Experiment . . . . .	67
4.6	Discussion . . . . .	68
4.7	Previous Work in Social Data Management . . . . .	70
4.8	Conclusion . . . . .	71
<b>Chapter 5</b>	<b>Inferring A Knowledge Graph Heterogenous Data Sources . . . . .</b>	<b>73</b>
5.1	Social Tagging on Expertise Driven Platforms . . . . .	73
5.2	Overview of DOBBY . . . . .	74
5.3	Parent-Child classifier . . . . .	76
5.3.1	Co-occurrence Features . . . . .	76
5.3.2	Network Features . . . . .	77
5.3.3	Wikipedia Distance . . . . .	78
5.4	Evaluating DOBBY on LinkedIn Skills . . . . .	80
5.4.1	Training Data Generation . . . . .	80
5.4.2	Cross Validation and Feature Importance . . . . .	82
5.4.3	Algorithm Comparison . . . . .	82
5.4.4	Human Evaluation . . . . .	84
5.5	Knowledge Graph Structure . . . . .	86
5.5.1	Pruning Reciprocal Edges . . . . .	87
5.6	Previous Research in Social Tagging . . . . .	88
5.7	Discussion and Future Work . . . . .	90
5.8	Conclusion . . . . .	91
<b>Chapter 6</b>	<b>Recovering Hidden Documents from Observed Text . . . . .</b>	<b>92</b>
6.1	Problem Overview . . . . .	93



6.2	Proposed Solution . . . . .	93
6.3	State Legislation Corpus . . . . .	95
6.4	LOBBYBACK Architecture . . . . .	95
6.4.1	Clustering Bills . . . . .	96
6.4.2	Prototype Synthesis . . . . .	97
6.5	Evaluation . . . . .	101
6.5.1	Model Legislation Corpus . . . . .	101
6.5.2	Baselines . . . . .	102
6.5.3	Evaluating Sentence Clusters . . . . .	102
6.6	Discussion . . . . .	104
6.7	Previous Work . . . . .	105
6.8	Conclusion . . . . .	106
<b>Chapter 7</b>	<b>Conclusion . . . . .</b>	<b>107</b>
7.1	Contributions and Future Directions . . . . .	107
7.1.1	Summary . . . . .	110
<b>Bibliography</b>	<b>. . . . .</b>	<b>111</b>

# List of Figures

## Figure

1.1	An example network: nodes represent affiliates of various American revolutionary organizations and edges represent the number of organizations with a shared affiliation. Data provided by [70]. . . . .	3
1.2	Scenario 1: missing edges . . . . .	5
1.3	Scenario 2: missing semantics . . . . .	6
1.4	Scenario 3: missing structure . . . . .	6
1.5	Network analysis pipeline . . . . .	7
1.6	Visualization of how each of the different components that comprise this dissertation fit together . . . . .	8
2.1	Table that visually categorizes previous works based on the type of input network and downstream analysis . . . . .	15
3.1	Diagram describing processing steps of EDGEBOOST . . . . .	20
3.2	NMI of Baseline Community Detection Methods. NMI of six community detection algorithms with varying percentages of removed edges $\delta$ . . . . .	22
3.3	RE of Baseline Community Detection Methods. RE of six community detection algorithms with varying percentages of removed edges $\delta$ . . . . .	22
3.4	NMI heat map of six community detection algorithms. The parameters $\mu$ and $\delta$ are represented on the $x$ and $y$ axis respectively. Each square is labeled with the corresponding NMI value. . . . .	23
3.5	RE heat map of six community detection algorithms. The parameters $\mu$ and $\delta$ are represented on the $x$ and $y$ axis respectively. Each square is labeled with the corresponding NMI value. . . . .	23
3.6	Intra-edge Precision of Link Prediction. Precision plots of three link prediction algorithms: Adamic-Adar (left), Common Neighbors (middle), and Jaccard (Right) for various values of mixing parameter $\mu$ : 0.1 (top), 0.3 (middle), and 0.5 (bottom). The $X$ -axis corresponds to number of top- $k$ edges as scored by the link prediction algorithm as a percentage of the number of edges in the network. Intra-edge precision is on the $y$ -axis. . . . .	25

3.7	Edge Weight Distributions. Histogram of edge weights on a benchmark graph with $\mu=0.4$ and 20% of the edges removed: scores from AA link predictor (top) and weights of co-community network (bottom). . . . .	27
3.8	Karate Club Co-community Network. Visualization of the co-community network for “Zachary’s karate club” network. Each panel shows the network pruned at various thresholds $\tau$ . . . . .	30
3.9	EDGEBOOST Performance on LFR Networks. Performance of six popular community detection algorithms on the LFR benchmark networks. Dashed yellow bar shows the improvement of EdgeBoost over using the baseline community detection method. . . . .	32
3.10	EDGEBOOST Paired With Lovain. Performance of EdgeBoost (solid) and the baseline Louvain algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of edge boost for NMI. The bottom row of plots shows the relative error of the partition size. . . . .	33
3.11	EDGEBOOST Paired With InfoMap. Performance of EDGEBOOST (solid) and the baseline InfoMap algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size. . . . .	33
3.12	EDGEBOOST Paired With WalkTrap. Performance of EDGEBOOST (solid) and the baseline WalkTrap algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size. . . . .	34
3.13	EDGEBOOST Paired With Surprise. Performance of EDGEBOOST (solid) and the baseline Surprise algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size. . . . .	34
3.14	EDGEBOOST Paired With Significance. Performance of EDGEBOOST (solid) and the baseline Significance algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size. . . . .	35
3.15	EDGEBOOST Paired With Label-Propagation. Performance of EDGEBOOST (solid) and the baseline Label-Propagation algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size. . . . .	36
3.16	Performance of EDGEBOOST on Standard Network Datasets. Comparison of EDGEBOOST on set of standard real network benchmarks community detection . . . . .	37
3.17	Distribution of community sizes for Facebook ego networks. Nodes were given community labels by ego users as part of a user study. . . . .	38
3.18	Performance of EDGEBOOST on Facebook Networks. Comparison of EDGEBOOST on ego-networks from Facebook . . . . .	39
3.19	Varying <i>NumIterations</i> for EDGEBOOST with Louvain. The parameters are set as follows: $\mu = 0.2$ (left) and $\mu = 0.5$ (right) over $\delta$ values ranging from 0.0 to 0.6. . . . .	40

3.20	Varying <i>NumIterations</i> for EDGEBOOST with InfoMap. The parameters are set as follows: $\mu = 0.2$ (left) and $\mu = 0.5$ (right) over $\delta$ values ranging from 0.0 to 0.6. . . . .	40
3.21	Varying $\tau$ for EDGEBOOST with Louvain. Varying the co-community threshold ( $\tau$ ) for EDGEBOOST with $\mu = 0.2$ (left) and $\mu = 0.5$ (right) over $\delta$ values ranging from 0.0 to 0.6. . . . .	41
3.22	Varying $\tau$ for EDGEBOOST with InfoMap. Varying the co-community threshold ( $\tau$ ) for EDGEBOOST with $\mu = 0.2$ (left) and $\mu = 0.5$ (right) over $\delta$ values ranging from 0.0 to 0.6. . . . .	41
3.23	Analysis of Execution Time. Comparison of the runtime between EDGEBOOST and baseline Louvain algorithm on networks ranging from size 1000 to 128000 nodes. EDGEBOOST has the <i>NumIterations</i> set to 50. . . . .	42
3.24	Number of clusters vs. threshold . . . . .	45
4.1	BUTTERWORTH has three components. The <b>list generator</b> groups a user’s social contacts into topically coherent lists. The <b>list labeler</b> synthesizes labels for those lists. The <b>topic ranker</b> learns a ranking model for each topic by using each list to heuristically label training data. . . . .	54
4.2	Distribution of the number of lists created by users in the test set (20 is the maximum allowed by Twitter) . . . . .	62
4.3	F1 scores for the list-generation module, varying $k$ and $\alpha$ . . . . .	63
4.4	Precision and recall of the hashtag, unigram, naïve ranking methods against a baseline. . . . .	66
4.5	Varying the $k$ parameter for the hashtag (left) and unigram (right) methods. . . . .	66
4.6	A comparison of the effects of varying the percentage of noise on precision and recall for the naïve, hashtag, and unigram methods. . . . .	67
5.1	DOBBY consists of two components. The first component classifies parent-child relationships between keyword tags using features described in section 5.3. The second component prunes edges that participate in reciprocal predictions. . . . .	75
5.2	ROC curve . . . . .	78
5.3	Feature importance scores produced by the random forest model . . . . .	79
5.4	Sample output of DOBBY on two LinkedIn skills, <i>History</i> and <i>Writing</i> . These examples were generated by expanding out two levels from the root node, choosing the top-3 children for each parent. . . . .	80
5.5	precision recall curves on hold-out test set . . . . .	81
5.6	distribution of precision scores on human evaluated skills . . . . .	83
5.7	A table showing motifs that comprise of at least 95% of all motifs of sub-graph size 2,3 and 4 . . . . .	85
5.8	Precision of edge pruning heuristics . . . . .	88
6.1	Diagram describing processing steps of LOBBYBACK . . . . .	95
6.2	An sample state bill segment from Michigan SB 343 (2011) . . . . .	98

6.3	A visualization of a multi-sentence alignment and the resulting prototype sentence. . . . .	100
6.4	Precision, Recall, and F1 scores of LOBBYBACK and baselines . . . . .	101
7.1	Extension of EDGEBOOST framework to other types of network algorithms	108

# List of Tables

## Table

3.1	List of seven NLP topics used in our experiments along with input size. . .	44
3.2	Pyramid scores obtained by different content models for each topic along with average scores for each model across all topics. The best performing method has been annotated with a * . . . . .	45
4.1	The top three ranked tweets for automatically generated lists labeled <i>environment</i> , <i>fashion</i> , and <i>sports</i> (list labels are also automatically generated). .	57
4.2	RMSE and average relevance values for labeling algorithms. . . . .	65
4.3	Precision at k. . . . .	68
5.1	Table that shows the frequency of loops for sub-graph patterns ranging in size from 2-5. The second column shows the frequency of all loops, and the third column shows the frequency of patterns that contain loops larger than size 2. . . . .	86
6.1	Mean edit distance scores for LOBBYBACK and LexRank . . . . .	104

# Chapter 1

## Introduction

Networks have become an increasingly popular abstraction for problems in the physical, life, social and computing sciences. They are easy to state and flexible representations that can model a wide range of complex systems. The study of networks—dating back to work in the 1930’s by Jacob Moreno—has matured into its own discipline of network science. Due to the increased availability of data and computational resources, network science has flourished in the last two decades. Researchers have developed a multitude of algorithms that can be used to analyze networks and extract insights from their structure, which in turn provide insights into the underlying system being represented. *Network Analysis*, describes the process of investigating network structures through the use of algorithms.

Networks have been used to model phenomena in many disciplines, ranging from networks of people in the social sciences, to networks of genes and animal species in the life sciences [119]. Network analysis is a powerful tool because once a problem is modeled as a network, algorithms can be repurposed for solving similar problems in different domains. For instance, community detection algorithms extract clusters of densely connected nodes. Depending on the application, clusters can represent a group of friends in an online social network or research papers pertaining to the same topic in an academic citation network. Algorithms that measure the importance or *centrality* of nodes, have been used in applications ranging from identifying terrorists [28], the importance of websites [124], and even an empirical analysis of the American revolution [64].

A demonstrative example of how, even simple network analysis, can lead to interesting empirical insights, is shown in Figure 1.1. Nodes in this network each represent members of various organizations (e.g. “tea party”, “royal nine”) that formed before the American Revolution. The edges are weighted by the number of shared group affiliations between each node. By visualizing this network and ranking the node size by their eigenvector centrality, we can see that the famous Paul Revere (colored yellow) is ranked the highest out of all of the nodes. The ability for networks to represent so many different systems and phenomena while also being relatively simple models is what makes them such an appealing tool to

many scientific disciplines.

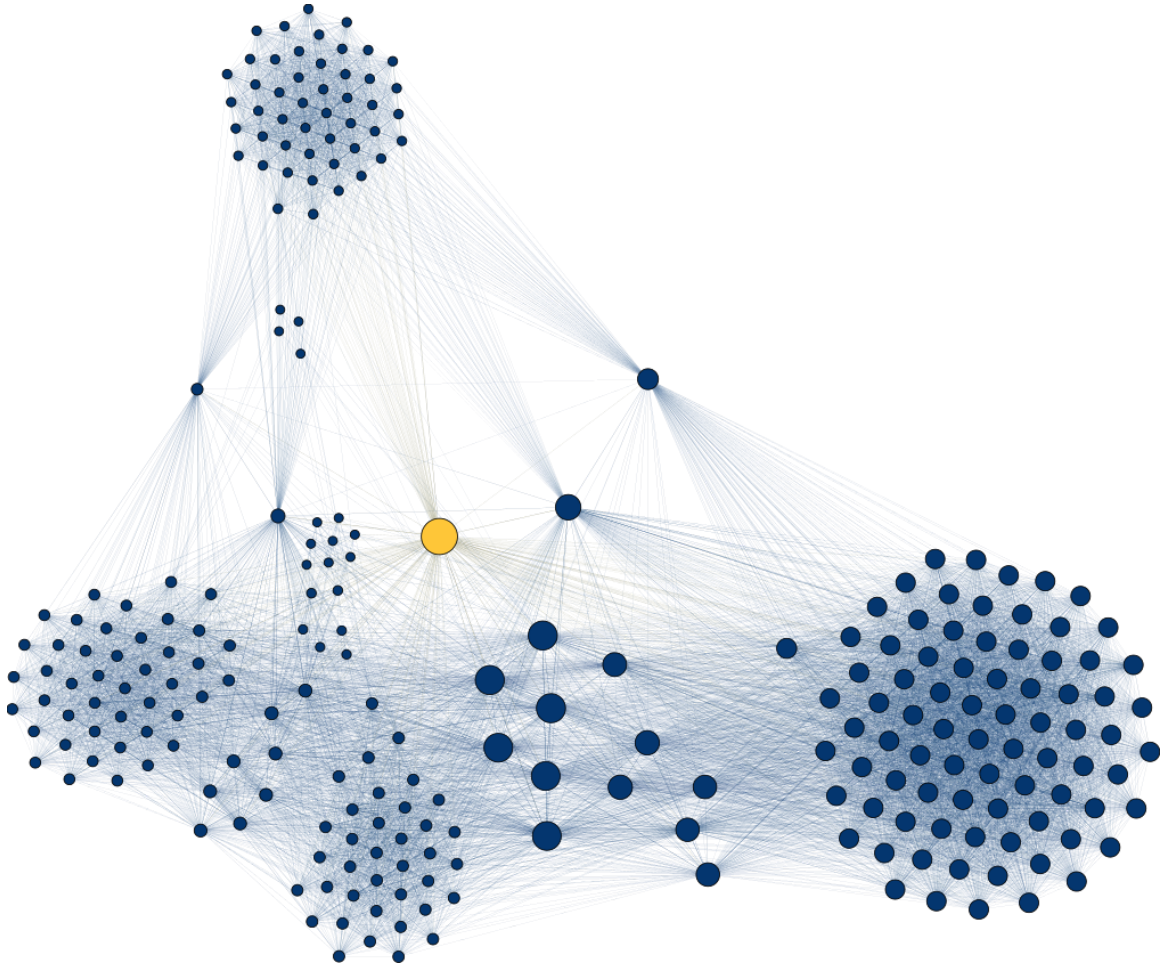
One of the aspects that makes network analysis a challenge for real-world problems is that a network is not always an observed and complete structure. One understated yet crucial step in applying network analysis, is the imputation of an existing structure or inference of the entire network for problems in which there is no inherent structure. Edges in networks can be missing for a variety of reasons, including: imperfect data collection or temporal constraints imposed on the network generation process. Missing edges in a network can cause degradation in the quality of network analysis, as algorithms that are used to analyze the network are less effective in the presence of missing information.

For certain types of problems that are modeled using networks, the structure of the network is not inherently expressed in the data. In these cases, networks are inferred by applying a data transformation on a dense similarity matrix in order to induce a sparser network representation. While this process can seem contrived, networks are often inferred for problems in which network analysis can provide better insight into an underlying system than approaches using similarity matrices [14, 42, 128]. For instance, systems biologists use network clustering algorithms to identify groups of genes that are highly co-expressed in microarray experiments. Gene co-expression networks are generated by thresholding the similarity measurements (e.g Pearson Correlation) based on experimental data [153]. Once the network structure is inferred, clustering can then be applied to the network to compute modules of genes that share similar function.

## 1.1 Network Representation of Data

Here we give a brief and more formal definition of networks. Networks, often referred to as *graphs*, are a representation for entities and the relationships between them. A network  $G = (V, E)$ , consists of a set of *nodes*, also known as *vertices*,  $V$ , which represent the entities to be modeled and the set of *edges*, also known as *links*,  $E$ , representing the pair-wise relationships between nodes. Networks are a flexible representation of data, in which  $V$  can be a set of any *one* or *many* type(s) of entity, representing people on social network, words in a corpus of scientific abstracts or even proteins in a protein-interaction network. Figure 1.1 gives an example of a network where nodes represent actors in the american revolution and weighted edges represent the number of group affiliations shared between the nodes. In many real-world network problems, nodes can have an associated meta-data consisting of attributes that describe the various properties of the node. For example, in the web graph (in which nodes are webpages), nodes have associated meta-data consisting of the HTML for





**Figure 1.1** An example network: nodes represent affiliates of various American revolutionary organizations and edges represent the number of organizations with a shared affiliation. Data provided by [70].

the webpage, title, and date of creation.

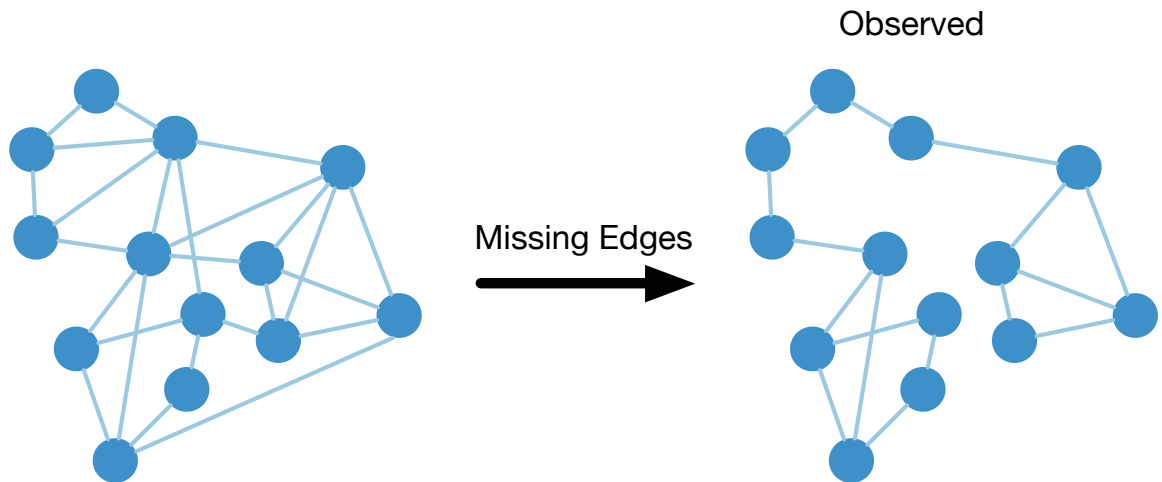
The edge set  $E$ , consists of a set of link vertex pairs,  $\{e = (v_i, v_j) \in E\}$ , where each element  $v_i, v_j$  is in  $V$ . In some cases, networks are *directed*, making the order for the pair matter but in other cases the graph is *undirected* making the order irrelevant. Edges can encode many types of relationships, such as: *friendOf*, *isA*, *similarTo*. Usually the type of relationship encoded by the edges determines whether a network is directed or undirected. Edge types such as *friendOf* form undirected edges, while edge types such as *isA* form directed edges. The set of nodes,  $\Gamma(v_i)$ , directly connected to a given node  $v_i$  are referred to as the neighbors of  $v_i$  and the *degree* of  $v_i$  is equal to  $|\Gamma(v_i)|$ .

## 1.2 Incomplete Networks

Networks are simply abstractions for describing data that is generated from an underlying social, physical, or computing system. No matter the representation, missing data is a common phenomena that needs to be considered for all types of data analysis. Depending on the circumstances, missing data can manifest in many ways, leading to networks that are missing nodes, edges and/or metadata. Missing nodes can occur in situations where the network is only partially observed, for instance, social networks constructed from survey data are only partially complete representations of the population they are modeling [2]. Missing nodes can also occur when analyzing cascade processes on networks [143]. In this case the network being modeled is the network of “infected” nodes, which is usually a partially observed phenomena. While missing nodes in networks is an important problem with a few proposed solutions [2, 91, 151], it is often the case that the set of observed nodes are the quantity one wants to analyze. For instance, even if the observed set of proteins in a dataset does not represent all proteins in a biological process, the goal of network analysis is to extract insight about the observed proteins. In this dissertation we investigate network structures that are missing edge information and when we refer to an incomplete network we are referring to the case where a network is missing edge information.

The quality of a network algorithm’s output depends on the structure of the input network. Missing edge information, whether in the form missing meta-data associated with links (i.e weights) or the non-existence of a link, leads to incomplete network structure. Since the output of a network algorithm is dependent on the observed structure, missing edges can degenerate the algorithm’s output. Networks can be incomplete in ways that differ based on the type of problem being modeled and and the intended type of analysis. For instance, if one wants to use a centrality algorithm to infer topical authorities, then a network structure containing only binary edges is less useful than weighted edges which reflect topical similarity between users.

We categorize network incompleteness into three scenarios. The first and simplest, the *missing edge* scenario, is one in which an observed network is only partially complete. In this scenario, the underlying network has edges that are not in the observed set of edges. The second scenario, *missing semantics*, involves network structures where the semantics of the edges are not present in the observed network. For example, the hyperlink edges in a web graph are not explicitly annotated with semantic information that describes the context of a link. The last scenario, *missing structure* is the scenario in which a network has no inherent structure but rather the structure is inferred from similarity data. In this scenario a network is “inferred” via a transformation on the similarity matrix.

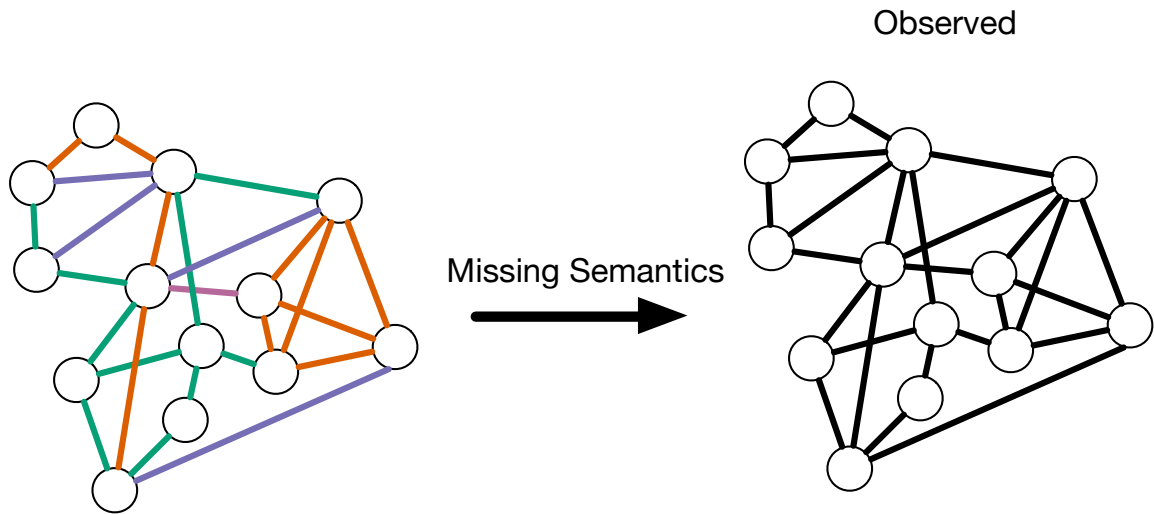


**Figure 1.2** Scenario 1: missing edges

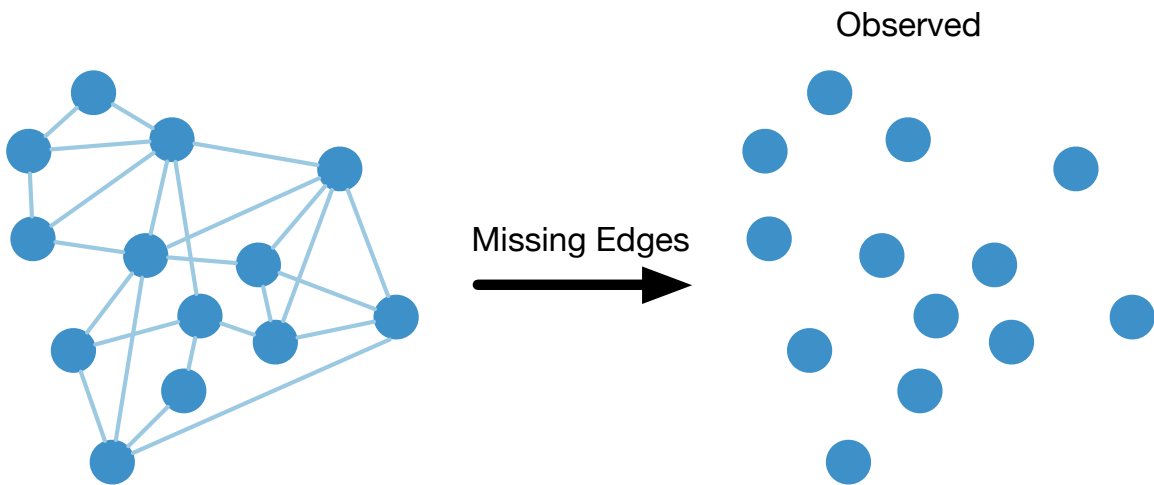
**Missing edges** represents the most well studied scenario in which networks are incomplete. Missing edges in networks can occur because of an incomplete data collection and/or network generation process. Data collection is prone to missing edges when networks are constructed from data that is intentionally sampled (i.e. Large Networks) or inadvertently sampled as is the case with crawled social network data in which some users make their connections private. Some processes that generate networks are inherently missing links. For instance, new users on social networks who haven't completed their connections and two proteins in protein-interaction networks that have not been analyzed in the same experiment. The main difference between the missing edge scenario and the other scenarios we discuss is that an inherent network structure exists and the problem is to infer *missing* links.

**Missing Semantics** in an incomplete network occurs when the edges are not annotated with information that describes the context of the link. In the case of a citation network for instance, edges can exist for a variety of reasons (e.g “criticism”, “validation”, “application”) but a vanilla citation network does not differentiate between these edge types. Semantic information about edges can come in the form of labels that describe the context of an edge, or edge weights which can measure the degree to which an edge represents a certain characteristic. Having contextual information about edges can lead to more fine-grained analysis. For instance, topic categories of webpages have been used to create topic specific networks that are used to construct an extension of PageRank that is topic centric [69].

**Missing Structure** in a network constitutes the most drastic scenario of incompleteness but occurs quite frequently. In this scenario, a network representation is not inherent to the data, rather a network structure is inferred from similarity data. Inferred networks



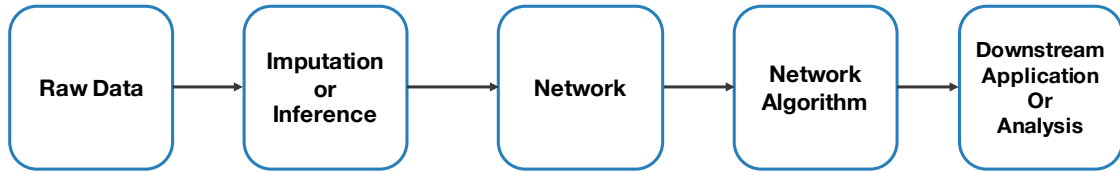
**Figure 1.3** Scenario 2: missing semantics



**Figure 1.4** Scenario 3: missing structure

can arise from simple transformations such as thresholding similarity matrices to induce sparser binary networks. Network inference can also be the result of more complicated transformations, consisting of aggregating similarity information from many different data sets or using a statistical models to infer structure instead of a fixed threshold. Missing structure denotes a scenario in which the network structure needs to be inferred from the data that describes the underlying system being modeled.

While these scenarios each describe different ways that networks can be incomplete, they are not mutual exclusive. It is possible that a network can be incomplete in two or all of the scenarios. For instance, a web crawl of Facebook will be missing edges due to privacy constraints of some users, while also missing semantic information.

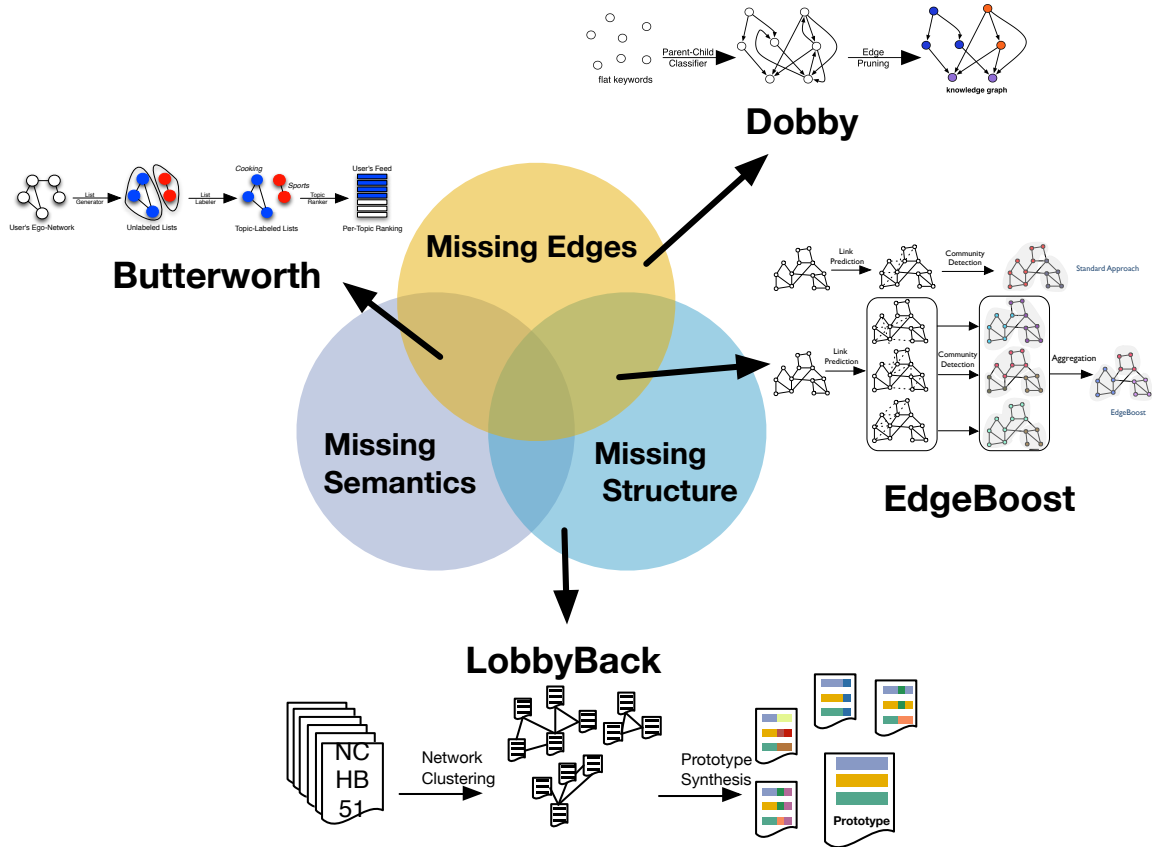


**Figure 1.5** Network analysis pipeline

### 1.3 Dissertation Overview

Applying social network analysis to data science problems, involves the construction of pipelines. A network analysis pipeline, as pictured in Figure 1.5, at a minimum consists of only the last three steps in the diagram. The core of network analysis is in the application of a given network algorithm and then the use of the algorithm’s output in a downstream application or analysis. In the case of an analysis, the last module is simply an examination the algorithm’s output. For many real-world problems, constructing network analysis pipelines poses a greater challenge then simply applying an algorithm on a given input network. Similar to how statistical modeling and machine learning pipelines deal with missing values, one important aspect of building a network analysis pipeline is the imputation and inference of the network structure. Depending on the problem and scenario, imputation and/or inference can have a big impact on quality of the overall analysis.

The thesis of this dissertation, is that the efficacy of network analysis can be improved by incorporating the inference or imputation of the network structure as a component of the pipeline. As shown in Figure 1.5, the choice of imputation or inference technique is dependent on the downstream application or network algorithm. For instance, in Chapter 4 we propose a network analysis pipeline for inferring topical communities in a social network. A critical component of this pipeline is the imputation of missing topical information in the links between users. By imputing this information we achieve a 15% percent improvement in community detection performance. We propose four novel systems, each of which, contains a module that involves the inference or imputation of an incomplete network for each scenario described in Section 1.2. In addition, the systems we propose solve real data science challenges, ranging from social feed ranking to community detection. The diagram in Figure 1.6 gives an overview of this dissertation. Each circle in the venn diagram represents a scenario of network incompleteness. Our proposed systems, described in Chapters (3-6) are shown along side the circles (scenarios) that they correspond to.



**Figure 1.6** Visualization of how each of the different components that comprise this dissertation fit together

## 1.4 Dissertation Organization

The remaining chapters in this dissertation are organized by the scenarios described in Section 1.2. Each chapter presents a data science problem in which the solution we propose involves the imputation or inference of an incomplete network as an integral component. First, In chapter 2 we give an overview of the existing literature on the incomplete networks and link prediction.

Chapter 3 presents EDGEBOOST, a meta-algorithm and framework that repeatedly applies a non-deterministic link prediction process to enhance community detection on incomplete networks with missing edges. EDGEBOOST first uses link prediction algorithms to construct a probability distribution over candidate inferred edges, then creates a set of imputed networks by sampling from the constructed distribution. It then applies a community detection algorithm to each imputed network, thereby constructing a set of community partitions. Finally, our technique aggregates the partitions to create a final high-quality community set. We show the efficacy of EDGEBOOST by testing its performance on both

randomly generated and real-world networks.

We address the problem of extracting topically cohesive clusters from an online social network in Chapter 4. We propose a social feed ranking system BUTTERWORTH, in which the first component is to extract a user’s topic(s) of interest by clustering her ego network. We propose a technique to re-weight the semantics of the ego-network using the textual content produced by the nodes in order to aid in the extraction of topically cohesive communities. BUTTERWORTH then uses these clusters to automatically train ranking models that rank a user’s social feed by the corresponding topic.

In Chapter 5 we propose a system DOBBY, for building a knowledge graph of user-defined keyword tags. The knowledge graph consists of a network where the nodes are keywords and directed edges represent hypernym *typeOf* relationships. By computing features based on various data sources that describe the tags and using a sparse set of hypernym edges labeled by humans, DOBBY trains a statistical classifier to infer the missing edges of the network. We evaluated the performance of DOBBY on a real data set of *skill* tags listed on the profiles of LinkedIn users.

Chapter 6 presents an application of network inference, in which we construct a network of bills from a corpus of state legislation. We propose a system, LOBBYBACK that uses community detection to extract clusters of bills that exhibit text re-use. LOBBYBACK then uses the clusters to construct “prototype” documents that represent the conical representation of the text shared between the documents. We apply LOBBYBACK to the task of reconstructing model legislation written by lobbyists that is known to have influenced clusters of documents in the corpus.

In each of these four systems the imputation or inference module either increases the performance of the pipeline or is requisite of the problem. The systems EDGEBOOST and BUTTERWORTH, for instance, are two examples of how by imputing the network, the efficacy of community detection algorithms can improve. EDGEBOOST shows an average improvement of 7% on artificial data and 17% on real social network data. Similarly, BUTTERWORTH shows that by imputing semantic information in a network, the F1-score of detecting topical communities can be improved by 16%. For certain problems, the network analysis pipeline requires the imputation or inference of a network to make network analysis feasible for the given problem. In Chapter 6, the first component of LOBBYBACK is the inference of a network structure in a corpus of text documents. The lack of network structure in this problem necessitates an inference module such that a community detection algorithm can be used to identify groups of documents. While each of the systems presented in this dissertation solve a specific technical challenge, the methods used for imputation and inference are more broadly applicable and could be re-purposed for the solution of problems.

# Chapter 2

## Literature Review

The imputation and inference of network structures is well studied in the literature. The work that has been done on imputation is referred to in the literature as *link prediction*, which involves the prediction of missing links in a network. Since link prediction can be done just by analyzing the observed structure of a network, many link prediction algorithms have been proposed that are agnostic to the domain or specific problem being modeled. The inference of network structures on the other hand is usually very tied to the problem being modeled—the features and data sources used to compute a similarity network of proteins is very different from those of lexical networks. We first give an overview of general link prediction algorithms many of which we will use in later chapters. We then provide an overview of the literature that focuses on using imputation and inference for certain types of network analysis. Finally we provide a brief overview of community detection, since it is the main type of network algorithm used in this dissertation.

### 2.1 Link Prediction

Link prediction has garnered a substantial body of literature, encapsulating a variety of methods and applications in many domains. Methods for link prediction utilize the topology and in some instances the meta-data of nodes, to infer missing, or predict future links in networks. Most of the literature on link prediction has focused on the task of predicting missing links as the end goal of the analysis. For instance, many of the papers focusing on predicting the missing links between friends in a social network in an effort to solve the “cold start” problem.

Methods for link prediction fall into two major categories, supervised and unsupervised, and within these categories methods rely on the use of node meta-data, neighborhood information, network topology or a combination to make inferences. Surveys for link prediction on social networks include [67, 176] and more generally, link prediction in complex networks [106].



### 2.1.1 Unsupervised Link Prediction

Similar to unsupervised methods in the machine learning literature, unsupervised link prediction methods do not rely on labeled examples of links. The standard workflow consists of a link-predictor scoring the likelihood of edges between nodes not connected in the input network  $G$ , and then ranking these edges by score. Once ranked, the quality of the ranking can be evaluated using metrics such as *area under a receiver operating characteristic* [65] as well as Precision/Recall [71]. In order for the ranked list of edges to be useful in real systems, a threshold must be chosen such that all edges ranked above the chosen threshold are then added to the network. Most of the unsupervised methods in link prediction fall under the category of similarity heuristics, in which a similarity function  $S(v_i, v_j)$  is used to score the likelihood of an edge between nodes  $v_i$  and  $v_j$ . Similarity functions over pairs of nodes can be arbitrary, but most methods fall under the categories of local, path, and meta-data based functions.

**Local Methods** — Local link prediction functions compute the likelihood of two nodes  $v_i, v_j$  having an edge on the similarity of their neighborhood structures,  $\Gamma(v_i), \Gamma(v_j)$  respectively. The most basic local method, common neighbors (CN), ranks edges based on the number of shared neighbors between  $v_i$  and  $v_j$ .

$$CN(v_i, v_j) = |\Gamma(v_i) \cap \Gamma(v_j)| \quad (2.1)$$

The CN method captures the notion that similar nodes share similar neighbors. Common neighbors encodes the propensity of two nodes to have an edge based on the number of triangles they close if connected, in other words two nodes are more likely to be connected if they have a high degree of triadic closure [36]. Other local methods normalize the the CN function in order to make the calculation equivalent for nodes of both high and low degrees. Methods proposed by [79, 98, 140, 145, 154] propose variations on the CN function with the following form:

$$\frac{|\Gamma(v_i) \cap \Gamma(v_j)|}{N(v_i, v_j)} \quad (2.2)$$

Each method differs in the normalization function  $N(v_i, v_j)$ . Among all of the the *normalization* methods, the Jaccard performs at least as good as or better then the others [100, 182].

In addition to normalizing the CN method, other proposed variations re-weight the contribution of each shared neighbor [3, 182]. The AA measure proposed by Adamic and Adar, re-weights a common nodes contribution by the inverse of the log, similar to the

popular TF-IDF metric used in information retrieval [107].

$$AA(v_i, v_j) = \sum_{z \in \Gamma(v_i) \cap \Gamma(v_j)} \frac{1}{\log(\Gamma(z))} \quad (2.3)$$

Resource Allocation (RA) is a newer measure proposed by Zhou *et al.*, re-weight each common neighbor contribution by just the inverse of the degree. Both of these methods are competitive, usually outperforming all other local methods in comparative studies [100, 182]. Local method are the most simple similarity functions but are very popular because of their effectiveness and scalability. Sarkar *et al.* [147] provide a theoretical justification for local based methods, showing that most links only exist if nodes are “close” via local similarity.

**Path and Random Walk Methods** — Path based methods compose of link prediction functions that compute a similarity score based on the number of paths between the query nodes. The set of common neighbors comprise of all paths of length two between the query nodes, therefore path methods are extensions of local methods to arbitrarily long paths. The Katz method [1] computes a similarity function based on the number of all paths between the query nodes. The function uses a standard discounted sum such that longer paths contribute less to the computed score. If we let  $\text{path}_{v_i, v_j}^\ell$  be defined as the number of paths between  $v_i, v_j$  of length  $\ell$ , then the katz score is defined as follows.

$$\text{Katz}(v_i, v_j) = \sum_{\ell=1}^{\infty} \beta^\ell \times |\text{path}_{v_i, v_j}^\ell| \quad (2.4)$$

A related method, Local Paths (LP) [105] truncates the summation in the Katz function, only including paths of size three and below. Their method is much more tractable then the katz method, and is competitive in performance.

Random walk based methods utilize random walks as a proxy for well connected two query nodes are. The *Hitting Time* (HT) and its symmetric variant Commute Time (CT)[50] computes the expected number of steps a random walker needs to take in order to go from  $v_i$  to  $v_j$ . SimRank [82] computes a score, based on how often two random surfers, one starting at  $v_i$  and the other at  $v_j$  are expected to meet at a common vertex. The *rooted PageRank* [100] is an adaptation of the original PageRank algorithm to measure the probability of reaching node  $v_j$  from  $v_i$  with a specified restart parameter. Random walk methods are computationally more complex then local methods and they do not show any increase in performance, in many cases showing worse performance in various comparison studies [100, 182].

**Node Methods** — In many real-world networks, nodes contain meta-data that could be leveraged to predict links. For example, profile text of users in a social network, web page

html in a web network, and publication meta-data in a citation network. The social theory of *Homophily* states that people that are more similar to each other are more likely to be friends and has been observed to be true on social networks in particular [109]. There are many methods to compute the similarity of unstructured text [59], all of which can be used to infer a metric of similarity between nodes with associated unstructured text. For nodes that contain structured attributes, vector based similarity metrics [27] can be used to compute similarity. Bhattacharyya *et al.* [11] proposed a tree-based method for computing similarity between nodes with hierarchical meta-data.

Since node based similarity does not consider the network topology, using node based methods in isolation can be noisy. Node similarity is better used to compute weights for existing edges, or incorporating into existing measures that utilize the network topology. Adamic and Adar [3] proposed a variant of their AA method that extends the concept of shared neighbors to all *shared items* which can be anything from node attributes to words in profile text. This variant of AA is used in Chapter 4 to predict links between users that discuss similar topics in a social network.

## 2.1.2 Supervised Link Prediction

Supervised link prediction is the formulation of link prediction as a supervised learning problem. There are many algorithms for learning a classifier of which ensemble techniques based on trees usually perform best [46]. The two necessary components for supervised learning are training data and features. All of the unsupervised similarity functions can be used as features in a supervised link predictor [67, 166]. The benefit of supervised link prediction is in the ability to combine many similarity functions into a framework that automatically learns to weight the relative importance of each function. Topological features (local, path, and random walk methods) are the most general features, because they apply to any link prediction problem. Supervised methods can also use features based on domain specific information, which can be anything from text similarity, node meta-data to data extracted from external sources. For instance, Hasan *et al.* [68] propose a feature for co-authorship link prediction that counts the number of categories that a pair of authors is associated with in order to ascertain the extent to which two authors are “interdisciplinary”, and therefore more likely to have co-authors. The creation of high quality is essential for good performance of a supervised link predictor, but beyond topological features, good feature design highly depends on the problem domain.

Training data is also an essential component of supervised learning, and one of the major challenges in the deployment of supervised link predictors. For supervised link prediction,

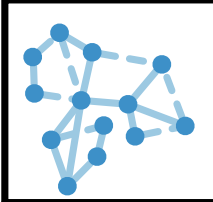
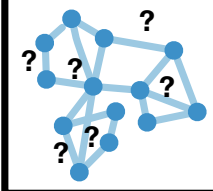

training data consists of edges that are labeled as positive (they are true edges) or negative (they are nonexistent). For an arbitrary network, the positive examples can be taken to be all links present in a given network. Negative links are not as easy to ascertain since edges that are not presented in the network are not necessarily negative examples. Due to the the difficulty in generating quality training data, supervised methods have been mostly applied in link prediction problems that involve time-evolving networks [68, 101, 164, 165, 178].

Many previous projects have used supervised learning for time-evolving, co-authorship [68, 178], social [164] and human mobility [165] networks. All of these methods propose the process of building a training set based on two non-overlapping and consecutive time intervals. By observing which links occur (or do not) in the second interval, positive and negative training examples can be generated. Even in scenarios where training data is easier to generate, the problem of class imbalance is still present since nonexistent links always significantly out number the positive examples. The method proposed by Yu *et al.* [178] addresses the problem of class imbalance using an optimization technique known as chance constrained programming. Other works such as [101, 165] utilize sampling based-techniques as a means to balance the training set.

## 2.2 Link Prediction for Improving the Network Analysis Pipeline

In addition to the previous work that has focused on building stand-alone imputation and inference methods, there is also a body of literature that use these methods to improve network analysis pipelines. This existing work can be classified along two dimensions based on which scenario(s) the problem is categorized in and the type of network algorithm applied after the imputation/inference component of the proposed pipeline. Figure 2.1 shows a visual representation of this categorization. We now describe the previous work on improving the two most common types of network analysis: community detection and centrality computation.

There has been a number of works that have focused on improving network analysis pipelines for community detection on incomplete networks. The majority of these works [26, 111, 174, 175] fall into the missing edge scenario and involve the use of link prediction to enhance community detection. Yan *et al.* [174] hypothesis different ways a network can be missing edges (e.g "incomplete web crawl", "random deletion) and show how the quality of community detection is affected. Papers by Mirshahvalad *et al.* [111] and Bowen *et al.* [175] use the common neighbors link prediction algorithm in two dif-

		Analysis Type		
		clustering	centrality	...
Input		[15],[79] [126],[127]	[11],[34] [89]	
		[72],[22]	[127],[47]	
		[132]		

**Figure 2.1** Table that visually categorizes previous works based on the type of input network and downstream analysis

ferent ways: Mirshahvalad *et al.* rank missing edges and imputes the top  $k$ , and Bowen *et al.* re-weight the existing edges by the score of the link predictor. Chen *et al.* [26] propose a method that ranks all missing *and* existing edges in the network, replacing the  $k$  bottom scoring existing edges with  $k$  top scoring missing edges.

Previous work also exists for incomplete networks with *missing semantics* and *missing structure*. Lin *et al.* [104] propose a new topic model based community detection algorithm that uses network structure and node attributes of the user. Social networks such as Twitter have many different link contexts (e.g follower-follow, mentions, hashtag usage) all of which can be used to contextualize the relationships between users. Work by Darmon *et al.* [34] propose link re-weighting schemes based on these different contexts, in order to infer communities that “answer different questions”. One of the more popular pipelines for clustering gene-co expression, WGCNA [181] use a weighted variant of common neighbors method, *topological overlap*, to re-weight edges in networks inferred from gene-co expression data.

Pipelines that improve centrality measures on incomplete networks have also been studied. Many studies have explored the extent to which centrality measures are robust to missing edges [15, 51, 130]. Borgatti *et al.* [15] and Platig *et al.* [130] investigate the effects of both missing and spurious edges have on the most common centrality measures (e.g. eigenvector and betweenness). The effects of missing edges can differ based on the topology of the network, Frantz *et al.* [51] show how missing edges effects certain network models, such as Erdos-Renyi and preferential attachment networks. Networks that exhibit *missing semantics* can also impact centrality computation.

A few works have used link prediction techniques that weight edges based on textual similarity of nodes. TwitterRank [169] is a system for inferring topical authorities on Twitter. The system re-weights the network structure by computing the KL divergence of topic distributions (computed using LDA) between users. Another system proposed by Haveliwala *et al.* [69] computes weighted variants of PageRank based on topic categories of webpages distilled from the DMOZ open directory.

## 2.3 Community Detection Overview

Community detection is a type of network algorithm used extensively throughout this dissertation. Certain algorithms—depending on the type of objective function they are trying to optimize—produce communities with certain properties (i.e. granularity) or are more/less effective depending on the degree to which the given network exhibits community structure. The downstream application of a network analysis pipeline can also dictate the choice of algorithm. For instance, in Chapter 6 of this dissertation we describe an application that requires a finer level of granularity of community, therefore we chose the InfoMap algorithm because it is known to detect communities at a finer granularity.

There are many variants of the community detection problem: communities can be disjoint, overlapping, or hierarchical. In this dissertation we use disjoint community detection algorithms. While the other variants, especially overlapping community detection, are of growing interest, detecting strict partitions is still a hard and relevant problem. In fact, recent work [134], has shown that disjoint algorithms can perform better than overlapping algorithms on networks with overlapping ground truth. The community detection algorithms we use in range from proven techniques including: Louvain [13], InfoMap [142], Walk-Trap [131], Label-Propagation [138] and newer state-of-the-art techniques: Significance [158] and Surprise [6, 157].

The Louvain algorithm [13] is very popular, mostly due to its balance of speed and

accuracy. It works by using a multistep technique based on a local optimization of Modularity [120]. Each iteration of the algorithm merges nodes in the same community into supernodes, yielding a new network that is used in the subsequent iteration. The algorithm terminates when modularity (computed on the original graph) reaches a local maximum. Louvain's optimization algorithm has been adapted to work with other quality functions. Many of the other algorithms listed below are the Louvain algorithm paired with a different quality function.

The InfoMap algorithm [142] is a technique that formulates community detection as the problem of optimally compressing a random walk in a network. The optimal compression is achieved using a quality function that is based on the minimum description length of the random walk. While the original implementation of InfoMap used simulated annealing for its optimization algorithm, the latest implementation uses the Louvain optimization algorithm. Similar to the InfoMap algorithm, Walktrap [131] uses random walks on a network to identify community structure. The algorithm computes distances between nodes based on the random walks in the network, using a hierarchical agglomerative clustering algorithm to cluster the computed similarity matrix.

The Label-Propagation algorithm [138], uses the concept of node neighborhood and simulates the diffusion of information in the network to identify communities. Initially, each node is labeled with a unique value. Then an iterative process takes place, where each node takes the label which is the most common in its neighborhood (ties are broken randomly). This process goes on until convergence. Communities are then obtained by considering groups of nodes with the same label.

Significance [158] and Surprise [6, 157] are two, more recent, objective functions for community detection based on a probabilistic interpretation. Significance is a measure that is based on whether a given partition, will have a certain number of intra-community edges, as compared to a random graph. The probability is computed in such a way that it is agnostic to the possible permutations of nodes, since permutations of nodes yield the same edge structure. Surprise is a statistical approach to assess the quality of a partition into communities. Surprise is based on the probability of drawing the number of intra-community edges observed in the network. This distribution follows a hyper-geometric distribution. The implementation of both Surprise and Significance use the Louvain optimization algorithm.

# Chapter 3

## Community Detection on Incomplete Networks

Many types of complex networks exhibit community structure: groups of highly connected nodes. Communities or clusters often reflect nodes that share similar characteristics or functions. For instance, communities in social networks can reveal user’s shared political ideology [30]. In the case of protein interaction networks, communities can represent groups of proteins that have similar functionality [86]. As a result, community detection has become one of the fundamental types of network analysis. The clusters that result from community detection enable network analysts to distill more abstract and higher order structures which can be used to reason about the underlying system being model or in downstream applications (an example of which we show in chapter 4). Despite the profusion of work on both the creation of community detection algorithms and their applications, very little attention has been given to how community detection algorithms fair on networks with missing edges.

In this chapter we analyze how missing edges can affect the quality of community detection algorithms and propose a novel meta-algorithm that uses link prediction to improve the performance of community detection on incomplete networks. Our algorithm, EDGEBOOST, utilizes existing community detection methods that process networks imputed by our link prediction based sampling method and merges their multiple partitions into a final consensus output. In addition, we propose an extension of EDGEBOOST for clustering inferred networks and apply our extension to an existing system that uses a network analysis pipeline to generate scientific surveys. The work presented in this chapter is based off of the algorithm described in our paper [18].

### 3.1 Problem Overview

Community detection algorithms rely on the topology of the input network to identify meaningful groups of nodes. Unfortunately, real networks are often incomplete and suffer

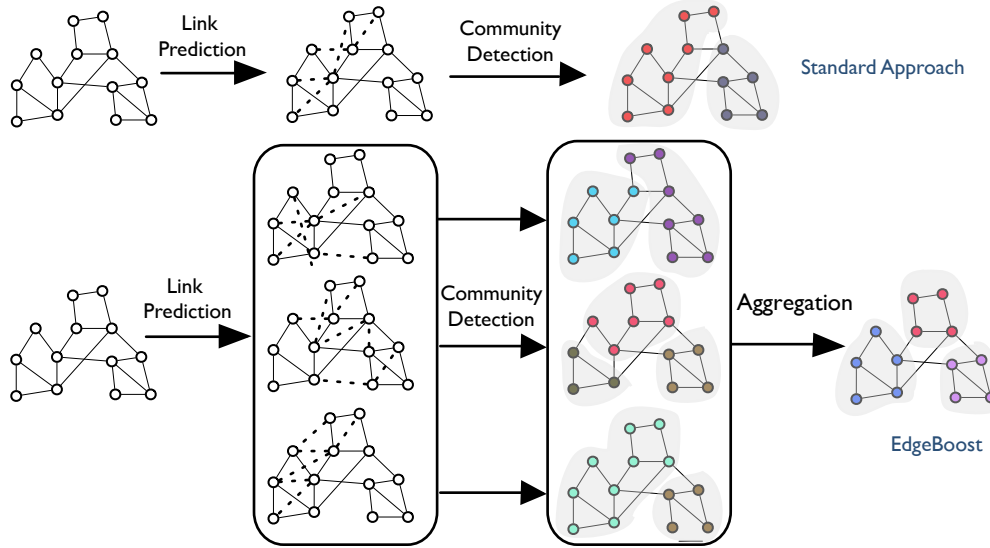


from missing edges. For example, social network users seldom link to their complete set of friends; authors of academic papers are limited in the number of papers they can cite, and can clearly only cite already-published papers. Missing edges can also be a result of the data collection process. For instance, Twitter often limits its data feed to only a 10% “gardenhose” sample: constructing the mention graph from this data would yield a graph with many missing edges [114]. Datasets crawled from social networks with privacy constraints can also lead to missing edges. In the case of protein-protein interaction networks, missing edges result from the noisy experimental process used to measure pairwise interactions of proteins [78]. Community detection algorithms rarely consider missing edges and so even a “perfect” detection algorithm may yield wrong results when it infers communities based on incomplete network information.

One straightforward approach for improving community detection in incomplete networks is to first “repair” the network with *link prediction*, and then apply a community detection method to the repaired network [111]. The link prediction task is to infer “missing” edges that belong to the underlying true graph. A link prediction algorithm examines the incomplete version of the graph and predicts the missing edges. Although link prediction is a well-studied area [100, 106], little attention has been given to how it can be used to enhance community detection. Imputing missing edges using link prediction can result in the addition of both correct intra-community and incorrect inter-community links. If one were to simply run a link predictor and cluster the resulting network, the output can only be improved if the link predictor accurately predicts links that reinforce the true community structure.

In this chapter we propose the EDGEBOOST method (Figure 3.1), a meta-algorithm and framework that repeatedly applies a non-deterministic link prediction process, thereby mitigating the inaccuracies in any single link-predictor run. EDGEBOOST first uses link prediction algorithms to construct a probability distribution over candidate inferred edges, then creates a set of imputed networks by sampling from the constructed distribution. It then applies a community detection algorithm to each imputed network, thereby constructing a set of community partitions. Finally, our technique aggregates the partitions to create a final high-quality community set.

An important and desirable quality of our method is that it is a *meta-algorithm* that does not dictate the choice of specific link prediction or community detection algorithms. Moreover, the user does not have to manually specify any parameters for the algorithm. We propose an easy-to-implement, black-box mechanism that attempts to improve the accuracy of any user-specified community detection algorithm.



**Figure 3.1** Diagram describing processing steps of EDGEBOOST

## 3.2 Detecting Communities in Incomplete Networks

To motivate the need for algorithms that are robust to missing edges, we experimented on existing community detection algorithms. To test these algorithm’s sensitivity to missing edges on a range of networks, we utilize the LFR benchmark [96]. LFR creates random networks with *planted partitions* (i.e., ground-truth community structure), parametrized by: number of nodes, mixing parameter  $\mu$ , and exponent of degree and community size distributions (see [96] for a full description). The mixing parameter is a ratio that ranges from only intra-community edges (0) to only inter-community edges (1). Previous studies [5, 93] have compared the quality of community detection algorithms using the benchmarks and used  $\mu$  as the variable parameter, roughly capturing how difficult a network is to cluster. As we are concerned with characterizing the effect of missing edges, we modify the LFR benchmark by randomly deleting edges from the networks it generates. We denote the parameter  $\delta$  as the percentage of removed edges.

The goal of our analysis is to characterize the effect of both  $\mu$  and  $\delta$  on two metrics: Normalized Mutual Information (NMI) and the Relative Error (RE) of the size of the inferred partitions. NMI is a standard information theoretic measure for comparing the planted partition provided by the benchmark to the inferred partition produced by the algorithm. There are various metrics classified as normalized mutual information; the metric we use throughout this chapter is the normalization of mutual information (I) based on maximum

entropy ( $H$ ) of the two partitions.

$$NMI(U, V) = \frac{I(U, V)}{\max(H(U), H(V))} \quad (3.1)$$

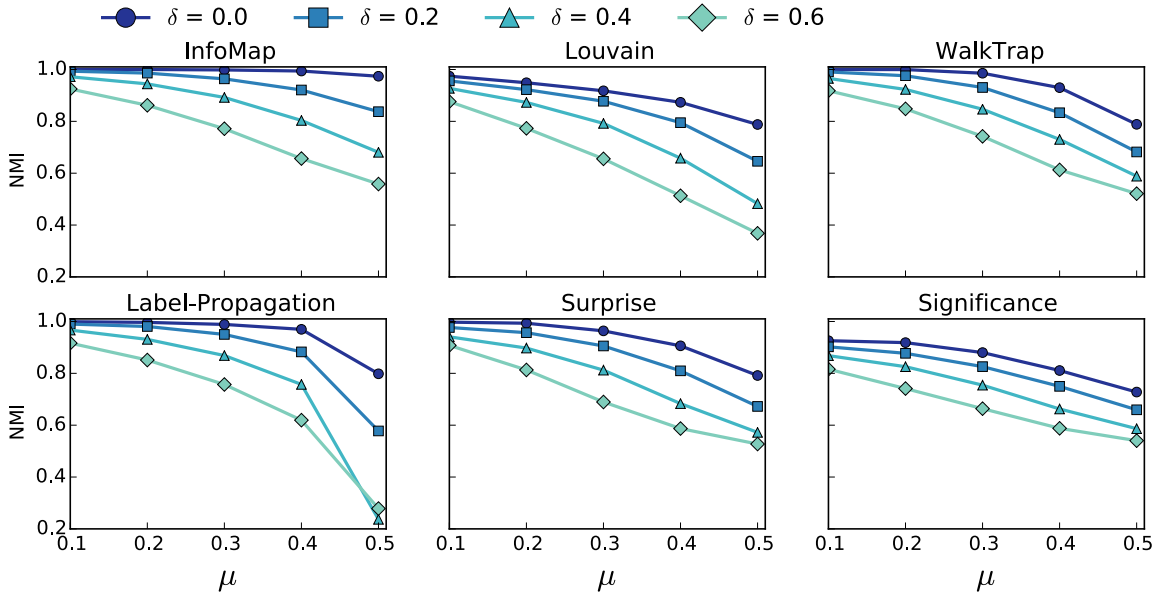
We define RE as the relative error of the number of communities inferred by the algorithm  $C$  compared to the number of communities  $C^*$  in the planted partition:

$$RE := \frac{C - C^*}{C^*} \quad (3.2)$$

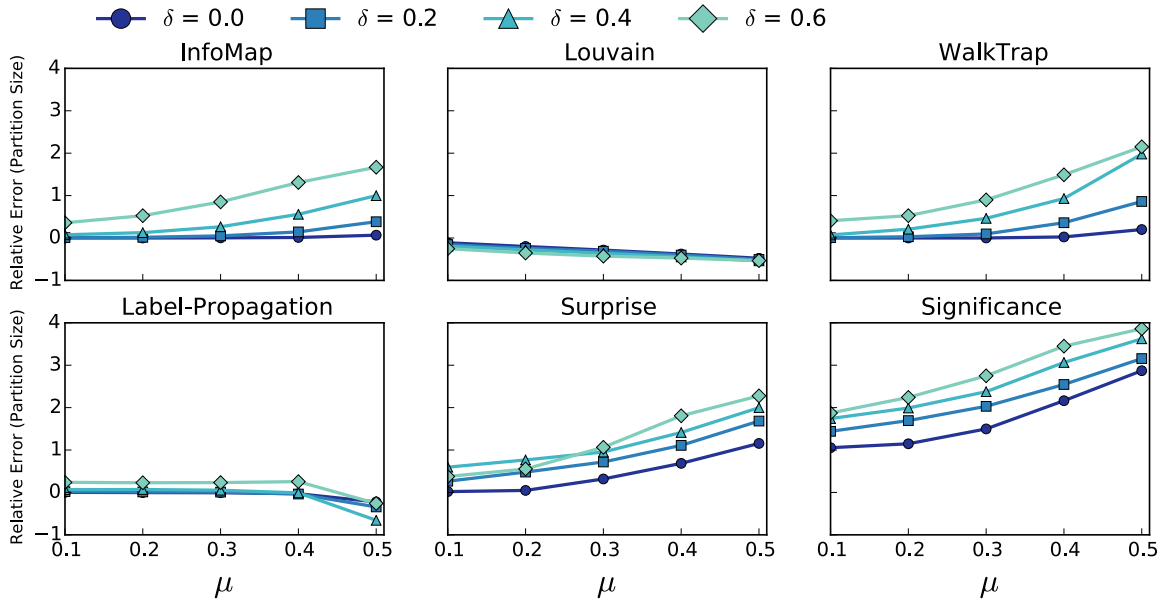
Since NMI can decrease for a variety of reasons (shifted nodes, shattered or merged communities), we include RE as a means to determine the more specific effects that missing edges can have on community detection. Each point in Figs. 3.2 and 3.3 are generated by averaging the corresponding statistic over 50 random networks generated by our modified LFR benchmark. We set static values for the following benchmark parameters: Number of nodes (1000), the average degree (10), the maximum degree (50), the exponent of the degree distribution (-2), exponent of the community size distribution (-1), minimum community size (10), and maximum community size (50). We varied parameters, such as “number of nodes” and “average degree”, finding qualitatively similar results for the effect of  $\delta$  on NMI and RE. Similar to previous research [111], we select an “average degree,” that results in the sparse networks that motivate the need for the methods presented in this chapter.

Fig. 3.2 shows how NMI varies with respect to  $\delta$  and  $\mu$  for six popular community detection algorithms. We limit the values of  $\mu$  to be in the range  $[0.1, 0.5]$  because it has been shown that LFR networks with  $\mu$  values of 0.5 and higher do not reflect the expected properties of real world networks [122]. All of the algorithms behave in a qualitatively similar manner: as  $\delta$  increases, the NMI score decreases. Similar to previous studies, InfoMap scores best with respect to  $\mu$  and not surprisingly is also the most robust to missing edges. More interesting are the results in Fig. 3.3 which show how the number of inferred communities differs with respect to the number of communities in the planted partition. Four of the six algorithms show a trend of detecting too many communities both as a function of  $\mu$  and  $\delta$ , while only the Louvain and Label-Propagation algorithms detect fewer than the correct number of communities on average. Modularity is known to suffer from a resolution limit [49], meaning that the measure tends to favor larger communities. Since Louvain uses modularity as its objective function, it is not surprising that Louvain, on average, infers communities that are larger than in the planted partition. Overall, it is more often the case that missing edges will cause community detection algorithms to “shatter” ground truth communities, sometimes producing 2-3 times more communities. Both in terms of NMI and RE, all 6 algorithms show a significant deterioration in community detection quality,

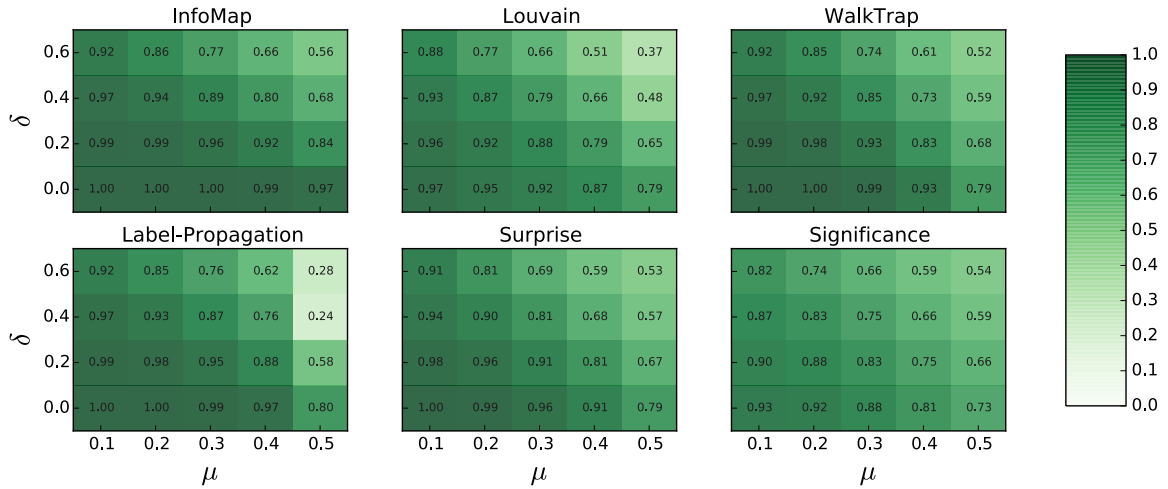
once again, underscoring the need for algorithms that are robust to missing edges. We have also included heat map versions of Figures 3.2 and 3.3 in the appendix. They are labeled as Figures 3.4 and 3.5 respectively .



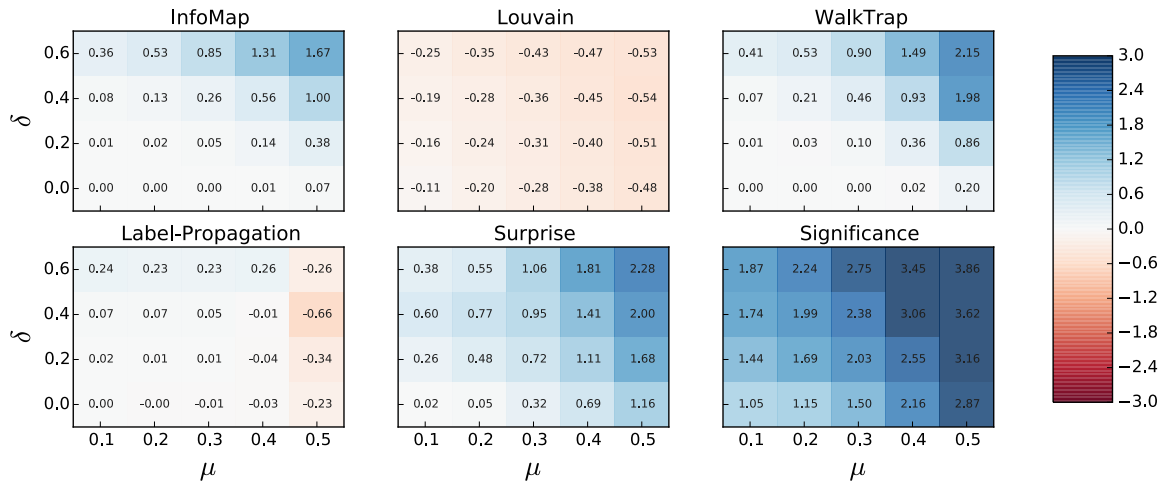
**Figure 3.2** NMI of Baseline Community Detection Methods. NMI of six community detection algorithms with varying percentages of removed edges  $\delta$ .



**Figure 3.3** RE of Baseline Community Detection Methods. RE of six community detection algorithms with varying percentages of removed edges  $\delta$ .



**Figure 3.4** NMI heat map of six community detection algorithms. The parameters  $\mu$  and  $\delta$  are represented on the x and y axis respectively. Each square is labeled with the corresponding NMI value.



**Figure 3.5** RE heat map of six community detection algorithms. The parameters  $\mu$  and  $\delta$  are represented on the x and y axis respectively. Each square is labeled with the corresponding NMI value.

### 3.3 Link Prediction for Enhancing Community Detection

The ideal scenario for community detection is one where a network consists of only intra-community edges and where the detection of communities reduces to the problem of identifying weakly connected components. The reality is that we rarely find such clean graphs as edges can be “missing” for anything ranging from sampling to semantics. This last factor is important as a missing edge between nodes in the same community is not necessarily incorrect—the semantics of a network does not necessitate an explicit relationship between users in the same community. In the case of an ego-network on Facebook, for

example, not all friends in the same community actually know each other as they may be grouped because they attend the same college as the ego-user. Similarly, a biological network may have a set of proteins working in concert as part of a functional “community” but many do not form a clique as the edges represent (up or down)-regulation. In both scenarios the edges that are missing are implicit edges representing the intra-community links (e.g., an edge representing the relationship *in-the-same-community-as*). It is these intra-community edges, whether they are implicit or explicit, that can have severe impact on the detection of communities.

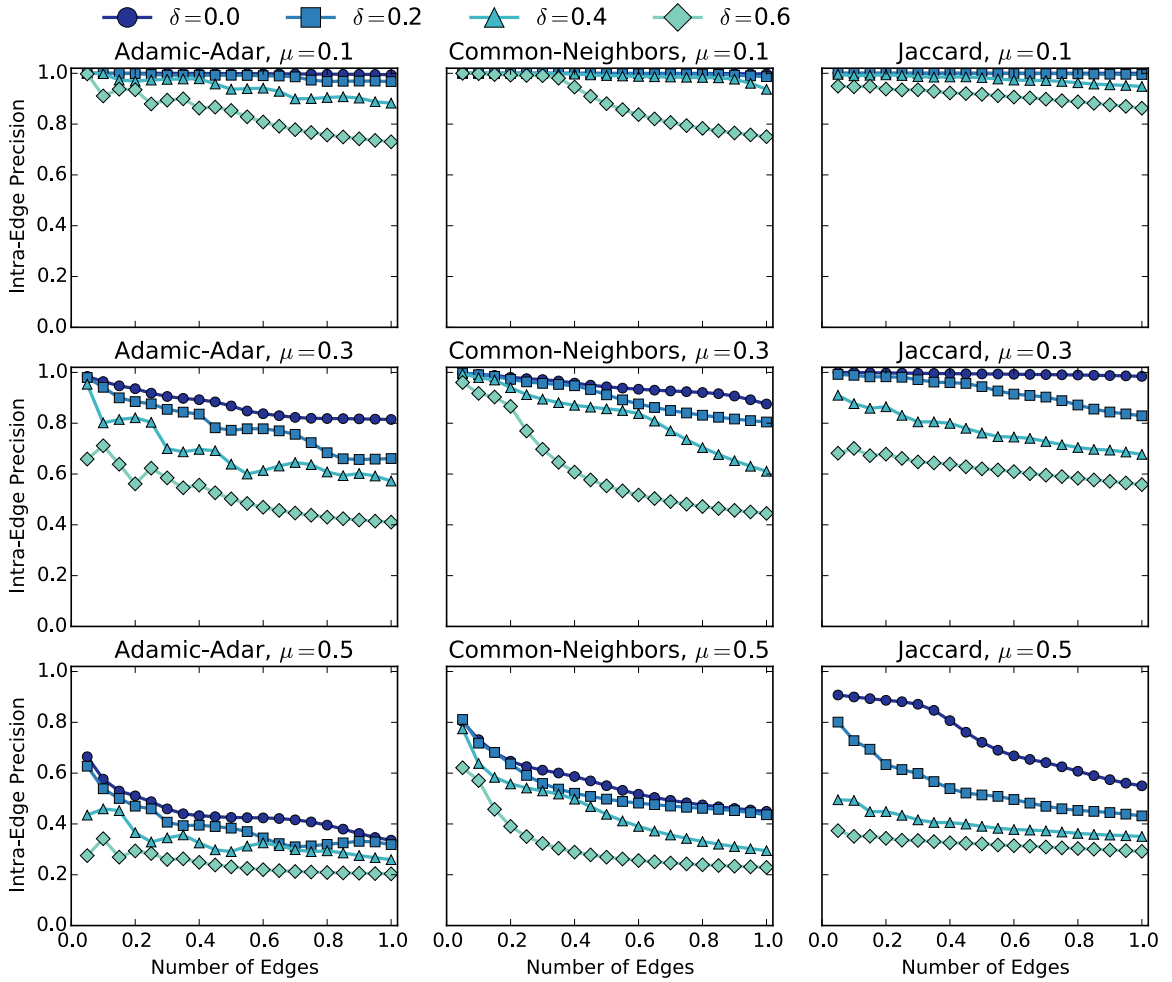
Our hypothesis is that by recovering edges in incomplete networks, community detection quality can be improved. If link prediction is to be an effective strategy at recovering lost community structure, it must be accurate at predicting intra-community edges that reinforce communities. If the link prediction algorithm has too high a false-positive rate, thereby predicting too many inter-community links, it is likely to degrade community detection performance. Using the modified LFR benchmark, we analyzed the intra-community precision of various link prediction algorithms over a range of  $\mu$  and  $\delta$  values. We do not intend to exhaustively test all of the link prediction algorithms proposed in the literature, but we select three computationally efficient techniques that are among the best [100, 106]: Adamic-Adar (AA), Common-neighbors (CN), and Jaccard.

Each of these algorithms can produce a score for missing edges that complete triangles in the input network, allowing us to create a partial ordering over the set of missing edges. Fig. 3.6 shows the results from our experiment. For each plot, the y-axis represents the intra-edge *precision-at-k* metric, which is the percentage of intra-edges in the top- $k$  edges of the ranking. The x-axis represents the edge-percent value, which is the number of top- $k$  edges as a percentage of the total number of edges in the original network (before random deletion). For example, if the original network had 2000 edges, then an edge-percent value of 20% would correspond to selecting the top-400 edges and a intra-edge precision value of 80% would correspond to 320 of those edges being intra-community edges. By varying  $k$  we are able to observe the classification quality inferred by the ranking produced by each link-predictor.

In Fig. 3.6 we first notice that as with community detection, link prediction performance decreased as a function of both  $\delta$  and  $\mu$  value. For low  $\mu$ , all link prediction algorithms are capable of achieving high intra-edge precision even for  $\delta$  values of 60%, but the quality of link prediction drops significantly for high levels of  $\mu$ . For  $\mu$  above 0.5, any link-predictor that uses the number of common-neighbors as a signal will do poorly, since the majority of a node’s neighbors belong to different communities. The Jaccard algorithm maintains the highest level of precision as a function of the number of edges. While the AA algorithm

sometimes outperforms Jaccard, AA is only better for low values of  $k$ .

The results in Fig. 3.6 show that link prediction can be effective at imputing intra-community edges, especially for sparse networks that have lower  $\mu$  values. The results also show that for networks with high  $\mu$  and  $\delta$  values, the top-scoring edges as predicted by all three link prediction algorithms contain a large percentage of inter-community links. While this demonstrates the feasibility of using link prediction to recover missing intra-community edges, we do not know how to set the parameters (e.g., the  $k$  value to use for partitioning the ranked edges) for real-world networks. We will return to this, but first we formalize the problem.



**Figure 3.6** Intra-edge Precision of Link Prediction. Precision plots of three link prediction algorithms: Adamic-Adar (left), Common Neighbors (middle), and Jaccard (Right) for various values of mixing parameter  $\mu$ : 0.1 (top), 0.3 (middle), and 0.5 (bottom). The X-axis corresponds to number of top- $k$  edges as scored by the link prediction algorithm as a percentage of the number of edges in the network. Intra-edge precision is on the y-axis.

Let  $G = (V, E)$  be the input network, and the set  $E_{\text{missing}} = (V \times V) \setminus E$  denote the set of

missing edges in the  $G$ . We formally define a link-predictor  $\mathcal{L}$  as a function that takes any pair of nodes  $(x, y)$  in  $E_{\text{missing}}$  and maps them to a real number.

$$\mathcal{L} : E_{\text{missing}} \mapsto \mathbb{R} \tag{3.3}$$

A community detection algorithm can be formally described as a function  $\mathcal{C}$  that takes as input any network  $G$  and produces a disjoint partition of the nodes  $\{C_1, C_2, \dots, C_k\}$ .

The most naïve algorithm for enhancing community detection consists of a few simple steps. First, score missing edges in  $G$  using  $\mathcal{L}$ . Next, select the top- $k$  missing edges according to the link-predictor and add these edges to  $G$ . Lastly, apply the algorithm  $\mathcal{C}$  to the imputed network. However, simply adding links with high scores for networks with large  $\mu$  and  $\delta$  values may be problematic, since many of these links can be inter-community, thereby having a negative effect on community detection.

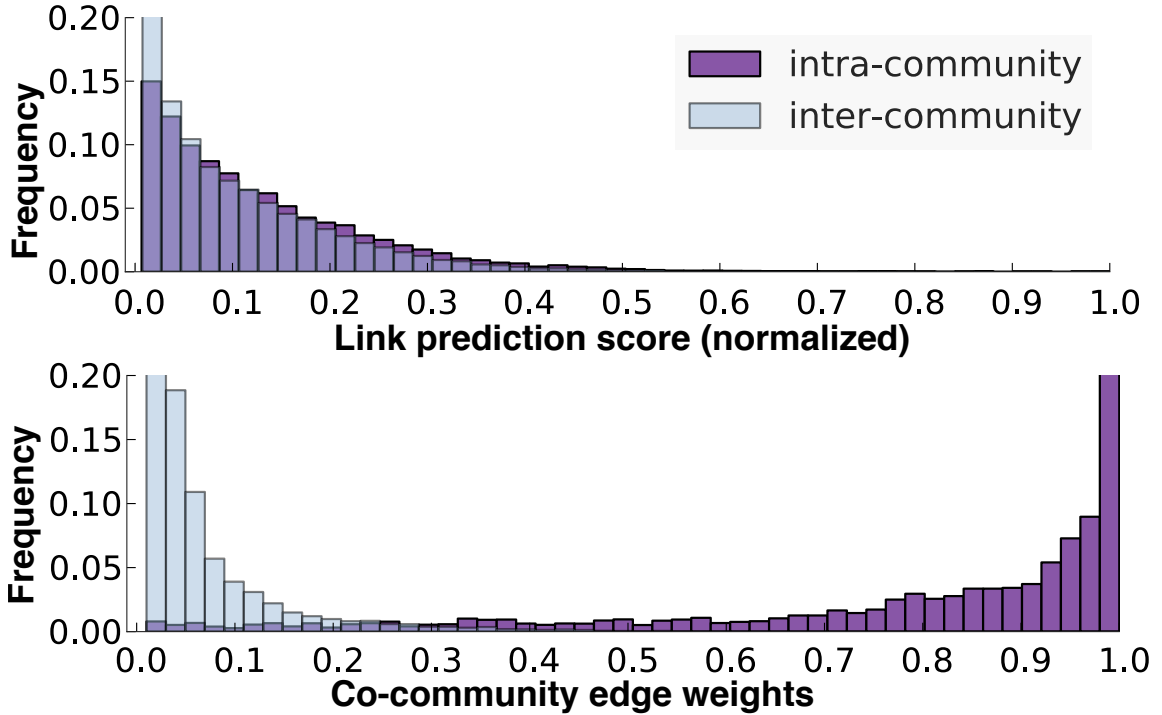
An intuition for why this naïve algorithm does not work is illustrated in the top histogram of Fig. 3.7. The plot shows the score distribution of both intra- and inter-community edges predicted by the AA link predictor on a randomly generated benchmark network. The distributions of the intra-community edges substantially overlaps with the inter-community distribution, thereby making any choice of a threshold for adding links not helpful for community detection. In addition, as this plot shows, the top- $k$  edges only comprise of a small percentage of the total set of intra-community edges. By simply selecting from the top- $k$  scoring edges, many of intra-community edges that are lower ranked will never be selected. As demonstrated in Fig. 3.6, the choice of  $k$  can have a significant impact on the quality of the edges, therefore selecting the right  $k$  becomes a challenge when the complexity and sparsity of the network is unknown.

### 3.4 Link Prediction Enhanced Consensus Clustering

Our core observation is that link prediction in both high- $\delta$  and high- $\mu$  settings is brittle: it can carry information, but for a single prediction is likely to be wrong. Therefore, we propose an improved method for applying link prediction to enhance community detection.

In order to mitigate the potential side effects of imperfect link prediction, we propose a sampling based algorithm that repeatedly applies link prediction to the input network. The EDGEBOOST pseudo code is shown in Algorithm 1 and proceeds in four steps. First, it uses a link prediction function to score missing edges and constructs a probability distribution over the set of missing edges (lines 2-3). The algorithm repeatedly samples a set of edges from this probability distribution, adding these sampled edges to the original network, and





**Figure 3.7** Edge Weight Distributions. Histogram of edge weights on a benchmark graph with  $\mu=0.4$  and 20% of the edges removed: scores from AA link predictor (top) and weights of co-community network (bottom).

runs community detection on the enhanced network (lines 5-7). Each iteration produces a new set of communities which are added to the set of partitions (lines 8-9). After the sample-detect-partition sequence is executed many times, we aggregate the overall set of observed partitions (line 11) to produce a final clustering.

### 3.4.1 Network Imputation

The network imputation component of EDGEBOOST uses the input network and a link prediction algorithm to produce a probability distribution over the set of missing edges. The number of edges sampled during each iteration of the imputation procedure (lines 5-6) is a uniform random number between 1 and the size of the input network. We experimented with many values of  $k$  and found this to work as well as when  $k$  was fixed.

We propose an imputation algorithm that constructs a distribution in which the probability of drawing an edge corresponds to its score produced by the link predictor. Missing edges that are scored higher by the link-predictor will have more probability mass than

---

**Procedure 1** EDGEBOOST Algorithm

---

**Input:** A network  $G = (V, E)$ , link-predictor  $\mathcal{L}$ , community detection algorithm  $\mathcal{C}$ , number of iterations  $n$

**Output:** A partition  $P^*$  of the vertices in  $G$

```
1:  $E_{\text{missing}} = \mathcal{L}(G)$  ▷ score edges in  $G$ 
2:  $\mathcal{D} = \text{IMPUTATION}(E_{\text{missing}})$  ▷ create edge distribution
3:  $P = []$  ▷ initialize list of partitions
4: for  $i \leftarrow 1, n$  do
5:    $k \sim U(1, |E|)$ 
6:    $\{e_1, e_2, \dots, e_k\} \sim \mathcal{D}$  ▷ sample  $k$  edges
7:    $G_i = (V, E \cup \{e_1, \dots, e_k\})$  ▷ impute  $G_i$ 
8:    $p_i = \mathcal{C}(G_i)$  ▷ cluster  $G_i$ 
9:    $P = P \cup p_i$ 
10: end for
11:  $P^* = \text{AGGREGATIONFUNCTION}(G, P)$ 
12: return  $P^*$ 
```

---

lower scoring edges. The probability function constructed from this process is:

$$P(X = x) = \frac{\mathcal{L}(x)}{\sum_{y \in E_{\text{missing}}} \mathcal{L}(y)} \quad (3.4)$$

Our imputation algorithm is more likely to pick higher scoring edges, which can result in a fairly accurate selection of intra-community edges as shown in the *Link-Enhanced Community Detection* section. At the same time, even low scoring edges have probability mass, which is important since for some networks, intra-community edges can also be low scoring.

### 3.4.2 Partition Aggregation

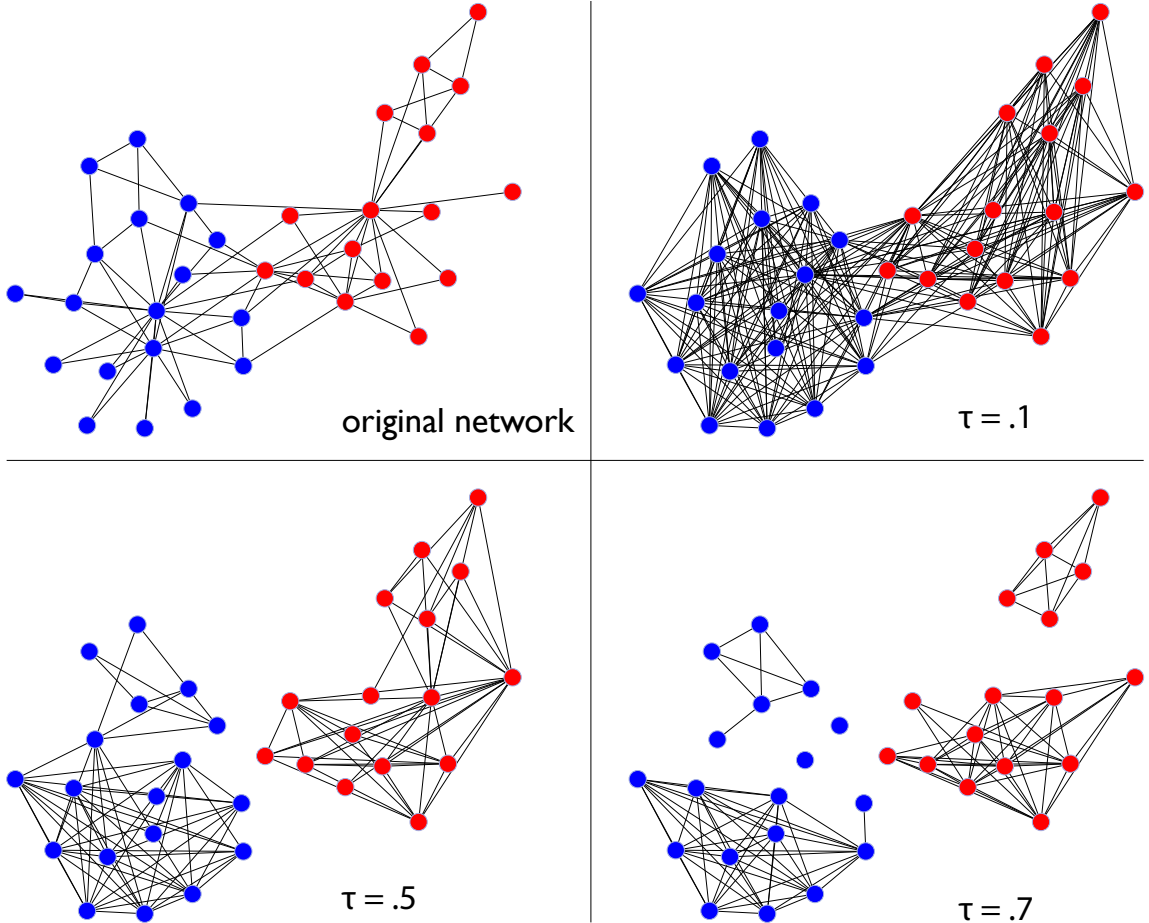
Having generated many possible “images” of our original graph via network imputation, we can apply community detection algorithms to each. Each execution of the algorithm produces a partition—possibly unique—based on the input graph. After generating many such partitions, we use *partition aggregation* to produce a final output. Previous ensemble clustering techniques [32, 94], construct a  $n \times n$  consensus matrix that represents the co-occurrence of nodes within the same community. The goal of such a data structure is to summarize the information produced by the various partitions. We propose a similar data structure, a co-community network  $G_{\text{cc}}$ , which consists of nodes from the input network and edges with weights that correspond to the normalized frequency of the number of times

the two nodes appear in the same community.

The  $G_{cc}$  graph is a transformation of the input network into one that represents the pairwise community relationships between nodes, rather than the functional relationships defined by the semantics of the input network  $G$ .  $G_{cc}$  links nodes that appear in the same community, and weights them based on frequency or co-occurrence (i.e., the edge between two nodes has a normalized weight equal to number of times the two nodes appear together in the same community over all partitions). Thus,  $G_{cc}$  exhibits community structure representing communities that appeared frequently in the input partitions. As shown in the lower plot of Fig. 3.7, there is a clear distinction between the intra-community and inter-community edge-weight distributions in  $G_{cc}$ . A simple mechanism for identifying a final “partitioning” is to remove all edges for which we have low confidence (i.e., inter-community edges) and study the resulting connected-components (CC). We parameterize the pruning with a threshold  $\tau$  and prune edges below that value. The semantics of the resulting graph is that all pairs of linked nodes have been seen in the same community at least  $\tau$  percentage of times and consequently all nodes captured in a CC maintain this guarantee.

Fig. 3.8 shows an example of a co-community network pruned at various thresholds. The network in this diagram is the famous Zachary’s karate club [179], the colors of the nodes denote the ground-truth community assignments of each node. The original network is shown in the upper left quadrant, and the remaining quadrants show the co-community network pruned at different thresholds. As we can see in the upper right quadrant, if we threshold at a small value of  $\tau$  we are almost certain to obtain a network with one large connected component. This is due to the fact that given enough iterations of the link prediction/community detection loop we are likely to find at least a few cases where nodes that would ordinarily fall into two communities are placed into the same one. At  $\tau = 0.5$  we see the CC’s reflect the community structure in the original network (the “correct partition”). As we increase the threshold the two true communities are further shattered into sub-communities, leaving some nodes completely isolated. One can interpret the connected components at these higher levels of  $\tau$  as capturing the *core members* of the true communities: members who co-occur with each other a very high percentage of time and do not co-occur often with nodes outside of their community.

While  $\tau$  may be set manually—appropriate for some applications when some level of confidence is desirable—there are other applications where we would prefer that this threshold be chosen automatically. As the last module of our framework we propose a way for selecting a  $\tau$  and constructing a final partitioning given that chosen value. Since the edge weights in  $G_{cc}$  correspond to the fraction of times two nodes appear in the same community, they are rational numbers. We can therefore enumerate all the possible values of



**Figure 3.8** Karate Club Co-community Network. Visualization of the co-community network for “Zachary’s karate club” network. Each panel shows the network pruned at various thresholds  $\tau$ .

$\tau, \{\frac{1}{n}, \frac{2}{n}, \dots, \frac{n}{n}\}$  on the interval  $[0, 1]$ . At each value of  $\tau$  we prune all edges with weights less than  $\tau$  and compute the partition of  $G_{cc}$  that corresponds to the connected-components. We then score this partition according to equation 3.6 and select the threshold and corresponding partition that maximizes this score. Our algorithm for automatically choosing  $\tau$  does add computational overhead as compared to simply selecting a  $\tau$  manually. We evaluate the selection of a fixed  $\tau$  threshold for the LFR networks in the experiments section.

In previous work, Monti *et al.* [113] propose a formula for computing the “consensus” score of an individual cluster. For a given community  $C_k$ , that is of size  $N_k$ , their score sums the co-community weights and divides it by the maximum possible weight.  $G_{cc}(i, j)$  corresponds to the fraction of times nodes  $i, j$  were grouped together in the same community.

$$m_k = \frac{1}{\binom{N_k}{2}} \sum_{\substack{i, j \in C_k \\ i < j}} G_{cc}(i, j) \quad (3.5)$$

We score a partition  $p_\tau$  parameterized by a threshold  $\tau$  by taking the weighted sum of the scores  $m_k$  for each community in the partition. We use a weighted sum because the score contribution of each community should be commensurate with its size.

$$S(p_\tau) = \frac{1}{N} \sum_{k \in p_\tau} N_k * m_k \quad (3.6)$$

If the final partition has any singleton nodes that do not belong to any community we connect each stray node to the community to which it has the highest mean edge weight to in the un-pruned co-community network.

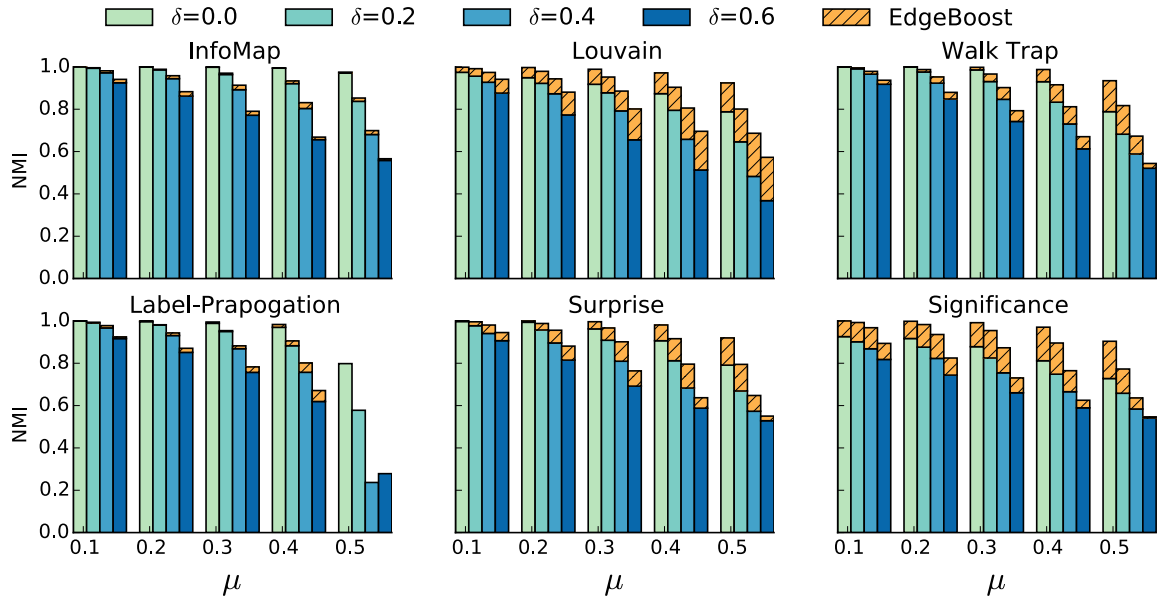
## 3.5 Evaluation

We have conducted a series of experiments to test EDGEBOOST on the LFR benchmark networks, standard real-world networks (e.g karate club), and a set of ego networks from Facebook. First, we present a comparison of EDGEBOOST with different community detection methods. Subsequent experiments include an analysis of various parameter settings of EDGEBOOST.

### 3.5.1 Comparing EDGEBOOST with Different Community Detection Methods

Similarly to the analysis we performed in the *Communities in Incomplete Networks* section, we evaluate our methods against the LFR benchmark over various settings of the mixing ratio  $\mu$  and the percentage of missing edges,  $\delta$ . In Fig. 3.9 we show the performance gain (striped yellow bars) of EDGEBOOST for six different community detection algorithms: InfoMap, Louvian, WalkTrap, Label-Propagation, Surprise, and Significance. The number of imputation iterations is fixed at 50 for both algorithms and the bars are generated by averaging over 50 randomly generated networks. While not shown in Fig. 3.9, we tested all 3 link prediction algorithms and did not find a substantial difference. Keeping with our link prediction analysis in the *Link Prediction for Enhancing Community Detection* section, Jaccard slightly outperformed the other methods, so we chose Jaccard as the link prediction algorithm for EDGEBOOST. We ran a Mann-Whitney U test for each parameter configuration and found that 90% of the results in Fig. 3.9 have a p-value less than 0.05.

We can see from Fig. 3.9 that our method improves performance for almost all input community detection algorithms. One exception is that EDGEBOOST shows a decrease in

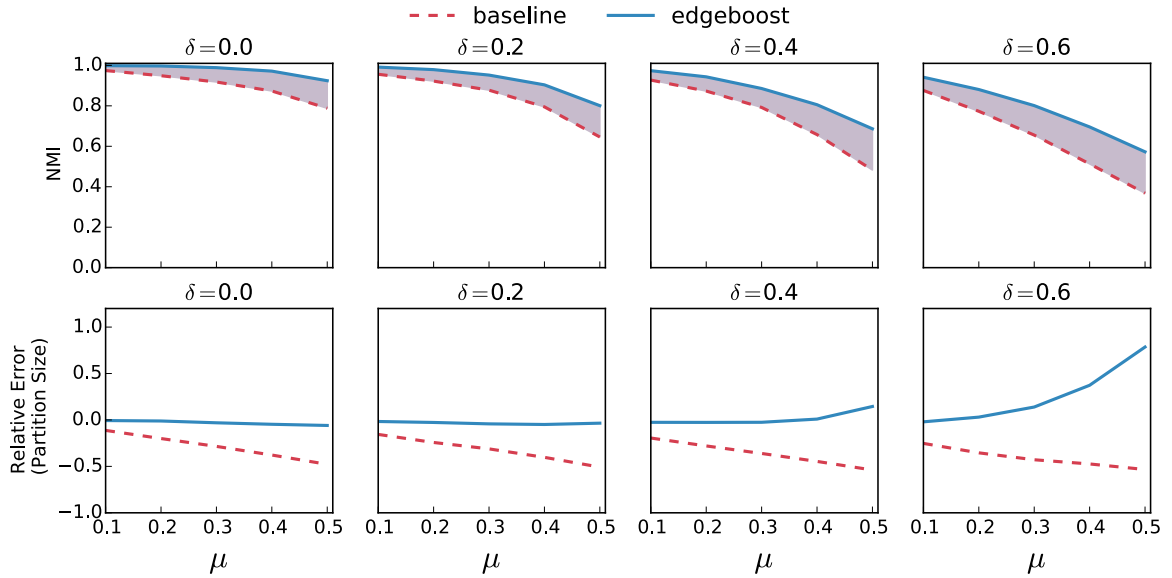


**Figure 3.9** EDGEBOOST Performance on LFR Networks. Performance of six popular community detection algorithms on the LFR benchmark networks. Dashed yellow bar shows the improvement of EdgeBoost over using the baseline community detection method.

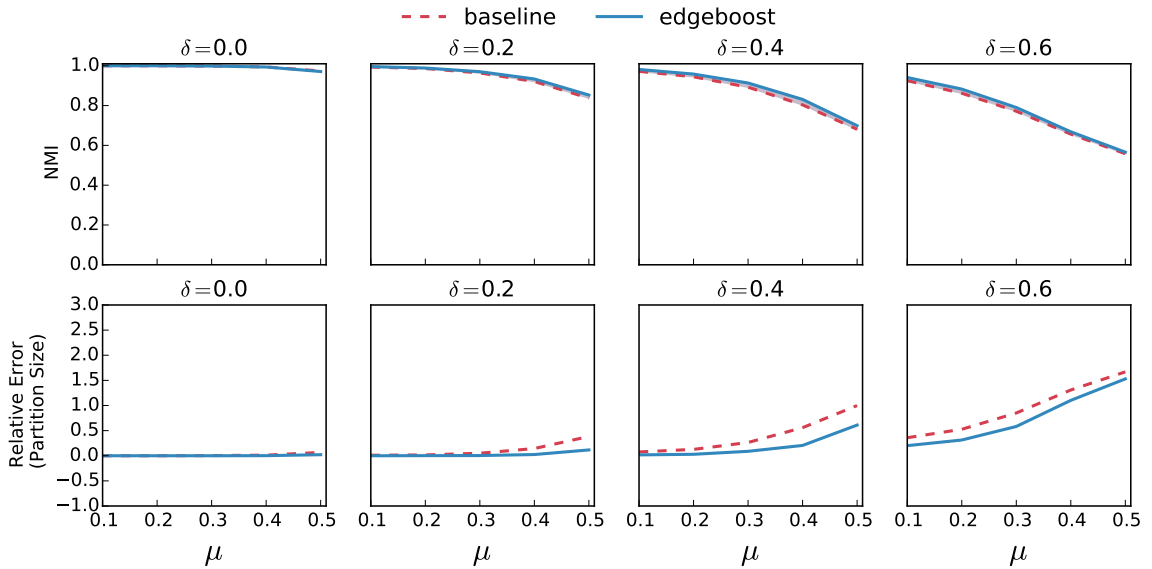
performance for the Label-Propagation algorithms at a  $\mu$  value of 0.5. As in other studies [94], the Label-Propagation algorithm’s performance becomes erratic at  $\mu$  values of 0.5 or greater, most likely due to the fact that Label-Propagation assumes that a node’s label should be chosen based on the labels of its neighbors. While EDGEBOOST is designed to work on stochastic algorithms, and variations of the input network, if an algorithm has too much variation, as is the case with Label-Propagation, it can lead to decreased performance.

Fig. 3.24 shows the performance gain of EDGEBOOST on the Louvain algorithm in more detail. As our previous analysis showed, the baseline Louvain algorithm tends to detect bigger communities on average than in the planted partition. The bottom row shows that for moderate values of  $\delta$ , EDGEBOOST is able to recover the smaller communities in the planted partition. At very high values of  $\delta$  ( $\geq 0.4$ ) a network may be so sparse that the perfect recovery of correct communities is most likely not possible. Even for these high  $\delta$  values, EDGEBOOST still shows an improvement in NMI over the baseline method. The Louvain algorithm shows similar performance gains as a function of the  $\delta$  parameter but as seen in Fig. 3.9, other algorithms show more variation with respect to  $\delta$ . Figures ?? show analogous plots for the five other algorithms we tested.

The LFR benchmark captures certain network properties, but it is an imperfect model of real-world networks. To test EDGEBOOST on real network data, we also performed experiments on two additional data sets. The first data set consists of a suite of standard networks

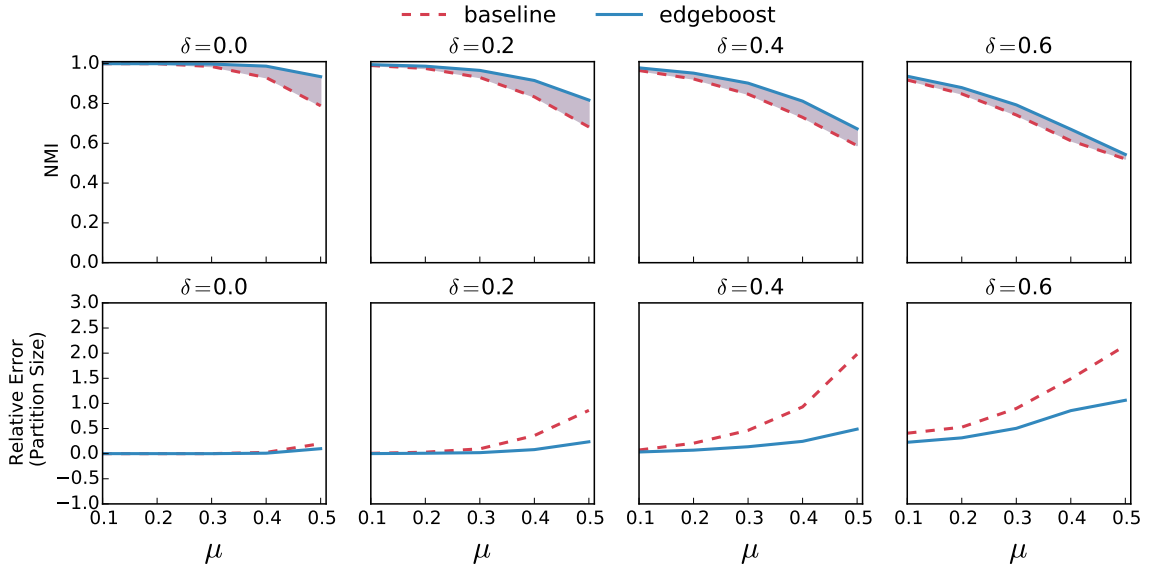


**Figure 3.10** EDGEBOOST Paired With Lovain. Performance of EdgeBoost (solid) and the baseline Louvain algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of edge boost for NMI. The bottom row of plots shows the relative error of the partition size.

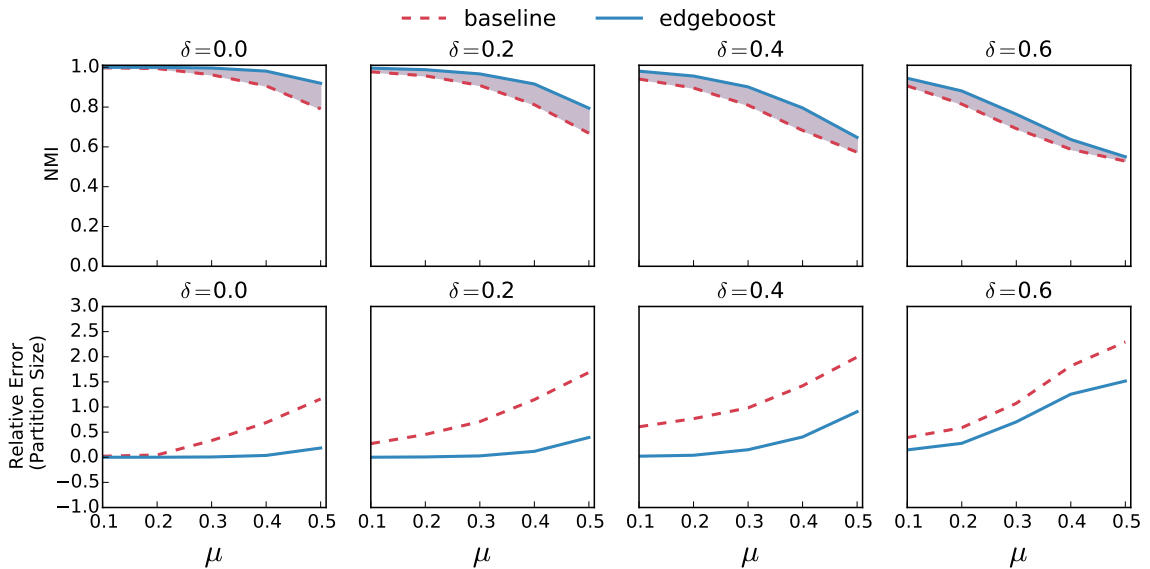


**Figure 3.11** EDGEBOOST Paired With InfoMap. Performance of EDGEBOOST (solid) and the baseline InfoMap algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size.

for benchmarking community detection. The data set includes: Zachary's *Karate Club* network (Karate) [179], network of political books (Books) [118], blog network (Blogs) [4] and the American college football network (Football) [58]. All of these networks have a ground truth partition such that we can use NMI to evaluate the performance of community detection. Fig. 3.16 shows the results of EDGEBOOST on each of the four networks with



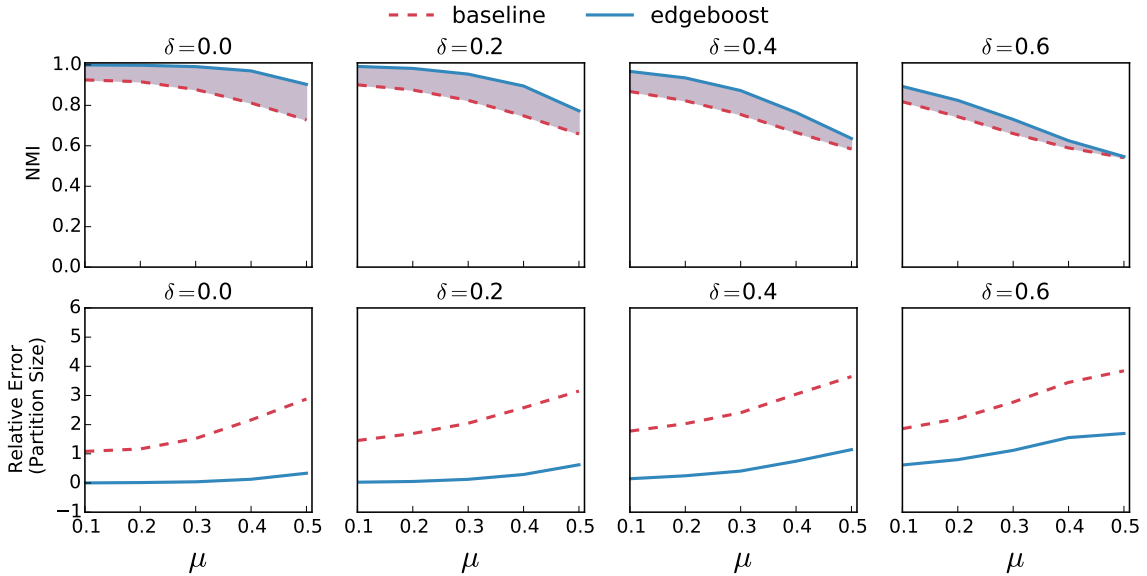
**Figure 3.12** EDGEBOOST Paired With WalkTrap. Performance of EDGEBOOST (solid) and the baseline WalkTrap algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size.



**Figure 3.13** EDGEBOOST Paired With Surprise. Performance of EDGEBOOST (solid) and the baseline Surprise algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size.

the same six input community detection algorithms used in the experiment above. In all but three of the 24 algorithm/network configurations, EDGEBOOST improves performance by an average of 14%. On the Football network, EDGEBOOST does worse with the InfoMap, Label-Propagation, and WalkTrap algorithms, but decreases performance by only an aver-

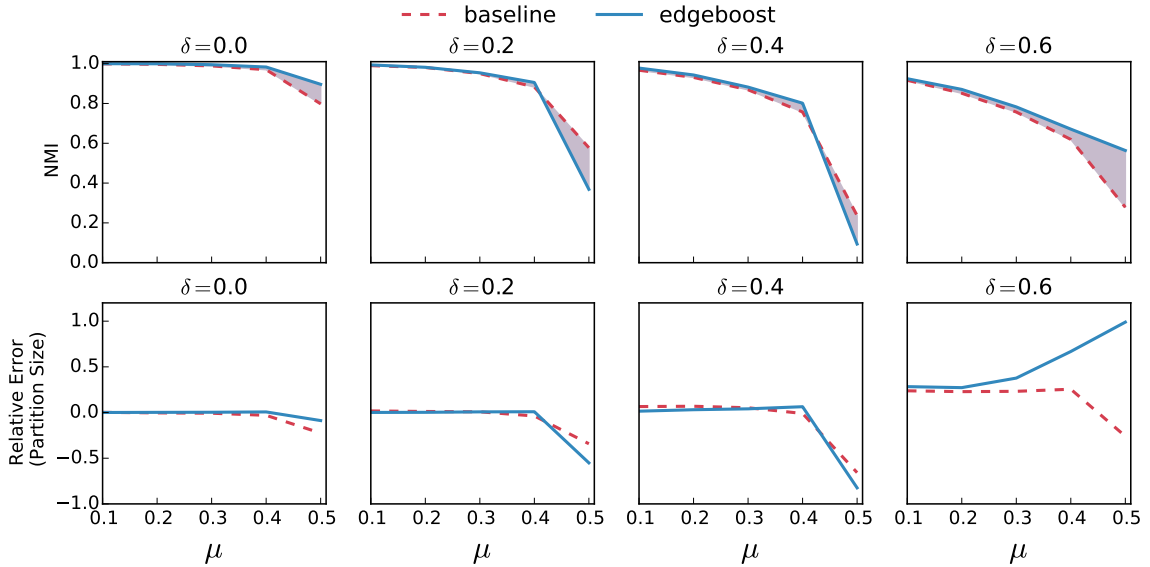




**Figure 3.14** EDGEBOOST Paired With Significance. Performance of EDGEBOOST (solid) and the baseline Significance algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size.

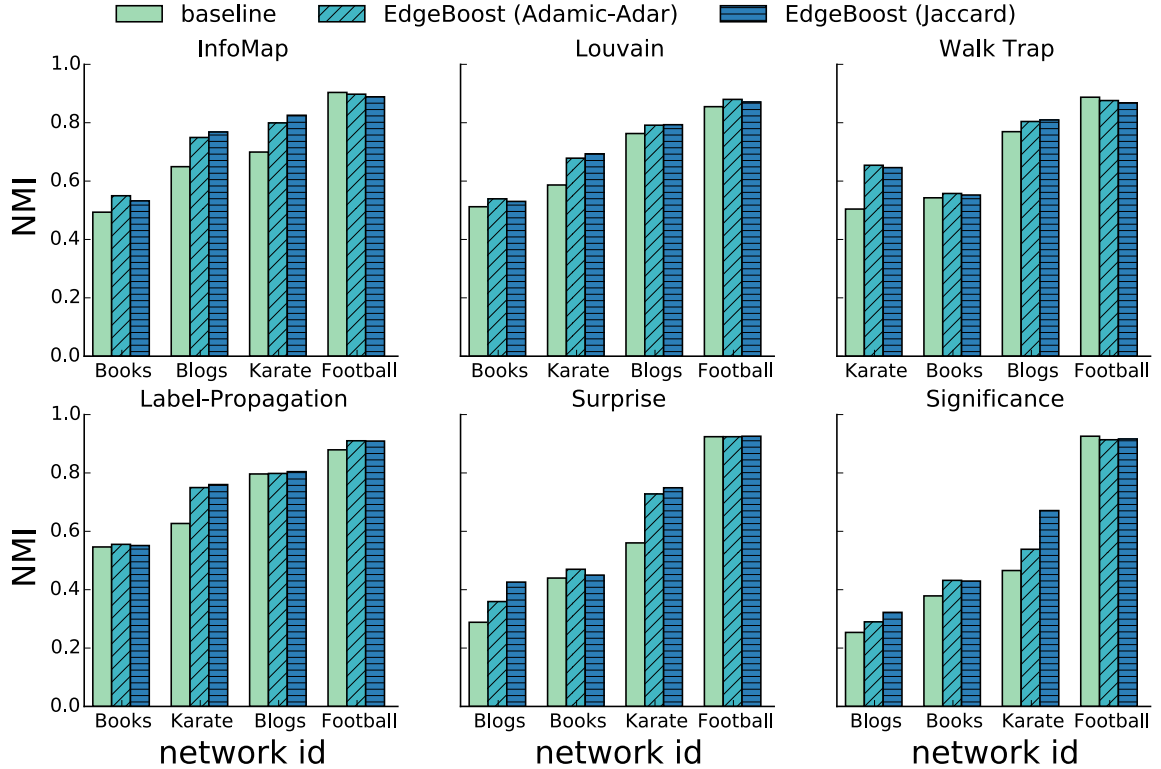
age of 1.6%. Overall, these datasets give some assurance that EDGEBOOST can improve performance on real networks.

We also tested EDGEBOOST on a data set of Facebook ego-networks [108] that capture all neighbors (and their connections) centered on a particular user. The data set described in the original paper by McAuley *et al.*, consists of networks from three major social networks: Facebook, Google+ and Twitter. The Facebook data set is likely the highest quality of the three; it contains ground-truth which was obtained from a user survey that had the ego users for each network provide community labels. The ground truth for the ego-networks from Twitter and Google+ is lower quality since it was obtained by crawling the publicly available lists created by the ego user. As such, for many of the networks, the ground truth consisted of only a small fraction of nodes in the network and for many networks the ground truth consisted of lists with very few members. Since the target of this chapter is non-overlapping and complete clustering, we chose to not use the Twitter and Google+ networks due to the sparsity of their ground-truth. The Facebook networks have complete ground-truth labeling, so we used those for evaluation. Despite the Facebook networks being the highest quality of the three datasets, it still contained ground-truth communities of 1-2 users. We pre-processed each network by removing all ground-truth communities with fewer than three nodes. We have included a plot (Figure 3.17) of the distribution of community sizes for the Facebook networks in the appendix.



**Figure 3.15** EDGEBOOST Paired With Label-Propagation. Performance of EDGEBOOST (solid) and the baseline Label-Propagation algorithm (dashed) on LFR benchmarks. The purple shaded region shows the improvement of EDGEBOOST for NMI. The bottom row shows the relative error of the partition size.

The ground-truth for the Facebook ego-networks can contain overlapping communities, therefore we cannot directly use the standard version of NMI for evaluation. To test EDGEBOOST on overlapping ground-truth data we use the NMI extension proposed by Lancichinetti *et al.* [95] that supports comparison of overlapping communities. Fig. 3.18 shows the results of using EDGEBOOST with the same six community detection algorithms used in the LFR experiments. The solid bars represent the performance of the baseline community detection algorithm without EDGEBOOST. The diagonal and horizontal striped bars shows the results from EDGEBOOST paired with the Adamic-Adar and Jaccard respectively. We set the number of iterations for EDGEBOOST at 50. Each bar was generated by averaging the NMI score over 100 runs of the baseline and EDGEBOOST paired with the Jaccard and Adamic-Adar link predictors. EDGEBOOST shows an improvement on most networks for each of the six community detection algorithms; this result is consistent with our experiments on the LFR benchmark. On the LFR benchmark networks EDGEBOOST paired with Jaccard link prediction was consistently better than the other link prediction methods but this is not consistently the case on the Facebook networks. Jaccard outperforms the Adamic-Adar most of the time, but there are some cases when the opposite is true. While EDGEBOOST shows improvement for most combinations of algorithms and network, there are some instances when the performance of EDGEBOOST is lower than baseline. Overall, in 52 of the 60 total configurations EDGEBOOST improves performance by an average of



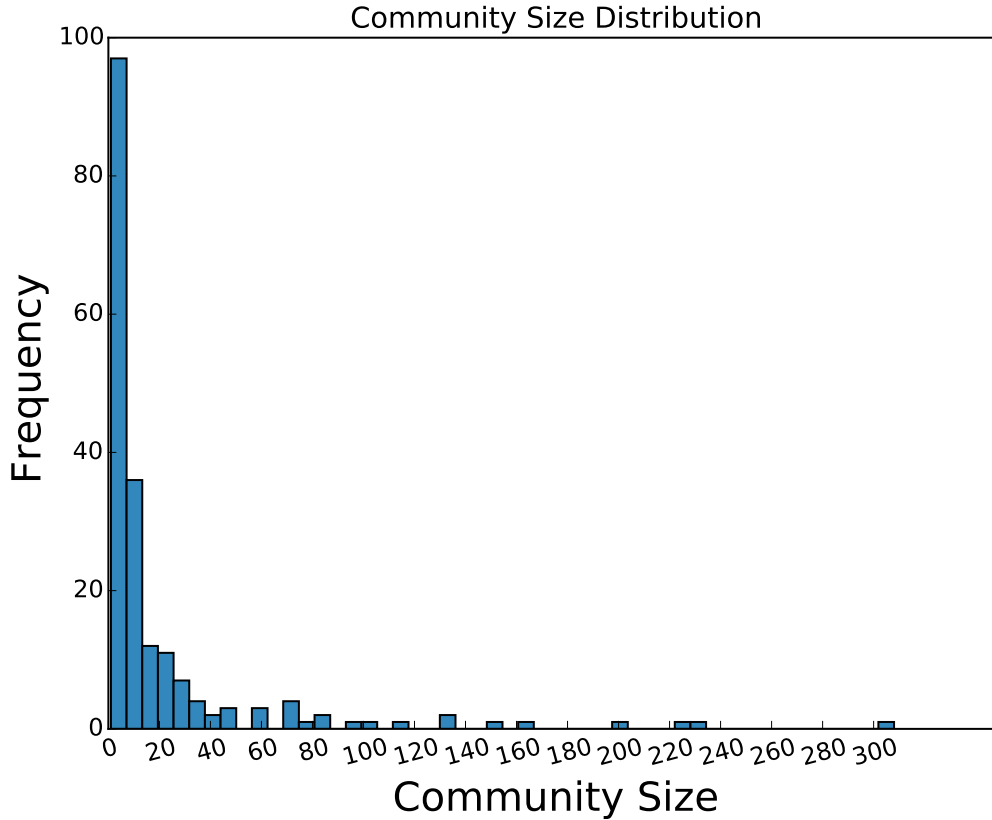
**Figure 3.16** Performance of EDGEBOOST on Standard Network Datasets. Comparison of EDGEBOOST on set of standard real network benchmarks community detection

21%. In the rare configurations (8 out of 60) when EDGEBOOST performs worse than baseline, EDGEBOOST performs only 5% worse on average.

### 3.5.2 Varying the Parameters of EDGEBOOST

In addition to comparing EDGEBOOST using different community detection algorithms we also analyzed how the performance varies with respect to different parameter settings. For these experiments, the curves were generated by averaging over 50 networks generated via the LFR benchmark. Figs. 3.19 and 3.20 show the convergence of the Louvain and InfoMap algorithms, as a function of the “number of community detection iterations” (*NumIterations*). Most of the performance gain from EDGEBOOST can be had with *NumIterations* set to 10, and setting the number of iterations beyond 50 does not give much benefit. The convergence of EDGEBOOST is qualitatively similar for low and high values of  $\mu$  and the entire range of  $\delta$  values.

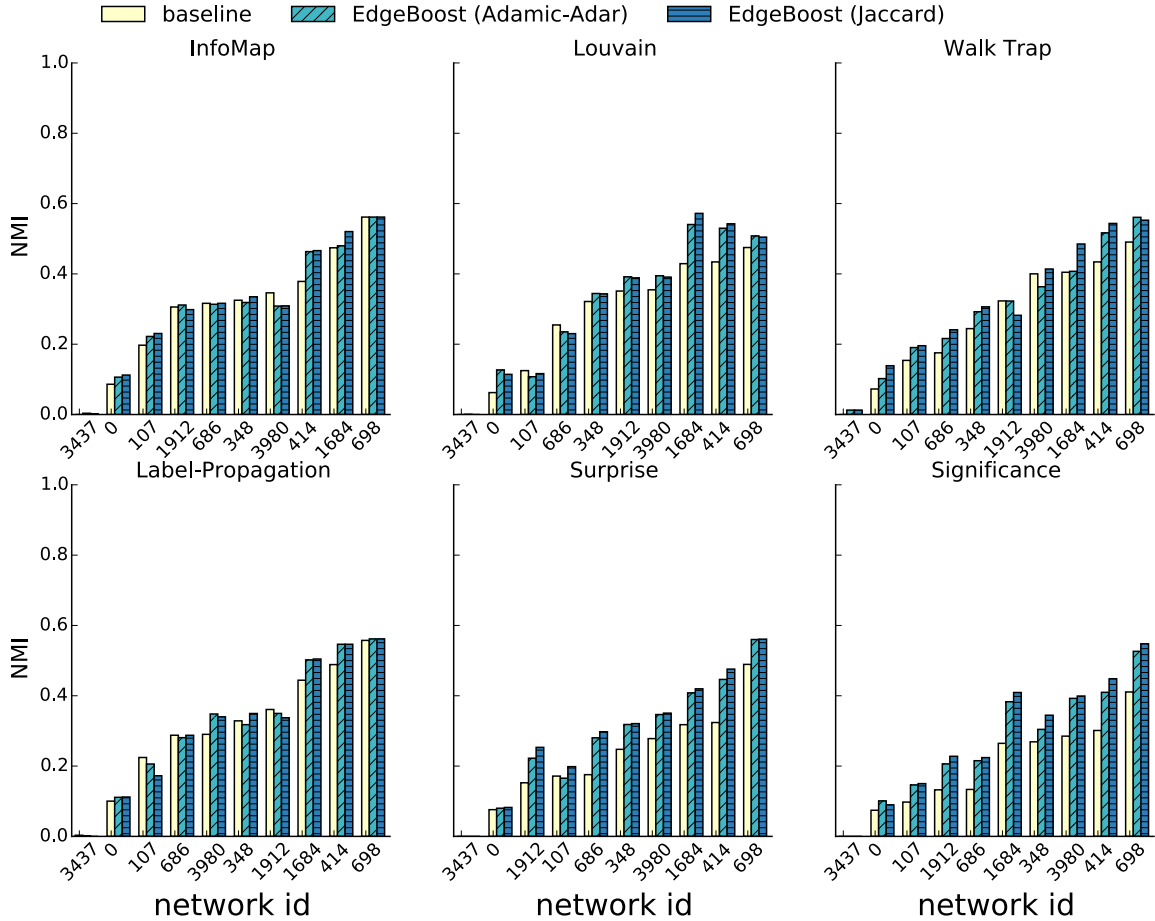
In the *Partition Aggregation* section we propose a method for automatically selecting the co-community threshold  $\tau$ , which we have used for all of the previous experiments. Since



**Figure 3.17** Distribution of community sizes for Facebook ego networks. Nodes were given community labels by ego users as part of a user study.

the selection of  $\tau$  is the most computationally expensive part of the entire EDGEBOOST pipeline, we present an analysis of how EDGEBOOST performs with a manual selection of  $\tau$ . Figs. 3.21 and 3.22 show how EDGEBOOST performs by varying the selection of  $\tau$  for EDGEBOOST paired with Louvain and InfoMap respectively. For both algorithms, EDGEBOOST can achieve good performance for values of  $\tau$  in the range 0.6-0.9, indicating that manual  $\tau$  selection can be an effective way to save computational resources and still boost performance over baseline. For higher values of  $\mu$ , the performance of EDGEBOOST is more dependent on  $\tau$ , especially for the Louvain algorithm. Since the Louvain algorithm performs less reliably for higher  $\mu$  values, the co-community network has noisier edge weights, therefore making the selection of  $\tau$  more critical to achieving good performance.

In conclusion, if the user is computationally constrained, simply selecting a manual threshold (or a few thresholds) can give good results without requiring the costly step of computing connected components at each threshold. A potential pitfall of manually selecting a threshold, is that EdgeBoost can give degenerate solutions. Degenerate partitions—those that put all nodes in one cluster or creating hundreds of small clusters—result from the

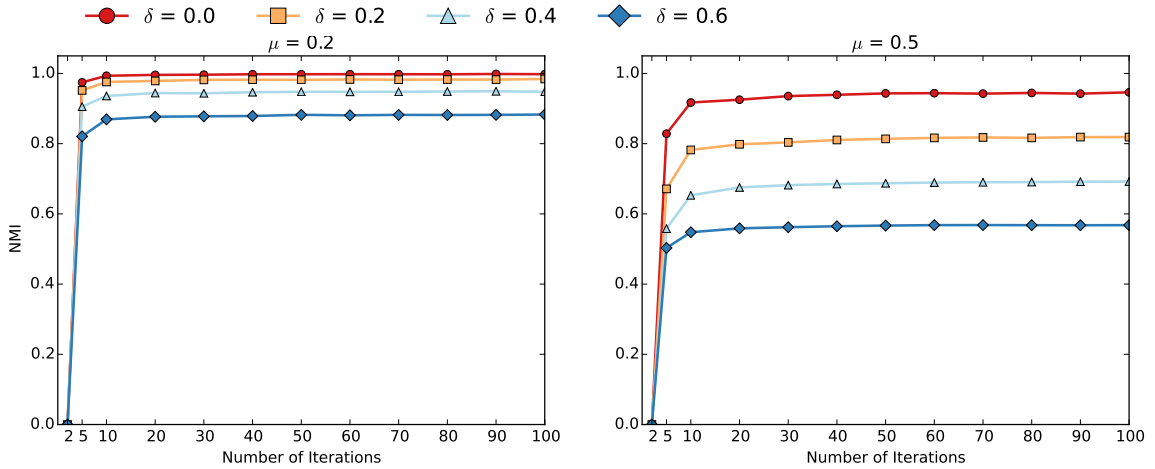


**Figure 3.18** Performance of EDGEBOOST on Facebook Networks. Comparison of EDGEBOOST on ego-networks from Facebook

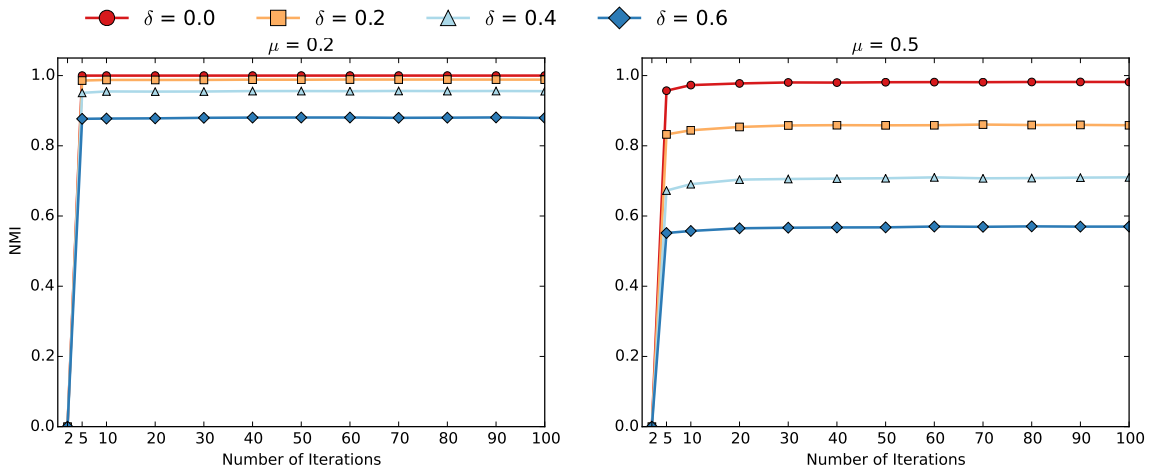
threshold value being too small or large respectively. For applications with a user in the loop, these degenerate solutions are easily detected and fixed by increasing or decreasing the threshold. The automatic threshold finder is intended for applications where full automation is required.

### 3.5.3 Runtime Analysis

The most computationally expensive module of EDGEBOOST is the aggregation algorithm which requires the computation of connected components at various thresholds. The complexity of computing connected components is worst case  $O(|E|)$ , where  $|E|$  is the number of edges in the network. The aggregation module computes the connected components on the co-community network, which can be much denser than the input network. In theory it is possible for the co-community network to have  $O(n^2)$  number of edges, therefore making the

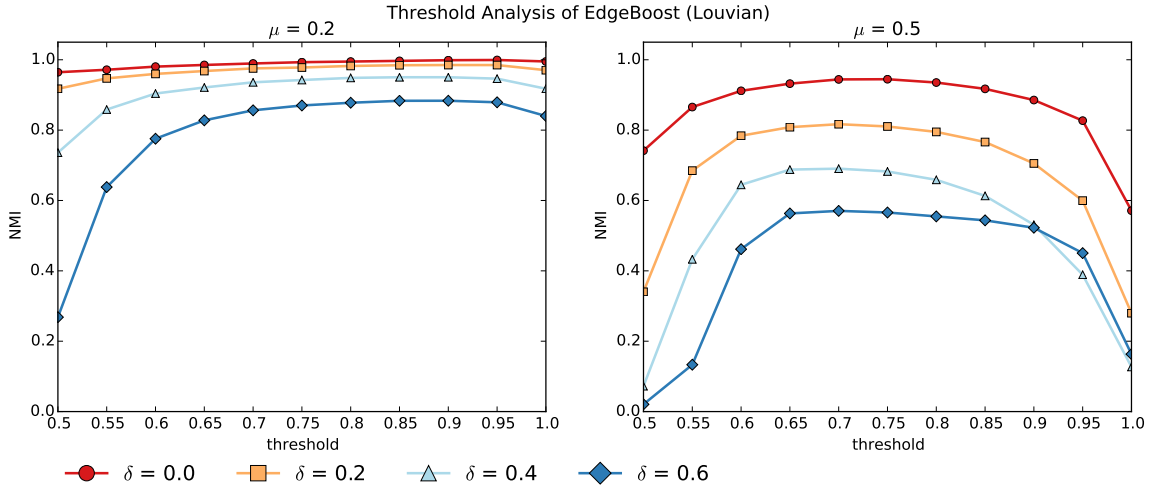


**Figure 3.19** Varying *NumIterations* for EDGEBOOST with Louvain. The parameters are set as follows:  $\mu = 0.2$  (left) and  $\mu = 0.5$  (right) over  $\delta$  values ranging from 0.0 to 0.6.

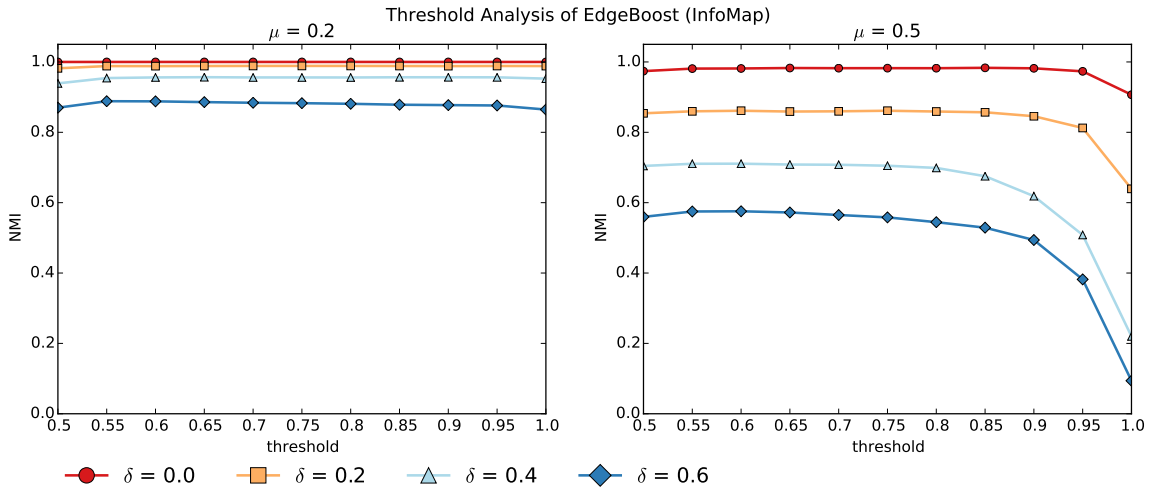


**Figure 3.20** Varying *NumIterations* for EDGEBOOST with InfoMap. The parameters are set as follows:  $\mu = 0.2$  (left) and  $\mu = 0.5$  (right) over  $\delta$  values ranging from 0.0 to 0.6.

aggregation module computationally expensive. In order to show that EDGEBOOST scales well when increasing to large networks we ran it on LFR networks of various sizes, ranging from 1000 to 128000 nodes. Fig. 3.23 shows the run time of EDGEBOOST with Louvain and Jaccard link prediction with the number of iterations set to 10. While EDGEBOOST does have a significant time overhead over Louvain, it still scales in the same manner as Louvain. There are also many components of EDGEBOOST’s pipeline that can be naively parallelized. The creation and clustering of the imputed networks are all independent of each other and can be done in parallel. In addition the process of identifying the  $\tau$  threshold can also be sped up by finding the connected components for various threshold values in parallel.



**Figure 3.21** Varying  $\tau$  for EDGEBOOST with Louvain. Varying the co-community threshold ( $\tau$ ) for EDGEBOOST with  $\mu = 0.2$  (left) and  $\mu = 0.5$  (right) over  $\delta$  values ranging from 0.0 to 0.6.

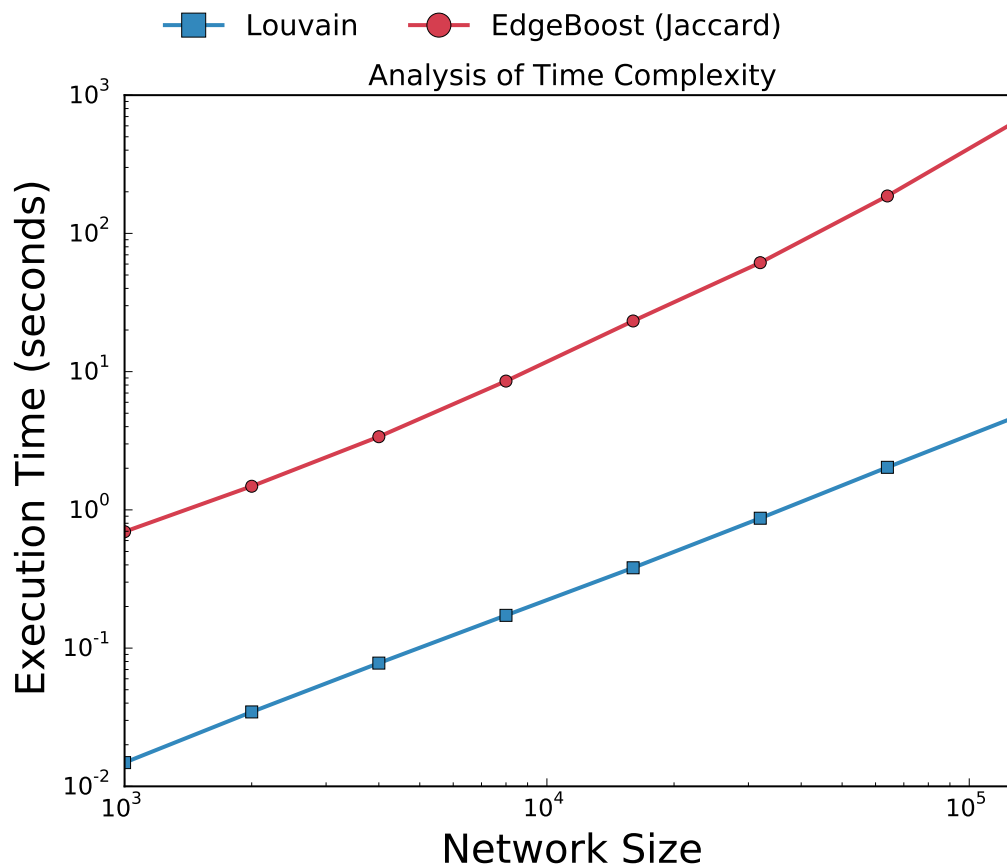


**Figure 3.22** Varying  $\tau$  for EDGEBOOST with InfoMap. Varying the co-community threshold ( $\tau$ ) for EDGEBOOST with  $\mu = 0.2$  (left) and  $\mu = 0.5$  (right) over  $\delta$  values ranging from 0.0 to 0.6.

### 3.6 Applying EDGEBOOST to Survey Generation

Network-based methods for the problem of text summarization have seen great success. In these methods, networks are used to model sentences as nodes and edges as the similarity between sentences based on a chosen metric. Clustering sentences — before applying a ranking algorithm such as LexRank [41] — has been shown to improve the quality of a summary by increasing the diversity of topics. Graph-based clustering has therefore become an integral component of many summarization systems [84, 112, 137].

The standard graph clustering pipeline for summarization consists of either clustering the similarity matrix directly (as a weighted network)[85] or to prune the similarity matrix by



**Figure 3.23** Analysis of Execution Time. Comparison of the runtime between EDGEBOOST and baseline Louvain algorithm on networks ranging from size 1000 to 128000 nodes. EDGEBOOST has the *NumIterations* set to 50.

selecting a threshold and deleting all edges with weights less than that threshold[135]. Both of these approaches are deficient and can lead to sub-optimal summaries. Since sentence similarity matrices are usually dense, with  $O(n^2)$  number of edges, clustering the network directly even with weights can lead to clusters that are too large. Pruning edges below a specified threshold, can lead to better clusters but the clustering results are sensitive to the chosen threshold, which is often chosen heuristically.

In this section we propose an extension of EDGEBOOST as a more robust way of clustering sentence networks. We replace the link prediction component of EDGEBOOST with a module that simply enumerates threshold values over a specified range. At each threshold, we cluster the graph and process the clusters using the same aggregation function used in the original version of EDGEBOOST. This extension of EDGEBOOST removes the need to heuristically define a threshold.

For testing this extension, we use the data for scientific topic summarization presented



in [83]. This dataset consists of summarization data for 7 topics. For each topic, the summarization input consists of introductory sections from 20 relevant papers on the topic. The input sentences are annotated with factoids extracted from human surveys for each topic, making it easy to use pyramid evaluation to compare output of different systems [117]. The seven topics along with input size for each topic are shown in Table 3.1.

### 3.6.1 Graph-based Summarization Algorithms

We compare the performance of EDGEBOOST to C-LexRank [135] and LexRank [41]. We now describe each of these models.

**LexRank** LexRank is a network-based content selection algorithm that serves as a baseline for our experiments. Given an input set of sentences, it first creates a network using these sentences where each node represents a sentence and each edge represents the tf-idf cosine similarity between the sentences. We can treat the sentence similarities as edge weights and use the adjacency matrix as a transition matrix after normalizing the rows; the formula for LexRank is thus:

$$\frac{1-d}{N} + d \sum_{v \in \text{adj}[u]} \frac{\text{cos}(u,v)}{\text{CosSum}_v} p(v)$$

Where  $\text{cos}(u,v)$  gives the tf-idf cosine similarity between sentence  $u$  and  $v$  and  $\text{CosSum}_v = \sum_{z \in \text{adj}[v]} \text{cos}(z,v)$ . The power method [121] can be used to efficiently solve the above equation to obtain the LexRank value for each sentence.

**C-LexRank** C-LexRank is a clustering-based summarization system that was proposed by [135] to summarize different perspectives in citing sentences that reference a paper or a topic. To create summaries, C-LexRank constructs a fully connected network in which vertices are sentences, and edges are cosine similarities calculated using the tf-idf vectors of citation sentences. It then employs graph clustering algorithm to find communities of sentences that discuss the same scientific contributions. The original implementation of C-LexRank uses a hierarchical agglomeration clustering algorithm proposed by [29]. We also implemented C-LexRank with a more current method, the ‘‘Louvain’’ algorithm, proposed by [13]. Once the graph is clustered and communities are formed, the method extracts sentences from different clusters to build a summary. It iterates through the clusters from largest to smallest, choosing the most salient sentence of each cluster, until the summary length limit is reached.

Topic	# Sentences
dependency parsing	487
named entity recognition	383
question answering	452
semantic role labeling	466
sentiment analysis	613
summarization	507
word sense disambiguation	425

**Table 3.1** List of seven NLP topics used in our experiments along with input size.

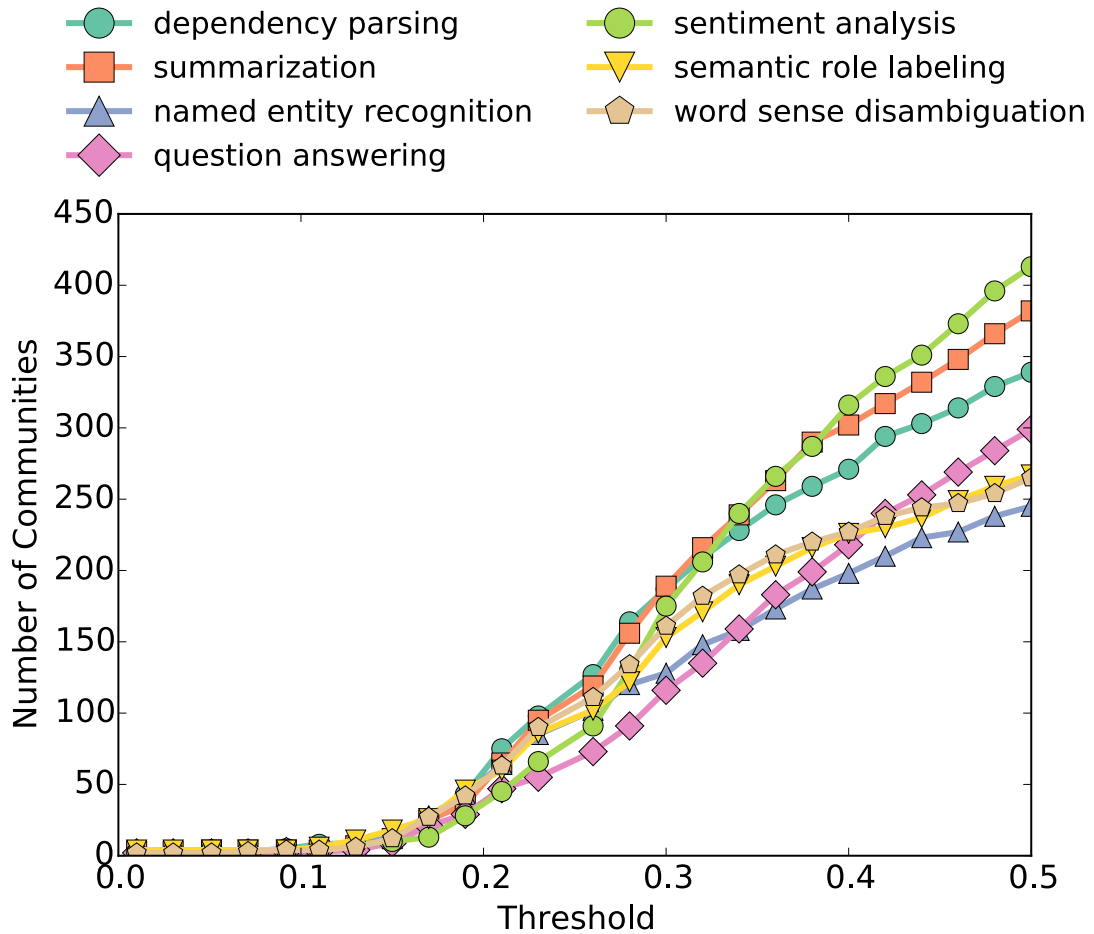
The salience of a sentence in its cluster is defined as its LexRank value in the lexical network formed by sentences in the cluster.

### 3.6.2 Combining EDGEBOOST with C-LexRank

Since most sentences have some degree of similarity, the similarity network is dense with most nodes being connected to each other. These spurious links can lead to the detection of overly large communities, which don't necessarily reflect topically cohesive groups of sentences. By selecting a threshold, and pruning edges below that threshold, many of the spurious edges can be eliminated, thereby resulting in better clusters. Choosing this threshold is tricky, since different thresholds can lead to very different clustering results. Figure 3.24 shows how the number of communities can vary with respect to the specified threshold. The x-axis represents a pruning threshold and the y-axis shows the number of clusters that result after applying the Louvain algorithm on the pruned network. Each curve, represents the results for 7 different networks, one for each topic in our dataset. We modify EDGEBOOST by removing the link prediction component and instead generate a set of clusters by varying the pruning threshold. For our experiment we set the threshold between 0.01-0.5 in increments of 0.01.

### 3.6.3 Survey Generation Experiment

For evaluating our content models, we generated 2,000-character-long summaries using each of the techniques (LexRank, C-LexRank,EDGEBOOST) for each of the topics. The summaries are generated by ranking the input sentences using each content model and picking the top sentences till the budget of 2,000 characters is reached. Each of these summaries is then given a pyramid score [117] computed using the factoids assigned to each



**Figure 3.24** Number of clusters vs. threshold

Topic	LexRank	C-LexRank	EDGEBOOST
dependency parsing	0.47	0.76*	0.72
named entity recognition	0.80	0.89*	0.84
question answering	0.65	0.67	0.86*
sentiment analysis	0.64	0.62	0.69*
semantic role labeling	0.75*	0.67	0.72
summarization	0.52	0.75*	0.71
word sense disambiguation	0.78	0.66	0.81*
<b>Average</b>	<b>0.66</b>	<b>0.72</b>	<b>0.76*</b>

**Table 3.2** Pyramid scores obtained by different content models for each topic along with average scores for each model across all topics. The best performing method has been annotated with a \*

sentence.

For the pyramid evaluation, the factoids are organized in a pyramid of order  $n$ . The top tier in this pyramid contains the highest weighted factoids, the next tier contains the second highest weighted factoids, and so on. The score assigned to a summary is the ratio of the sum of the weights of the factoids it contains to the sum of weights of an optimal summary with the same number of factoids. Pyramid evaluation allows us to capture how each content model performs in terms of selecting sentences with the most highly weighted factoids. Since the factoids have been extracted from human-written surveys and tutorials on each of the topics, the pyramid score gives us an idea of the survey-worthiness of the sentences selected by each content model.

Table 3.2 shows the pyramid scores of all the methods for each of the 7 topics. The best performing method on average is EDGEBOOST with a mean score of 0.76. EDGEBOOST obtains a dramatically better score on the topics “word sense disambiguation”, “question answering” while giving good performance on the other topics. Each of the baseline methods performs the best on at least one of the topics, but EDGEBOOST still outperforms the leading method C-LexRank by 4% on average.

For some networks, clustering may not be the performance bottlenecks, therefore making any cluster improvements provided by EDGEBOOST not reflected in the final score. There are cases when both C-LexRank and EDGEBOOST obtain worse scores than the baseline LexRank indicating that for some networks, clustering may not be an appropriate choice. These results suggest that a larger evaluation dataset is needed in order to characterize the kinds of networks that EDGEBOOST can perform better on. One added benefit of EDGEBOOST is that it takes away the need for a user to specify a pruning threshold for a network, therefore making the implementation of the algorithm easier and more robust for the end user.

### **3.7 Discussion and Future Work**

As shown in our experiments, EDGEBOOST is able to make bigger improvements for certain community detection methods than for others. Our hypothesis for why EDGEBOOST works better for certain algorithms has to do with the type of objective functions used by a given method. Objective functions such as modularity are less robust to missing edges in a network because the presence of direct links between nodes in a community are computed directly in the objective. In contrast, the MAP objective function used by the InfoMap algorithm relies on evaluating the community structure based on random walks, and is therefore less affected

by direct links between nodes in a community. Our hypothesis is that objectives such as the MAP equation are harder to improve with EDGEBOOST because their objectives are already more robust to missing edges. Despite the differences between objective functions, EDGEBOOST is still able to show an improvement for most community detection methods for both LFR benchmark and real-world networks.

Our modified LFR benchmark used random edge deletion to model missing edges in networks. While we chose this model because we think that it is the most generally applicable, there are other possibilities for modeling missing edges. One area of future research is to see how community detection algorithms are affected using different edge deletion strategies. Further experiments are necessary to determine if our techniques withstand biased edge removal, but we believe that repeated link prediction will nonetheless boost performance. Further, if missing intra-community edges could be modeled more accurately, development of better link prediction algorithms for enhancing community detection may be possible.

In order to increase the quality of community detection, EDGEBOOST trades off time and space efficiency. The construction of the co-community network can be memory intensive because it is likely to be much denser than the input network. In addition, EDGEBOOST requires many runs of a sometimes costly community detection algorithm. While EDGEBOOST can scale to reasonably large networks (see the *Runtime Analysis* section), we acknowledge these trade-offs and emphasize that EDGEBOOST is not designed for million node networks. Instead it was designed for use on small and medium networks (i.e., ego-networks, citation networks), in which data sparsity problems are common, and communities reflect meaningful structures in the data.

While we have shown the efficacy of EDGEBOOST in computing better partitions, it is possible that the approach can also improve other types of community analysis. Given different thresholds for which we can prune the co-community network (see the *Partition Aggregation* section) and the corresponding set of connected components, we can obtain a set of communities with a specified confidence. Some applications may not require a complete partitioning of nodes and may even be better suited with an incomplete partition which has higher quality communities. In future work we would also like to see how EDGEBOOST can be used in the detection of overlapping and/or hierarchical communities. This extension would require a different aggregation function as our current method is only capable of creating strict partitions, via computing connected-components.

The link-predictors tested in this chapter are all based on shared neighbors, and therefore are only capable of inferring missing connections between nodes that are at maximum 2 hops from each other. One issue with predicting links that are further apart is the computational complexity, since most of the metrics that are not neighborhood based are based off the

number of shortest paths between pairs of nodes. While not presented in this chapter, we experimented with the local path index proposed by [105], which predicts links between nodes that are as far as 3 hops from each other, but did not see any noticeable improvement. Other link predictors that we did not explore are those that utilize node attributes (*e.g.*, school and city) and/or link structure to score missing edges. Since most of the methods in disjoint community detection do not account for node attributes, in the future EDGEBOOST could be a robust way to integrate node attributes into existing algorithms.

### 3.8 Related Work

Though single-technique community detection is by far the most common, a number of recent projects have proposed ensemble techniques [5, 32, 94]. Aldecoa *et al.* describe an ensemble of partitions generated by different community detection algorithms, which differs from our approach of using the same algorithm and creating the ensemble by creating different networks. Both [32] and [94] present techniques for consolidating partitions generated by repeatedly running the same stochastic community detection algorithm. We implemented both of their methods but neither was suitable for consolidating clusters in our system; this is most likely because the partitions generated from our system have more variation than partitions generated from multiple runs of a stochastic algorithm. At a high-level, our proposed technique is a type of ensemble. Most ensemble solutions take the network as-is and assume that a “vote” between algorithms will produce more correct clusters. While this may work in some situations, bad input will often reduce the performance of all constituent algorithms (possibly in a systematic way) and therefore the overall ensemble. Our proposed method is novel in its iterative application of link prediction to increase the efficacy of community detection algorithms.

In the community detection literature, techniques have been proposed for both evaluating the significance/robustness of communities, as well as, for detecting significant communities. Karrer *et al.* [87] propose a network perturbation algorithm for evaluating the robustness of a given network partition. Methods have also been developed [77, 97] that measure the statistical significance of individual communities. Our goal, however, is not to generate confidence metrics on communities but rather to generate more accurate communities overall. Previous work has also proposed techniques for finding significant communities using sampling based techniques [55, 110, 142]. Rosvall *et al.* and Mirshahvalad *et al.* propose algorithms for detecting significant communities by clustering bootstrap sample networks and identifying communities that occur consistently amongst the sample networks. The method proposed

by Gfeller *et al.* attempts to identify significant communities by finding unstable nodes using a method based on sampling edge weights. Their methods differ from ours in that they create samples from the *existing* network topology. Most similar to our work is the paper by Mirshahvalad *et al.* [111] which attempts to solve the problem of identifying communities in sparse networks by adding edges that complete triangles. Their method is simply to add a fixed percentage of triangle completing edges and cluster the resulting network; in contrast, our approach involves the repeated application of any link prediction algorithm.

The problem of detecting communities at various levels of granularity is a well studied and related problem. Work by [49, 172] has analyzed the “resolution limit” of detecting communities at all granularities. In response to this resolution problem, many methods [7, 38, 99, 141] have been proposed for community detection at different granularities. New objective functions that improve the resolution limit [99] as well as tunable objectives [7, 141] that allow community detection at various resolutions have been proposed. Delvenne *et al.* propose a method for identifying the stability of communities by using the Markov time of a random walk on the network. Granularity is a related problem in that missing edges can lead to communities detected at wrong granularities. However, these methods do not address the problem of detecting communities on incomplete networks.

The design of EDGEBOOST was partly inspired by ensemble methods in the data clustering space. Ensemble data clustering (for a survey see [56]), first proposed by Strehl *et al.* [156], involves the consolidation of multiple partitions of the data into a final, hopefully higher quality partitioning. While many of the ensemble clustering methods share a similar work flow to our method, the fact that these techniques were developed for data clustering and not community detection make them distinct from our work. For instance, Dudoit *et al.* use bootstrap samples of the data to generate an ensemble of partitions, which in the case of network community detection would be difficult since networks have an interdependency between nodes, and nodes cannot be sampled with replacement like data in euclidean spaces. Monti *et al.* [113] propose a consensus clustering technique with the goal of determining the most stable partition over various parameter settings of the input algorithm. Similar to our work, many ensemble clustering algorithms [39, 45, 113, 156] use a consensus matrix as a data structure to aggregate the ensemble of partitions. Unlike previous methods [39, 156], which use agglomerative clustering to compute the final partition we propose an aggregation algorithm that uses connected components, which is not possible on data clustering problems.

## 3.9 Conclusions

Networks inferred or collected from real data are often susceptible to missing edges. We have shown that as the percentage of missing edges in a network grows, the quality of community detection decreases substantially. To counter this, we proposed EDGEBOOST as an algorithm to improve community detection on incomplete networks. EDGEBOOST is capable of improving all the community detection algorithms we tested with its novel application of repetitive link prediction, on real ego-networks from Facebook. EDGEBOOST is an easy-to-implement meta-algorithm that can be used to improve any user-specified community detection algorithm and we anticipate that it will be useful in many applications.

While in this chapter we focused improving community detection algorithms on networks with missing links, in the next chapter we look at how to improve community detection algorithms for detecting topical communities in a social network.



# Chapter 4

## Using Topically Coherent Communities to Rank Social Feeds

When community detection algorithms are applied to binary networks, they group nodes into clusters based purely on the structure of the network. Groups of highly connected nodes are usually placed together while nodes that have a high degree of separation are usual grouped separately. By only leveraging the structure of a network, community detection algorithms are limited to detecting communities that are reflected in that structure. Depending on the type of analysis and therefore the intended insight of applying community detection to a network; the network structure alone may not carry enough information for that analysis to be successful.

In this chapter, we propose a link re-weighting strategy that uses the network structure and text data associated with nodes to generate a modified network where the edges carry a signal about the semantic information shared between nodes. We use this link re-weighting strategy to aid in the detection of topically coherent communities of users in an online social network. We present this technique as a component, in an information retrieval system, that identifies and ranks a user’s social feed by their topic(s) of interest. The system, BUTTERWORTH, proposed in this chapter is based off work presented in this paper [20].

### 4.1 Chapter Overview

Over the past few years, online social networks have shifted focus from socialization platforms to information hubs. Users log onto social networks to obtain the latest updates from their personal friends as well as commentary, links, and news from colleagues and subject-matter experts. The conventional mechanism for displaying this information is the *social feed*, a long (usually) time-ordered list of updates. Active users of social networking sites like Twitter can follow the updates of hundreds or thousands of other users. This can produce feeds that contain hundreds of new items per hour, many of which are irrelevant to

the user. As social networks snowball in popularity and friend networks grow in size, users become increasingly less able to find relevant content as both context and channels collapse.

Users' feeds often contain a heterogeneous mixture of information from many sources and on a variety of topics. This heterogeneity is caused by *context collapse*, the phenomenon in which many people, usually situated in different contexts, are suddenly grouped together. For instance, a user Alice may be primarily interested in data mining, but also interested in cooking and literature. She follows topic-matter experts on all these subjects, but because the primary consumption mechanism is a single social feed, all messages will become intermingled regardless of what they are about.

The converse of the context collapse problem is *channel collapse*. While context collapse describes the consumption of information—all the people a user follows push their updates into one place—channel collapse describes the production of content. A particular user, Jane, may be followed by others for many reasons: her family for pictures of the grandchildren, gamer buddies for the latest news on new releases, and work colleagues for Java coding tips. However, because Jane has only one output mechanism available to her, any content she creates—be it pictures of her children or Java tips—will be seen by all subscribers to her feed, whether they are interested in the topic or not. Alice, who follows Jane only because of their shared interest in games, will be forced to sift through irrelevant Java tips and family pictures. Conversely, Jane's father must get through coding and gaming messages before getting to the pictures.

In response to the problems of context and channel collapse, social networking sites have implemented features that allow users to group their friends into *lists*. Lists let users better organize their feeds by grouping friends who share a similar context; users can then access sub-feeds that only contain content from friends in that grouping. In many situations these lists are intended to aid in consumption, but recent trends in interfaces for these systems also allow lists to be used for targeting produced content. Allowing users to target their posts based on the topic of the content has the potential to alleviate channel collapse. While such features can help users better organize their feeds, they require significant human effort, as well as widespread adoption, in order to be effective.

In this chapter we propose BUTTERWORTH, a system that provides a solution to both context collapse and channel collapse. We observed in pilot experiments that when users create lists, they are also implicitly indicating which content produced by members of that list they are interested in. This is most often the topics discussed in common by all members of that list. For example, when creating a *visualization* list, a user includes other users who post about visualization. Each user in the list may post about whatever other topic she also finds interesting, but a common topic across all of their posts will likely be visualization. By

finding this common core, it becomes easy to train a ranking algorithm to highlight posts related to visualization. Unfortunately, most users do not create lists.

To address this issue, we designed BUTTERWORTH to leverage manually created lists when available, but also to automatically identify people who should be placed together in a list. Unlike topic-modeling approaches, which are highly sensitive to input, computationally complex, require substantial tuning, and are difficult to present to users, BUTTERWORTH’s novel approach leverages the inherent homophily of social networks.

One reason a user may choose to follow or friend another user on a social network is because of their mutual interest in a topic. This type of edge in the social graph has the semantics of “TopicOfinterest” and is common on social networks like Twitter. Since most real-world social graphs are going to obtain edges with different semantics, link prediction can be used to infer missing and re-weight existing links in the network. We use the bag-of-words generated by each node (user) in conjunction with a link predictor to generate a weighted network, that infers the topical social graph. Highly weighted edges in this transformed network, consist of edges corresponding to topical similarity between users. Highly connected groups of nodes then translate to communities of people that discuss similar topics.

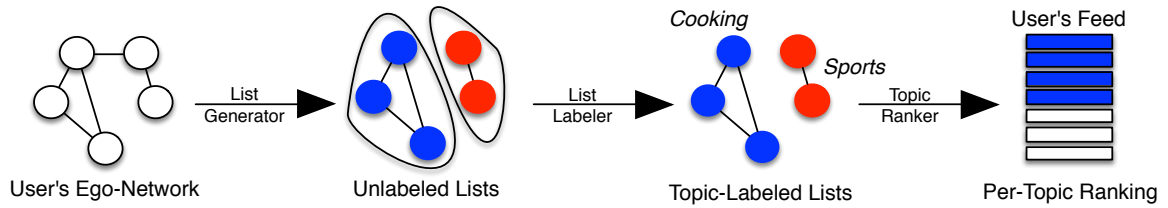
By detecting communities on the imputed version of a user’s social network, BUTTERWORTH can scalably build lists that closely simulate manually created lists, down to picking human readable list labels. For each of these topics BUTTERWORTH generates a “ranker” that re-orders the user’s feed by the relevance of the items to the selected topic. BUTTERWORTH leverages only a user’s social network and content produced by their friends; it therefore requires no direct supervision from the user.

We have tested each component of BUTTERWORTH and have shown that it is able to achieve high levels of precision and recall, on both topic discovery and ranking. We have designed BUTTERWORTH to be broadly applicable to all mainstream social network platforms, but have chosen Twitter as our platform due to its popularity and public API.

## 4.2 User Scenario

Since BUTTERWORTH is focused on an end-user burdened by irrelevant feed content, we present user scenarios that describe today’s current situation as well as the feed interaction enabled by BUTTERWORTH.

To see how users struggle with today’s social feeds, recall our example user Alice. Alice has joined Twitter to find information about data mining, literature, and cooking. She follows



**Figure 4.1** BUTTERWORTH has three components. The **list generator** groups a user’s social contacts into topically coherent lists. The **list labeler** synthesizes labels for those lists. The **topic ranker** learns a ranking model for each topic by using each list to heuristically label training data.

many users who generate content on her main interests, but she also follows other users for various reasons (*e.g.*, they are co-workers, family members, *etc.*). While sometimes Alice enjoys browsing her feed to see the latest content from her friends, at other times she wants to see only content related to her main topics of interest. With the traditional feed mechanism, Alice suffers from *context collapse*; if she wants to see only content related to cooking, she must manually search through her feed to find cooking tweets. Since Alice is interested in cooking, she could devote some time to using Twitter’s list feature to group together all of her friends who write about cooking. When Alice clicks on her cooking *list* she is shown a sub-feed consisting of only content from the users in the cooking list. Sadly, this does not help Alice very much, since she realizes that the friends she follows for cooking information also write about many other topics. Alice’s lists allow her to break up her feed into contextualized groups, but this is not enough, as channel collapse fills even her list-specific feeds full of irrelevant content.

Alice’s experience is very different when she uses BUTTERWORTH. Once Alice logs into Twitter, BUTTERWORTH automatically identifies her topics of interest and presents them to her. If Alice has already created a list, such as her cooking list, then BUTTERWORTH can incorporate it into the topics presented (though manual list creation is not required). Alice is then able to choose a topic that interests her; when she does, messages in her feed are ranked by their relevance to the topic selected. Alice’s feed is no longer affected by context collapse, since BUTTERWORTH automatically generates lists that correspond to her interests. In addition, her feed is no longer affected by channel collapse, since BUTTERWORTH pushes the relevant content to the top of her feed.

### 4.3 System Architecture

BUTTERWORTH comprises three main components, as seen in Figure 5.1. The **list genera-**  
**tor** partitions friends into lists by analyzing their social network. These user lists are then

fed into the **list labeler** that generates a concise label representing the list’s (central) topic. The generated lists are then sent to the **topic ranker**, which trains ranking models for this core topic. The models can then be used to rank the user’s feed by the selected topic.

## List Generator

The first step in the BUTTERWORTH pipeline is to automatically discover groups of users that discuss similar topics within a user’s *ego network*, and to partition the members of this network into lists. The ego-network is a subset of the social network (in this case, a modified and weighted follower/followee graph). It is derived from the friends connected to a core “ego” individual and the links among these friends, excluding any edges to the ego node. The goal of this module is to generate lists (we call these *topical lists*) such that each list corresponds to one of the user’s topics of interest. Even if a user’s main objective on Twitter is to obtain information on topics of interest, they likely follow users for a variety of other reasons. For instance, a user may follow their co-workers or family members. The generated lists for a user may therefore also include these *contextual lists* that are less topically coherent (e.g., lists that contain users who all live in the same town or are all family members). After computing these topical and contextual lists, the **list generator** filters out all contextual lists.

For an example we turn back to Alice. The list generator takes as input Alice’s ego network and partitions all of her friends into lists. Some of these lists will correspond to her interests like cooking and data mining, and others will contain her college friends, co-workers, *etc.*. The **list generator** then removes the contextual lists, so that only the topical lists remain.

While generating high-quality lists is an important problem, it is not BUTTERWORTH’s end goal; we formulate these lists in order to produce high quality topic rankers. While low-quality lists can have a bad effect on downstream ranking performance, we show experimentally that perfect lists are not required for high-quality ranking. The ideal output of the **list generator** is a set of lists that comes close to the quality of lists that users generate themselves, which is how we evaluate this component in the experiments.

## List Labeler

BUTTERWORTH’s second component is the **list labeler**. It generates relevant and human-understandable labels for each of the lists generated in the previous step. These labels are also attached to the generated rankers and are intended for display to the user in the system

interface, enabling her to select which topic ranking she wants applied. For example, the list labeler should name Alice’s cooking list as “cooking” or something similar. While it is, of course, possible to formulate lists without generating accompanying labels, the resulting interface would be substantially harder to use—Alice would need to resort to clicking on all the BUTTERWORTH-given topics and reading sets of tweets to discover the topic behind each one.

We propose two different labeling algorithms, using network as well as textual features (including list labels generated by others, content, and user biographies). An ideal output for the list labeler is a topic string that is semantically close to the label a human would give to the same list. We evaluate the list labeler’s output by comparing it to some simple synthetic baselines, as well as asking human judges to rate its relevance.

### **Topic Ranker**

The final component of BUTTERWORTH is generating a ranking model for each identified topic. The **topic ranker** takes as input the past tweets from the set of users in each list that was labeled in the previous step. It proceeds in two steps. First, it generates a label (“relevant” or “irrelevant”) for a subset of the user’s tweets. Second, it uses this synthetically generated labeled data to train a naïve Bayes model. Once a ranking model is trained for each topic, the model can be applied to rank a user’s feed by the corresponding topic. For example, after Alice selects the cooking topic, her *entire* feed would be re-ordered such that the cooking tweets are pushed to the top of the list. Note that we purposely rank the entire feed to deal with both false-positive and false-negative assignment of users in the list generation.

In order to train each ranker, we propose various heuristics to automatically generate “relevant” and “irrelevant” labels for training examples—a form of distant or self-supervised learning. The main intuition behind these heuristics is that since the users contained in each list discuss similar topic(s), the most frequently discussed topics are most likely to be relevant. An ideal output of the ranker would order a user’s feed such that all of the content relevant to the list appears at the top of their feed. We use standard information retrieval techniques to evaluate the ranking quality.

**Table 4.1** The top three ranked tweets for automatically generated lists labeled *environment*, *fashion*, and *sports* (list labels are also automatically generated).

<b>Environment</b>	Join us in helping get the 2012 Olympics off plastic bags!...
	Heading to Patagonia Santa Monica store to bring... on plastic pollution..
	Starbucks Trash: Behind the Scenes :: My Plastic-free Life — Less Plastic...
<b>Fashion</b>	Anchors away in this Vintage Yellow Sailor dress! Newly listed...
	STUNNING Vintage Paisley Teal Spring Dress!...
	Vintage Silk Heart Blouse with a Tie Neck listed...
<b>Sports</b>	I posted 12 photos on Facebook... ”Warrior Around the NHL...
	LeBron James and Michael Jordan (92): are thus the only...
	Denver Nuggets Sign Quincy Miller...

## 4.4 Algorithms

Here we describe how we implement each of the components above: the list generator, the labeler, and topic ranker.

### 4.4.1 List Generation

The goal of list generation is to take a user’s ego-network and partition the friends into topically coherent lists. Fortunately, partitioning the nodes of a graph into subsets is one of the topics addressed by graph clustering research. In particular, the algorithm of Pons and Latapy [132] takes as input an undirected weighted graph, then produces a disjoint set of node sets. An edge weight in the input network generically describes the strength of the link between two nodes. In principle, we can take the (unweighted, directed) ego-network to create a (weighted, undirected) input to the clustering algorithm, and then call each of the algorithm’s clusters a *list*.

Of course, the quality of clustering depends largely on the network we formulate as input. By choosing edge weights differently, we can obtain different kinds of clusterings. We have found that using weights that take into account shared topics leads to groupings that emphasize these topics while de-emphasizing *contextual* relationships (family, childhood friends, *etc.*) that are, for our purposes, “spurious.”

Each ego-network consists of a set of nodes,  $N$ , and a set of directed edges,  $E$ , where each edge indicates the presence of a follower-followee relationship. For simplicity, we first convert each directed edge in  $E$  to an undirected edge. Then, we use a combination of network features and textual features to produce  $E'$ , the set of weighted edges.

For each pair of nodes  $(u, v)$ , where  $u \in N$  and  $v \in N$ , we produce a weighted, undirected

edge  $e_{u,v}$  using a modification of the user similarity measure presented in [3]:

$$sim(u, v) = \sum_{x \in sharedneighbors} \frac{1}{\log[degree(x)]} + \sum_{t \in sharedterm} tfidf(u_t) * tfidf(v_t) + E(u, v), \quad (4.1)$$

where  $E(u, v) = 1$  when  $e_{u,v} \in E$ , and 0 otherwise. If  $sim(u, v) > 0$ , then  $e_{u,v}$  is added to  $E'$  with weight  $sim(u, v)$ . To produce the “shared term” weights, we first produce a TF-IDF-weighted term vector for each node  $n \in N$ , limiting the per-user vocabulary to the 10 top-scoring words. In examining the networks produced by our edge weighting algorithms, we realized that many networks appeared extremely dense. To correct for this, we discard all edges with weight less than a threshold parameter  $\alpha$ .

After we have produced a weighted, undirected graph, we perform graph clustering to obtain a set of lists. Currently, we apply a common approach based on random walks [132], as implemented in the iGraph R package. The clustering algorithm uses properties of random graph walks to produce a disjoint set of communities, and we consider each community produced as a *list*.

After identifying the lists, we must decide whether each list and its members are topically cohesive. To do this, we compute the *entropy* [149] of the list, in which the list’s probability distribution is defined as a bag-of-words model over all tweets in the list. Our intuition behind using entropy is that if a list’s tweets share common topics, then the distribution over words will be skewed to a small subset of the whole vocabulary and thus have low entropy. We classify all lists with entropy lower than an empirically learned  $\epsilon$  value as *topical*.

## 4.4.2 Topic Labeling

When generating a label for a topic list, we use one of two algorithms: the BESTOVERLAP method, or the USERINFOBIGRAM method.

In the BESTOVERLAP method, we exploit Twitter’s large user base to implicitly “crowd-source” topic list labeling. For each topical list, we create a set of candidate names by looking at all previously-created Twitter lists that contain members of our newly generated list. To label our new list, we simply pick the name of the previously created list that maximally overlaps with the individuals in our list. For example, if our list contained users A, B, and C and we find two other previously created lists—“foodies,” which contains A and B and “cooking,” which contains A, B, C, and D—we label our new list “cooking.” If



multiple Twitter lists overlap equally, up to three names are concatenated to form the list label.

For the USERINFOBIGRAM method, we examine the optional text that Twitter users can enter to describe themselves (e.g. “I love NYC, tech & funk” or “CS PhD student at the University of Rochester... My research involves real-time crowdsourcing, human computation, and AI”). For each newly created list we generate a “corpus” of all user information fields of list members. The list is then labeled with the most frequent bigram in this synthetic corpus.

We found through experimentation that the BESTOVERLAP method works best when our generated lists have more members (increasing the chance of overlap and the size of overlap with previously created lists). After considering various limits, we use a minimum threshold of 10 list members to decide when to switch from BESTOVERLAP to USERINFOBIGRAM.

### 4.4.3 Topic Ranking

The last step in the BUTTERWORTH pipeline is to build ranking models for each topic. The input for each ranking model is the past tweets of each member of the corresponding list, comprising a set of unlabeled tweets  $T$ . Of course, these tweets are not labeled as “relevant” or “irrelevant.” However, we can exploit the fact that all of the tweets come from users grouped together in a single list. We propose various heuristics for automatically labeling a subset of the examples in  $T$ . Our learning scenario falls into the category of distant supervision, in which a heuristic labeling function  $H$  is applied over all of the examples in  $T$  to produce a labeled set  $T_{\mathcal{L}} \subseteq T$ . We only use  $H$  to try to infer positive training examples—obtaining high-quality negative examples is much easier. For each list, we randomly sample  $|T_{\mathcal{L}}|$  from the corpus of tweets not produced by users in the list. After obtaining a labeled training set, we then extract the bag-of-words features from each training example and train a naïve Bayes model for each list. The final feed ranking is produced by sorting the tweets by their relevance probability, as scored by the trained model for each list. Below we propose three different variants of the labeling function  $H$ .

#### Naïve Method

In the naïve method we use all of the tweets produced by the list members as our positive training set, and sample from out-of-list tweets for our negative training set. More formally, we let  $H$  be a constant function, labeling all of the examples in  $T$  as positive, to create  $T_{\mathcal{L}} = T$ . While this method is very simple, it achieves surprisingly good results, as our

experiments show.

### Top-K Hashtags and Unigrams

By labeling all of the examples in  $T$  as positive, the naïve method creates a noisy training dataset. Instead, we would like a heuristic that tries to label as positive only the examples that are surely positive. Here we propose heuristics that try to leverage the observation that the more prevalent content in  $T$  likely corresponds to the more relevant content to the end user. Instead of labeling all of the examples in  $T$  as positive, we label a smaller subset that contains examples that are the most likely to belong to the positive class. We use a modified TF-IDF score to find the unigrams and hashtags<sup>1</sup> that are most likely to be contained in the positive class. We score each hashtag or unigram occurring in  $T$  using the following TF-IDF scoring function for a given list  $l$ :

$$TF * IDF(w) = F_l(x) \times \log \left( \frac{N}{IDF(x)} \right), \quad (4.2)$$

where  $F_l(x)$  is the number of occurrences of the hashtag or unigram  $w$  in the list  $l$ ,  $N$  is the number of tweets in the corpus and  $IDF(w)$  is the number of tweets that contain  $w$ . We define two heuristic functions, one for unigrams and one for hashtags.  $H$  labels all of the examples in  $T$  that contain at least one of the top- $k$  unigrams or hashtags, where  $k$  is a chosen parameter.

## 4.5 Evaluation

Below we describe a set of four experiments to test various aspects of BUTTERWORTH. All of our experiments use data collected from Twitter which we describe below.

### 4.5.1 Dataset Description

In order to evaluate BUTTERWORTH, we need a ground-truth dataset containing real users who have manually created their own lists. If our algorithmic list-generation component can accurately recreate these lists, and if we can use the resulting lists to formulate accurate

---

<sup>1</sup>Hashtags are user-specified single-word descriptions that signal the topic of a tweet and always begin with a “#”, i.e., #Food or #Climate

topic labels and rankings, then we can be confident that BUTTERWORTH will be effective in a general deployment.

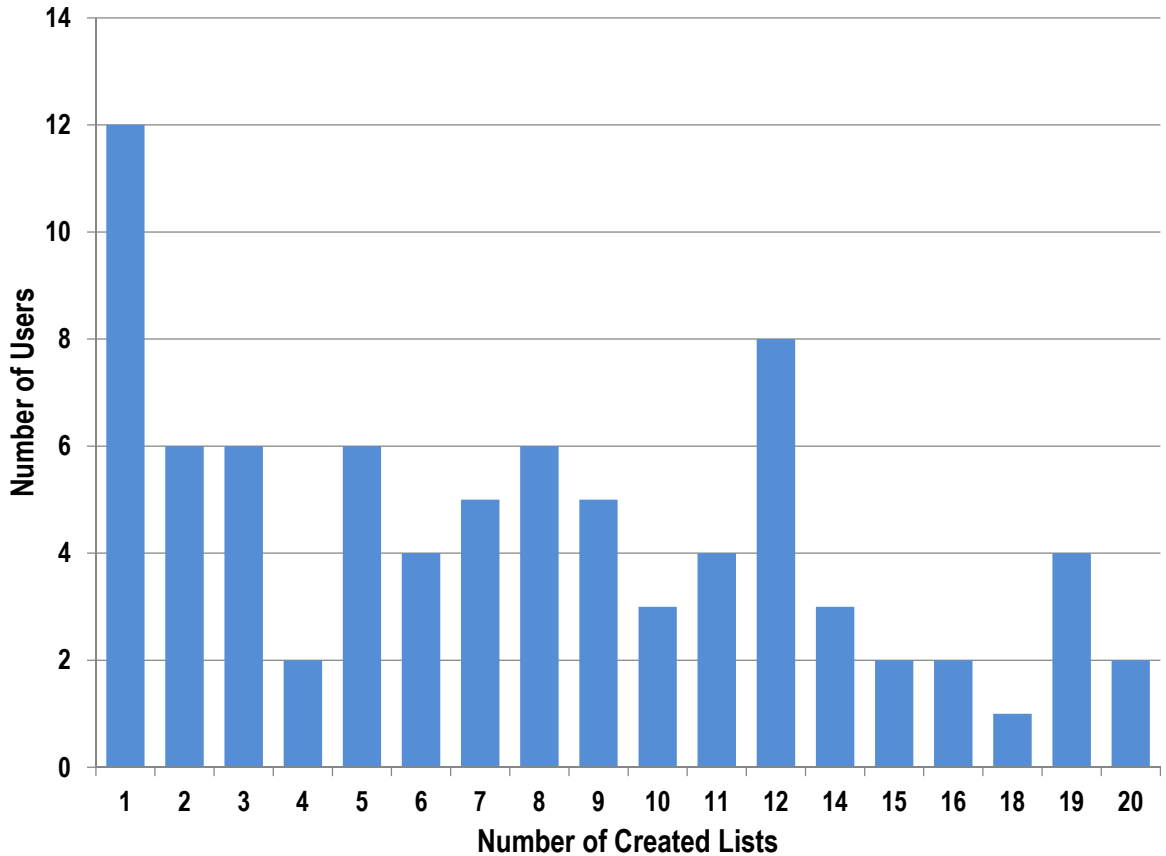
Unfortunately, it is not straightforward to find a random sample of users that has created high quality lists. We obtained such a sample from Twitter as follows:

1. We chose by hand a set of 10 high-quality lists drawn from [www.listorius.com](http://www.listorius.com). They cover a range of topics, including computer science, cooking, and others. Each list has between 300 and 500 users. Together, these users formed a large seed set of users who are likely to be members of many lists.
2. We then obtained all lists that contained *any* of the seed users. This formed a large set of lists likely to be of high quality.
3. We found the creator of each list, yielding a set of nearly 400,000 users. We have now found a subset of Twitter users who have created high-quality lists — this is exactly the subset from which we want to draw our test sample.
4. Last, we randomly sampled 100 users from the follower set, and removed users who were using private accounts or who were non-English speakers. The resulting 80 users formed the *test-user* set. These users had created anywhere from 1 to 20 lists (mean of 7.79 and median of 7 lists). Figure 4.2 shows the final distribution of lists created by these test users.

Each of the obtained lists, which we call *organic lists*, has a user-given name (i.e., label) and a set of list members. Finally, we created an ego-network for each test user that comprised all of the test user’s followers/followees, all members of a test user’s organic lists, and any friend relationships among these users. The average ego-network size is 1383.72 and the median is 805. For our experiments we downloaded up to 1,000 tweets from each user in the resulting network.

## 4.5.2 List Generation

To evaluate our list-generation algorithms, we experimented with several alternative edge-weighting schemes. For each ego user in our dataset, we produced one weighted, undirected ego-network graph for each weighting scheme. Next, we ran the clustering algorithm on each graph, producing a set of disjoint lists. Because all users in our dataset had at least one organic list, we evaluated the sets of lists produced by the community detection algorithm against the user’s manually created organic lists. We evaluated the following weighting

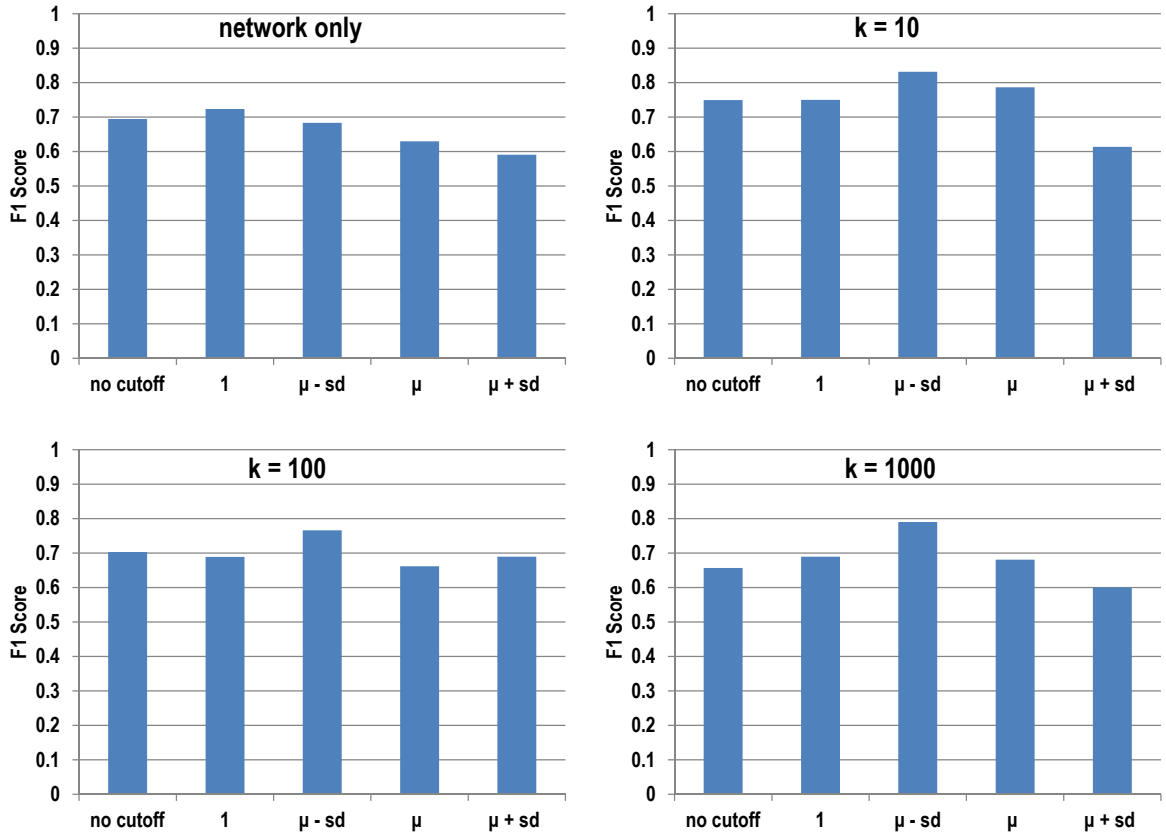


**Figure 4.2** Distribution of the number of lists created by users in the test set (20 is the maximum allowed by Twitter)

schemes: network only, in which the edge-weighting function  $sim(u, v)$  considers only the neighbors as features, and three additional schemes in which the edge-weighting function  $sim(u, v)$  considers neighbors and a TFIDF-weighted term vector as features, limiting the per-user vocabulary to  $k$  words, where  $k \in \{10, 100, 1000\}$ .

We also tested the edge-weight parameter  $\alpha$  with values that were functions of the average edge weight and standard deviation for each network:  $\alpha \in \{1, \text{average edge weight} - \text{standard deviation}, \text{average edge weight}, \text{average edge weight} + \text{standard deviation}\}$ . Figure 4.3 shows the results of this experiment. We see that taking  $k = 0$  and  $\alpha = (\text{average edge weight} - \text{standard deviation})$  results in the highest value of F-measure, 0.83. Thus, we chose this algorithm as our edge-weighting scheme.

Our goal in this list-generation experiment was to produce lists that mimicked each user’s manually created organic lists. However, community finding was performed on each user’s entire ego-network, which included friends that were not placed on any organic lists. Therefore, in evaluating the machine produced lists against the manually created organic lists, we created a confusion matrix for each user as follows: for each pair of friends that



**Figure 4.3** F1 scores for the list-generation module, varying  $k$  and  $\alpha$ .

was placed in any organic list  $(f1, f2)$ , the pair is a true positive if the friends were both placed in the same organic list, and were both placed in the same machine produced list; the pair is a false positive if they were both placed in the same machine-produced list, but were not placed in the same organic list, etc. We cannot include users who were not placed on any organic lists in our evaluation metrics because we do not know why they were not included in the list — they may belong on the list, but the user lazily left them off, or they truly do not belong. However, we will show in the next few sections that even with imperfect list generation, our results are still quite good.

To evaluate our topicality threshold, we performed a 5-fold cross-validation on a set of 100 manually classified lists. Two human evaluators constructed the training and testing set by manually classifying 100 randomly chosen organic lists as either topical or non-topical. We then calculated the entropy of the lists, and learned an  $\epsilon$  cutoff value by choosing the cutoff to maximize the F-measure. Our final  $\epsilon$  value was then chosen as the average of the 5 runs, yielding  $\epsilon = 1.374$  and F-measure = 0.92.

### 4.5.3 Topic Labeling

We evaluated eight potential labeling algorithms before choosing the BESTOVERLAP and USERINFOBIGRAM methods. We performed two experiments to evaluate these algorithms. First, we chose a random sample of 100 organic lists from our dataset. For each list, we produced 8 labels—one from each labeling algorithm. To evaluate how similar each label was to the organic list’s original label, we computed the pointwise mutual information (PMI) score of each label, relative to the original label. To compare all labeling techniques, we then calculated the root mean squared error (RMSE) of each algorithm’s PMI scores.

The algorithms we evaluated were: (1) the BESTOVERLAP method; (2) the LISTUNIGRAM method, in which topic lists are labeled with the most common unigram from the names of all previously created Twitter lists; (3) the TWEETUNIGRAM method, in which topic lists are labeled with the most common unigram occurring in the past 1,000 tweets from all list members; (4) the TWEETBIGRAM method, in which topic lists are labeled with the most common bigram occurring in the past 1,000 tweets from all list members; (5) the TWEETWIKI method, in which we compute the top-5 unigrams occurring in the past 1,000 tweets from all list members, search Wikipedia with these unigrams, and label the topic list with the most common unigram occurring in the parent categories for the top 10 search results; (6) the USERINFOUNIGRAM method, in which topic lists are labeled with the most common unigram occurring in the user info fields from all list members; (7) the USERINFOBIGRAM method; and (8) the USERINFOWIKI method, which is identical to the tweet wiki method, but chooses the top-5 unigrams occurring in the user info fields from all list members in the first step. Results from this experiment are presented in Table 4.2.

Because the BESTOVERLAP, TWEETBIGRAM, and USERINFOBIGRAM methods all had similarly low RMSE scores, we conducted a second evaluation, just using these three algorithms. Our goal in this second evaluation was to evaluate the meaningfulness and relevance of a label. Two human evaluators manually classified each label for the 100 random organic lists as very relevant (a score of 2), relevant (a score of 1) or irrelevant (a score of 0) when compared with the organic list’s original label. These results are also presented in Table 4.2. The BESTOVERLAP method outperforms the TWEETBIGRAM and USERINFOBIGRAM methods significantly ( $p < .01$ ). Although the USERINFOBIGRAM method appears to outperform the TWEETBIGRAM method, the difference is not significant.

**Table 4.2** RMSE and average relevance values for labeling algorithms.

Algorithm	RMSE	Avg relevance
BESTOVERLAP	6.79	1.72
LISTUNIGRAM	7.38	–
TWEETUNIGRAM	7.54	–
TWEETBIGRAM	6.64	0.91
TWEETWIKI	7.87	–
USERINFOUNIGRAM	7.35	–
USERINFOBIGRAM	6.56	1.14
USERINFOWIKI	7.52	–

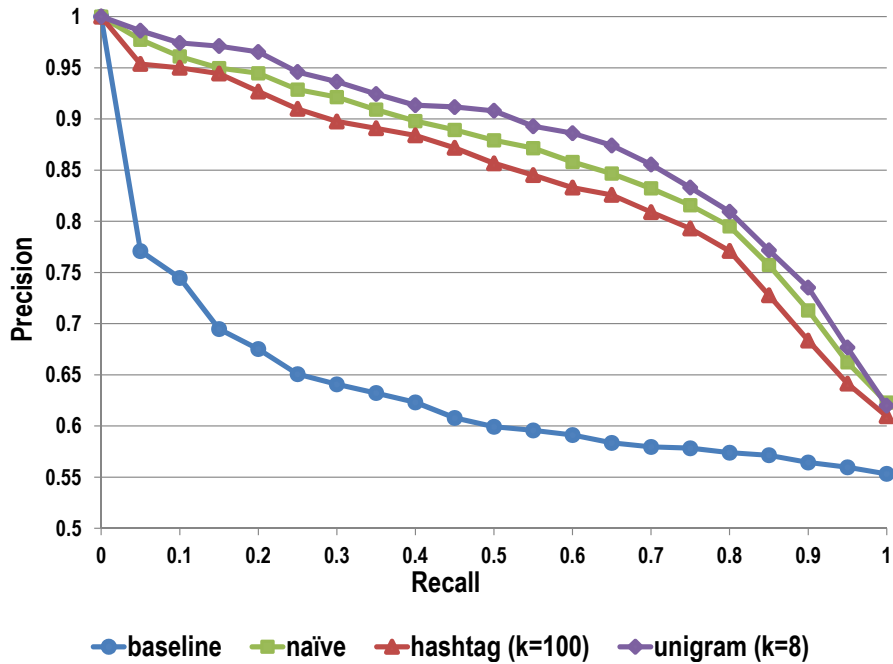
#### 4.5.4 Topic Ranking

In order to evaluate the ranking, we needed to obtain ground truth about the relevance of tweets to the topic name of a list. We randomly sampled 100 organic lists from the test user set for which we wanted to obtain labels. We used Amazon’s Mechanical Turk to obtain binary labels of relevant or irrelevant to a prescribed list name. For each of the 100 organic lists we sampled 100 tweets, and asked three Mechanical Turk workers (“Turkers”) to grade whether a tweet was relevant to the list name or not. Since the list names were generated for personal use by Twitter users, some were less informative (*e.g.*, “mac,” “vegas tweeple”), and as a result the Turkers were not able to give high-quality ratings. Since we only gave the Turkers two options, for the incoherent list names the vast majority of the answers selected were “irrelevant,” as the Turkers did not understand the list name. We used for ranking only those lists for which Turkers unanimously agreed that at least 15% of tweets were relevant. After filtering out lists with incoherent names, we obtained a set of 55 lists that we used for testing. From each list, we used only the examples to which the Turkers responded unanimously, yielding a total of 3,215 labeled examples. The mean percentage of relevant tweets over the 55 lists was 52%. We evaluated the performance of the ranking models using standard information retrieval metrics.

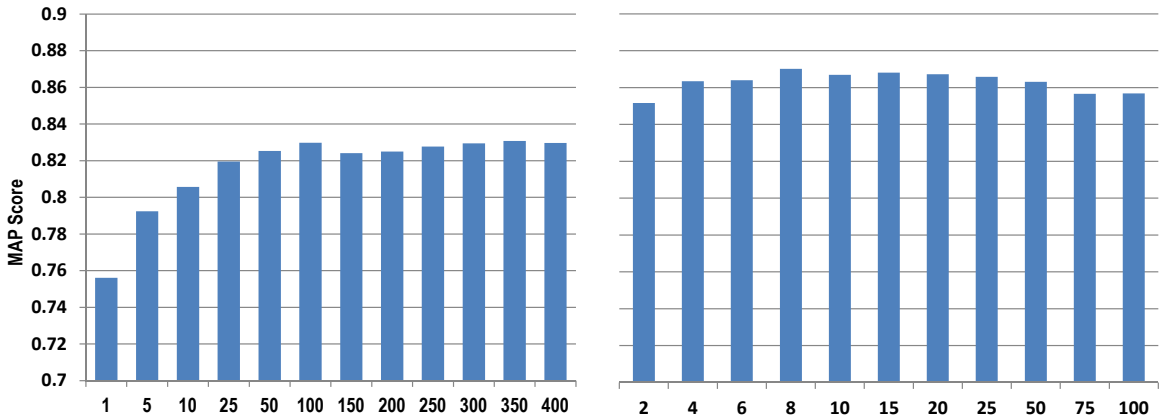
Our first set of experiments test the effect of the  $k$  parameter for the hashtag and unigram methods. We then compare the hashtag, unigram, and naïve methods against a baseline. We show the robustness of each heuristic under various amounts of noise in order to determine the sensitivity of the ranking module to the output of the list-generation module.

#### Tuning the $k$ parameter

Figures 4.5 show the Mean Average Precision (MAP) scores for various values of the  $k$  parameter. As the plots show, the MAP scores do not vary much between the different  $k$



**Figure 4.4** Precision and recall of the hashtag, unigram, naïve ranking methods against a baseline.



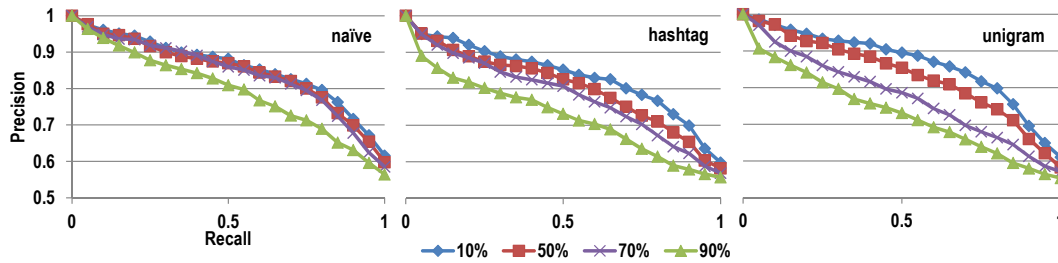
**Figure 4.5** Varying the  $k$  parameter for the hashtag (left) and unigram (right) methods.

values. As long as we choose a  $k$  value that is not very small, the performance of the ranker is consistently good. When  $k$  is too small, the naïve Bayes model is not supplied enough training data and performance suffers. We choose  $k = 8$  (unigram) and  $k = 100$  (hashtag) for the following experiments, as those values performed slightly better than others.

### Comparing Proposed Methods to Baseline

By default Twitter ranks a user’s feed in reverse chronological order, so we compared our method against this baseline. Ranking the tweets by time is essentially the same as randomly ordering with respect to their topical relevance. Figure 4.4 shows the precision-recall curves





**Figure 4.6** A comparison of the effects of varying the percentage of noise on precision and recall for the naïve, hashtag, and unigram methods.

for the naïve, hashtag, and unigram methods as compared to the baseline. As shown in the figure, our methods significantly outperform the baseline, which quickly converges to a precision of around 50%, which is the average percentage of relevant topics over all lists. All three of our proposed methods have a very similar performance, in which the unigram method slightly outperforms the other two methods. The naïve Bayes classifier is robust to noise, so it is not surprising that the naïve method performs well.

### Robustness to Noise

The previous experiments tested the proposed methods on user-generated lists, which we assume are high quality. Since part of BUTTERWORTH’s purpose is to automatically generate lists, we also wanted to see how sensitive each of the methods were to varying levels of noise. We simulated mistakes in list generation by replacing a varying percentage of tweets in each test list with random tweets from outside the list. For example, a noise level of 30% for a given list would correspond to 30% of the tweets in the training set being replaced with random tweets sampled from users outside the list. Figure 4.6 shows the precision-recall plots for all three methods with various levels of noise. We can see that the naïve method is the most robust to increasing levels of noise—it is only truly affected at levels of noise above 70%. The other methods are more affected by noise, likely because they quickly begin to only include hashtags/unigrams that correspond to irrelevant topics.

### 4.5.5 End-to-end Experiment

The previous experiments evaluated each component of BUTTERWORTH in isolation, but we have yet to test the effectiveness of BUTTERWORTH as a whole. In order to test BUTTERWORTH as a complete system, we need to experiment on an end-to-end workflow that tests the quality of the rankers given generated lists and their labels from the list generator

**Table 4.3** Precision at k.

<b>precision@k</b>	<b>unigram (<math>k = 8</math>)</b>	<b>baseline</b>
1	0.77	0.52
5	0.76	0.46
7	0.76	0.44
10	0.78	0.45

and list labeler, respectively. We propose an experiment with the following workflow:

1. Randomly select 100 generated topical lists (from our set of test users) that contain at least 5 members
2. Label each of the 100 generated topical lists using the BESTOVERLAP list-labeling algorithm
3. Use the unigram (at  $k=8$ ) topic ranker to rank a random sampling of 100 tweets from each of the generated lists and output the top 10 ranked results. For a baseline, we also output 10 randomly ordered tweets.

To evaluate the results of this experiment, we employed three human evaluators to score the top 10 results from both the baseline and the topic ranker. We present the results from this experiment in Table 4.3. As we can see, the topic ranker greatly outperforms the baseline, with 78% precision@k, compared with 45% precision@k achieved by the baseline. Table 4.1 shows a few examples of BUTTERWORTH’s output.

We can compare the average precision achieved by the baseline, 45%, with the average rate of relevant tweets found in organic lists—52%. These very similar rates indicate that the topical lists generated by the list-generation module mimic the quality and topical coherence of human-constructed lists. Because there are many components in BUTTERWORTH’s pipeline, there is a high potential for error propagation and compounding. However, we can see that BUTTERWORTH performs quite well, maintaining high-quality ranking despite potential noise in list generation.

## 4.6 Discussion

While BUTTERWORTH performs well for the general scenarios we experimented with, there are some user cases which require adaptation of the strategies we employ. Currently, BUTTERWORTH places each friend in exactly one cluster (i.e., a hard-clustering). If a friend

tweets about multiple topics of interest to the user, then the clustering procedure will identify only one of these interests. We propose two possible solutions to this problem. First, we could simply modify our clustering algorithm to allow friends to be part of many clusters (e.g., through soft- or hierarchical-clustering). Figure 4.6 demonstrates that BUTTERWORTH is capable of ranking well under noisy list generation. Thus, we believe the potentially noisier clusters produced by a fuzzy clustering algorithm would not drastically affect ranking quality.

Instead of modifying our clustering algorithm, we could alternatively use the ranker on *all* tweets rather than those produced by a specific list. A solution at the interface level would enable users to choose the set of friends to which the ranker would be applied. For instance, if BUTTERWORTH detected that the user is interested in cooking, the modified interface would enable the user to apply the cooking-ranker to just a specific friend, just the generated list of cooking friends, or to their entire set of friends. To test the feasibility of this modified interface we re-ran the end-to-end experiment (Table 4.3) using a user’s entire feed as input to each ranker. We found that BUTTERWORTH performs slightly better, giving 82% precision@10. This slight improvement may be due to the fact that BUTTERWORTH can find the “best” tweets on a given topic regardless of who produced them. In some cases, these may be as good or better than those produced by list members.

A second potential issue with BUTTERWORTH is linked to our user interaction model. Our system explicitly ranks a user’s feed by topics of their interest and therefore loses temporal ordering. We believe this issue can also be fixed with a modified interface design that surfaces BUTTERWORTH’s ranking in a different way. Instead of completely reordering a users feed on a page, we propose an interface that uses the generated rankers to color the tweets in a user’s temporally-oriented feed. Thus, the salience of a tweet in the feed would vary based on the score that the ranker produces for each tweet. This interface would allow a user to view both topically and temporally relevant content.

We have a few ideas about how to improve BUTTERWORTH’s future performance. First, users who have overlapping interests may also build overlapping lists, write similar text, retweet related content, and link to similar URLs; the list-generation step should thus incorporate information sources beyond network structure. Second, each social networking site has some particular features — say, privacy settings unavailable in other systems—that might shed additional light on how to construct better lists, which we should exploit. In addition, now that we have constructed a robust back-end architecture, we have begun to consider the design of the UI for BUTTERWORTH. The mechanism by which the user accesses lists (manually created and automated, topical, and contextual), provides feedback, controls lists and ranking behavior can, and should, be taken into account to create a usable

user experience. The interactions of the user with the system can be utilized to further inform the intelligent components of BUTTERWORTH.

## 4.7 Previous Work in Social Data Management

There has been a substantial amount of recent research on managing social media data. However, this work has either emphasized text-only approaches (e.g., better ranking and topic modeling) or network-only strategies (e.g., community detection). BUTTERWORTH seeks to leverage both approaches to create a more robust technique that can be readily integrated into user interfaces in a manner consistent with users' expectations.

The bulk of work in social media ranking has focused on classifying posts in a user's feed [31, 35, 74, 123, 125, 155, 160]. Both Hong *et al.* [74] and Das Sarma *et al.* [35] proposed ranking mechanisms based on collaborative filtering techniques. Hong *et al.* used a click-through rate based model while Das Sarma *et al.* compared various mechanisms based on different types of user supervision, including having the user provide pair-wise comparisons of feed items. Paek *et al.* and Dahimene *et al.* [31, 123] built ranking and filtering methods that are personalized for each user and which explicitly model the user that is producing the tweet. Uysal *et al.* [160] describe methods to predict the likelihood of retweets (a user propagating a tweet to friends), and used the likelihood of retweet as the ranking score. Pal *et al.*'s algorithm for finding relevant Twitter users [125] is an unusual point in the space as it attempts to identify topic experts using a classifier that leverages features such as follower counts and retweeted content. These approaches address both content and context collapse by learning user preferences and hiding irrelevant content. However, these models do not necessarily aid a user in organizing her information—an explicit goal of BUTTERWORTH. Additionally, while effective with enough training data, such systems often require a great deal of training data to be useful. One of the goals for BUTTERWORTH was not to require direct input from the user, but instead leverage the inherent homophily in a user's social network to provide high quality groups, and subsequently, rankings.

A different approach to data management focuses on organizing and classifying rather than ranking. If content is clustered for a user based on topic, she may identify and select the information that is relevant to her. Topic modeling based methods (both on users and content) feature prominently in this space [76, 103, 139, 173]. Inspired by these ideas, an earlier version of BUTTERWORTH attempted to build Latent Dirichlet Allocation (LDA) topic models. While useful for describing lists, we encountered many common problems: they were computationally costly, highly sensitive to noise, and required too much tuning

to work effectively across a broad spectrum of users. Further, providing interpretable descriptions for topic models is a notoriously difficult problem, and even “optimal” models may not be consistent with reader preferences [16]. Prior work has demonstrated that unlike topic models, the names of lists in which a particular Twitter user appears are much better representations of the expertise of that user [161]. Because of these common problems, we chose to move away from topic modeling.

In addition to the topic-modeling work for analyzing the individuals a user may follow, there has been a small amount of work on explicitly managing lists on social networks. Kim *et al.* [89] conducted an analysis of lists on Twitter and concluded that topic-centric lists contain users who tend to publish content on similar topics, reinforcing the motivation for our distant supervision heuristics. Another analysis by Fang *et al.* [44] studied sharing “circles” on Google+, and discovered that circle sharing enabled users with few contacts to grow their networks more quickly. In this arena, our work is most similar to that of Guc [61], which describes a filtering mechanism for “list-feeds”; however, that work is framed as a standard supervised learning problem that also requires explicit training data from the user.

Finally, a number of research systems have focused on creating interfaces that enable users to more efficiently browse their feed [9, 75, 159]. For example, *Eddi* [9] displays a browsable tag cloud of all the topics in a user’s feed, allowing the user to more easily find tweets related to her interests. *FeedWinnower* [75] is an interface that allows users to rank tweets by different tunable parameters such as time and topic. Tseng *et al.* proposed a (graph) visualization system called *SocFeedViewer* [159], that allows users to analyze a topological view of their social graph. SocFeedViewer implements community detection algorithms to group similar users; as with BUTTERWORTH’s lists, these community feeds may contain irrelevant content, a problem that SocFeedViewer does not address. Many of these systems apply one or more of the ranking and classification techniques described above. This, unfortunately yields lower precision and recall. The higher quality grouping and ranking strategy of BUTTERWORTH can be readily integrated into various commercial and research interfaces and directly improve their function.

## 4.8 Conclusion

In this chapter we have described BUTTERWORTH, a system for automatically performing topic-sensitive grouping and ranking of the messages in a user’s social feed. Unlike existing approaches for message ranking, BUTTERWORTH requires no explicit guidance from the user and works across a spectrum of users and scales. By breaking apart the simple “fol-

low” relationship into multiple topical lists, BUTTERWORTH addresses the context collapse problem. By leveraging each of these lists to generate rankers that can be applied to the user’s feed, the system alleviates channel collapse. This is achieved through use of a novel architecture that leverages a user’s ego network structure.

As part of the BUTTERWORTH pipeline we propose a method for detecting topically cohesive communities by re-weighting network structure. The re-weighting strategy utilizes the semantic information contained in the text content produced by the nodes. We have shown the efficacy of this technique for detecting topical communities in ego-networks but we believe it will be useful in other applications such as, clustering research papers in citation networks. BUTTERWORTH represents a good example of how the re-orientation of network structure can enable better downstream analysis in a practical and novel data mining system.

# Chapter 5

## Inferring A Knowledge Graph Heterogenous Data Sources

In chapters 3 and 4 we proposed methods to impute networks with the goal of improving community detection algorithms. Chapter 3 proposed a general framework for imputing missing information in networks with an application to improve community detection performance on networks with missing edges. In chapter 4 we proposed a method to impute missing semantic information in a network in order to extract topical clusters. The focus of this chapter is on the problem of inferring missing links in a network using node attributes and relationships defined in a heterogenous set of data sources.

It is often the case that data describing pairwise relationships between nodes exists in many different data sources, but inferring edges based on only one data source is not sufficient. In this chapter, we focus on the problem of inferring subsumption relationships between keyword tags listed by users in online social platforms. The existing structure of the subsumption network is a very sparse set of edges that are labeled by human experts. The structure of the existing network is not dense enough to predict missing edges, and therefore we compute features based on node attributes from auxiliary data sources. We propose a supervised learning method that builds a knowledge graph. Our method trains a model on the existing edges and predicts novel subsumption relationships between nodes. The work presented in this chapter is based off of work presented in [21].

### 5.1 Social Tagging on Expertise Driven Platforms

Professional social networks such as LinkedIn and Academia.edu as well as question-answer sites such as StackExchange and Quora are examples of expertise driven online communities, which comprise an important niche in the social web landscape. In order for these expertise driven applications to offer compelling features, they rely on data that describe user's interests and skill set. Most often this is achieved by enabling users to list keyword

tags describing their expertise. For example, LinkedIn has a form of standardized keywords, called *skills*, Quora uses *topics* and Academia.edu has *research interests*.

Products like LinkedIn *Recruiter Search* leverage user-listed skills to help fulfill queries related to finding users based on their skill set. Quora can leverage topics to help ascertain the expertise of their user base and match questions to the appropriate respondent. The recommendation of news articles, status updates and advertisements can also be enhanced by matching the keywords of the suggested item to those keywords listed by the user. While keyword tags provide descriptive and succinct descriptions of users, they are not comprehensive and do not have any inherent organization or structure. The unstructured nature of tag creation, enables users to generate large and descriptive tag vocabularies but, at the expense of a disorganization.

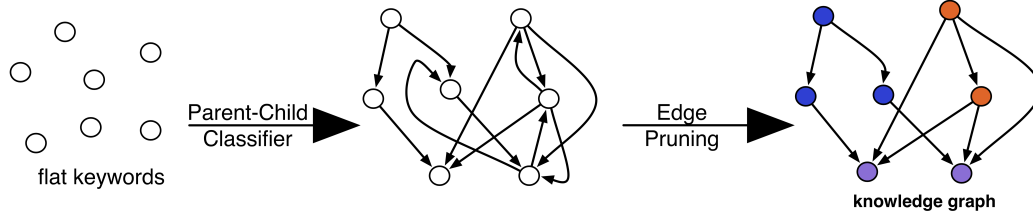
Search and recommendation applications can be improved by having knowledge of relationships between tags. For example, a recruiter using LinkedIn might like to find scripting language experts. Unfortunately, due to the proliferation of these languages and variants, profiles are tagged with a wide array of tags describing specific languages. In addition, users tend to use language that describes their specific expertise like “Python”, rather than list more general terms like “scripting” as one of their skills. If a recruiter cares more about the high level skill, it may not be possible for them to create a query without enumerating all specific scripting languages in the query. Having a structured representation of tags would also help with recommending keyword tags to new users and can also be used to aid in clustering users that share similar skill sets [144].

Due to the diversity of keyword tags that occur in different expert communities, automatically inferring *typeOf* relationships between keywords poses a challenge. For example, skills on LinkedIn and topics on Quora can range from the very broad, such as “Management” and “Programming”, to the very specific such as “SQL” and “CPLEX” (mathematical programming software). Since many keyword tags are domain specific acronyms and do not have any corollary to terms in common language vocabularies, one cannot simply use an existing ontology such as WordNet. Previous methods proposed for organizing keyword tags on sites like Flickr rely on brittle rule-based algorithms [73, 90, 102, 152].

## 5.2 Overview of DOBBY

In this chapter, we propose DOBBY, a system for building a knowledge graph of user-defined keyword tags. The knowledge graph consists of a network where the nodes are keywords and directed edges represent hypernym *typeOf* relationships. Having a knowledge graph





**Figure 5.1** DOBBY consists of two components. The first component classifies parent-child relationships between keyword tags using features described in section 5.3. The second component prunes edges that participate in reciprocal predictions.

allows for a robust and descriptive representation of tags, which can be leveraged by many applications. For example, a knowledge graph can be used to automatically expand search queries on LinkedIn recruiter search or to infer missing tags used to annotate questions on sites like Quora and StackExchange. We can also use the knowledge graph to measure the similarity of two tags using various path based metrics, such as semantic relatedness [66].

DOBBY is a technique for constructing a knowledge graph of user tags for expertise-driven web applications. While many such applications are classified as online social networks, such as LinkedIn; many applications (e.g Quora.com), do not explicitly contain a social graph. Our system does not require a social graph, instead only requiring that a target web application allow users to create profiles that contain keyword tags and user generated text. More formally we require, a set of user profiles  $U$  where each user  $u \in U$  is annotated with a set of keyword tags  $A_u$ . Each profile  $u$  also contains a document  $d_u$  comprised of the user generated text, which can be items such as: profile descriptions, status updates, and blog posts.

DOBBY constructs a knowledge graph for keyword tags in a series of two components. The first component (described in section *Parent-child classifier*) of DOBBY predicts parent-child relationships for pairs of tags in  $A \times A$ . The parent-child prediction task is formulated as a supervised learning problem where each pair of tags  $(a_i, a_j) \in A \times A$  is assigned a real value  $p(a_i, a_j)$  that represents that probability that  $a_i$  is a parent of  $a_j$ . A threshold  $\tau$  is set such that the constructed graph contains all edges  $(a_i, a_j)$  where  $p(a_i, a_j) > \tau$ . Since the parent-child classifier makes predictions in an independent manner, reciprocal predictions can occur. The second component of DOBBY proposes strategies to remove reciprocal predictions using heuristics based on the classifier output and topology of the inferred knowledge graph.

## 5.3 Parent-Child classifier

The parent child classifier is a supervised model that takes as input a feature vector  $x$  corresponding to a pair of attributes  $(a_i, a_j)$  and produces a probability score for the likelihood that  $a_i$  is a parent skill of  $a_j$ . We first describe the features (grouped by type) used in the model and then our procedure for generating training data. For clarity, we refer to a pair of potential parent child skills as  $(p, c)$ , where  $p$  is the parent and  $c$  is the child attribute.

### 5.3.1 Co-occurrence Features

Co-occurrence features are all based on the frequency of which attributes are listed in user profiles. We denote  $U_a$  as the set of users *explicitly* annotated with the attribute  $a$ , i.e a user listing *Machine Learning* in their interest or expertise section of profile. The set  $\hat{U}_a$  consists of users that have the attribute  $a$  contained in their document  $d_u$  which we refer to as the set of users that *implicitly* list the attribute on their profile. For example, a user that mentions *machine learning* in their profile text or a status update would occur in this set. We include both “implicit” and “explicit” versions of some features because we found that users have different behavior when choosing attributes then when writing profile text. Users tend to not list broader attributes explicitly, but many times these broader attributes appear in their profile text.

**Jaccard Similarity:**

$$J(p, c) = \frac{|U_p \cap U_c|}{|U_p \cup U_c|}, \hat{J}(p, c) = \frac{|\hat{U}_p \cap \hat{U}_c|}{|\hat{U}_p \cup \hat{U}_c|} \quad (5.1)$$

The Jaccard score is used as a feature to determine how similar two attributes are based on how often they co-occur normalized by their individual frequencies. We implemented a Jaccard similarity score based on both the explicit and implicit counts.

**Conditional Probability:**

$$P_{p|c}(p, c) = \frac{|U_p \cap U_c|}{|U_c|}, \hat{P}_{p|c}(p, c) = \frac{|\hat{U}_p \cap \hat{U}_c|}{|\hat{U}_c|} \quad (5.2)$$

The conditional probability measures the degree to which attribute  $p$  subsumes  $c$ . if the value is 1 then this means that all of the times a user lists attribute  $c$  they also list attribute  $p$  indicating that  $p$  is most likely a parent concept. See [146], who were the first to propose using the conditional probability as a measure of subsumption. We implemented two versions

of the conditional probability score using both the explicit and implicit counts.

**Co-occurrence Count:**

$$CoocCount(p, c) = |U_p \cap U_c| \quad (5.3)$$

The number of times  $p$  and  $c$  are co-listed on user profiles. This feature is included so the model has knowledge of how sparsely a skill occurs within the data, since all the other count features depend on co-occurrence, the quality of these features depend on how frequently the two attributes co-occur.

**Co-occurrence Entropy:**

$$P(x|y) = \frac{|U_x \cap U_y|}{|U_y|}, \quad \bar{H}(x) = - \sum_{y \in U_y} P(x|y) \log(P(x|y)) \quad (5.4)$$

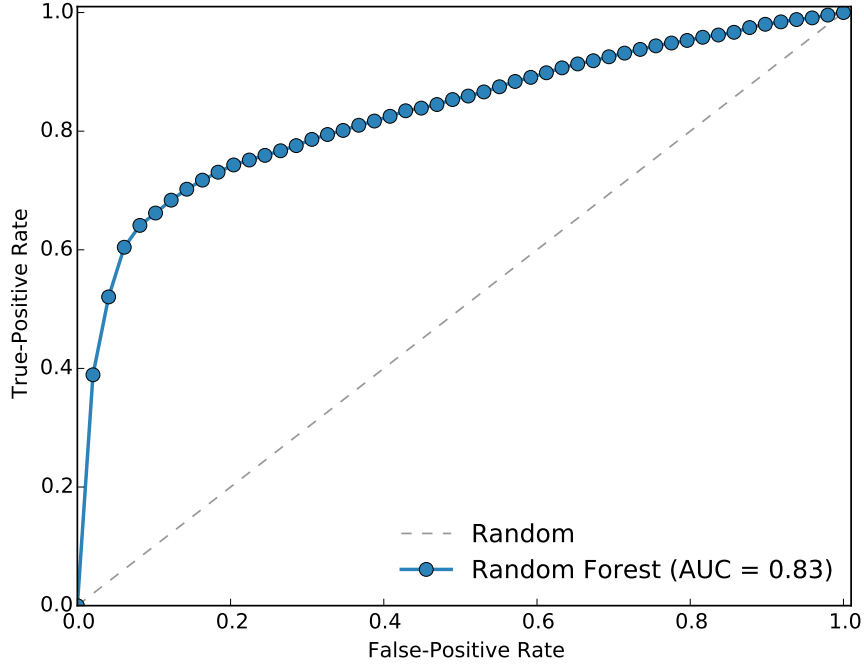
$$H(p, c) = \begin{cases} 1 & : \bar{H}(p) \geq \bar{H}(c) \\ 0 & : \bar{H}(p) < \bar{H}(c) \end{cases}$$

The entropy of an attribute’s co-occurrence distribution given by  $\bar{H}$  describes how general an attribute is. An attribute that is more narrow and specialized will most likely co-occur more often with other specialty skills therefore having a more peaked distribution, while broader skills will have a more uniform co-occurrence distribution and therefore a higher entropy. We encode the entropy as a binary feature with a value of 1 if the parent has a higher entropy than the child.

### 5.3.2 Network Features

Let  $\mathcal{N}_A = (A, E)$  be a similarity network comprised of attributes as nodes and the set of binary edges comprised of all attribute pairs that have a Jaccard similarity above a threshold  $\tau$ . We experimented with various values of  $\tau$  and did not see a significant change in feature quality, so we used  $\tau = 0.01$ . Centrality measures on a network compute the importance of a node which we use as a proxy for the generality of an attribute. As with the entropy feature, both centrality features were transformed into binary functions with value 1 if the parent has a higher centrality and 0 otherwise.

**Eigen Centrality:** Eigenvector centrality denoted by  $\bar{E}$  is a measure of a nodes importance measured by its corresponding eigenvalue in the adjacency of the network. The measure is very similar to PageRank [124] except applicable to a network without a stochastic adjacency



**Figure 5.2** ROC curve

matrix. See Newman [119] for a more complete description.

$\bar{E}$  : *eigenvector centrality score function* (5.5)

$$E(p, c) = \begin{cases} 1 & : \bar{E}(p) \geq \bar{E}(c) \\ 0 & : \bar{E}(p) < \bar{E}(c) \end{cases}$$

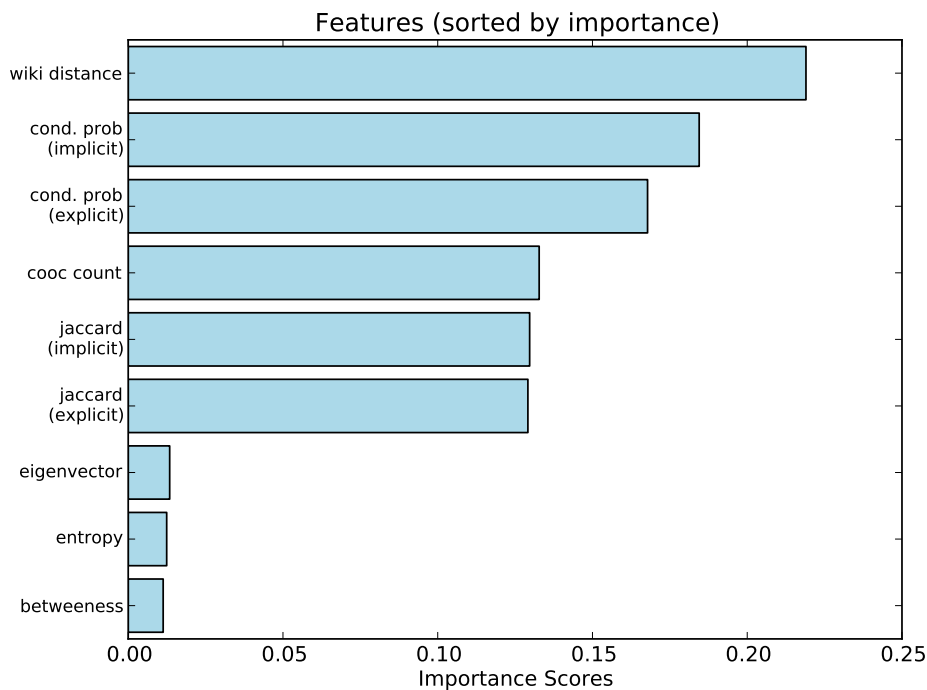
**Betweenness Centrality:** the betweenness centrality of a node  $v$ , denoted by  $\bar{B}$ , is computed as the number of shortest paths in the network that include  $v$ . See Newman [119] for a more complete description.

$\bar{B}$  : *betweenness centrality score function* (5.6)

$$B(p, c) = \begin{cases} 1 & : \bar{B}(p) \geq \bar{B}(c) \\ 0 & : \bar{B}(p) < \bar{B}(c) \end{cases}$$

### 5.3.3 Wikipedia Distance

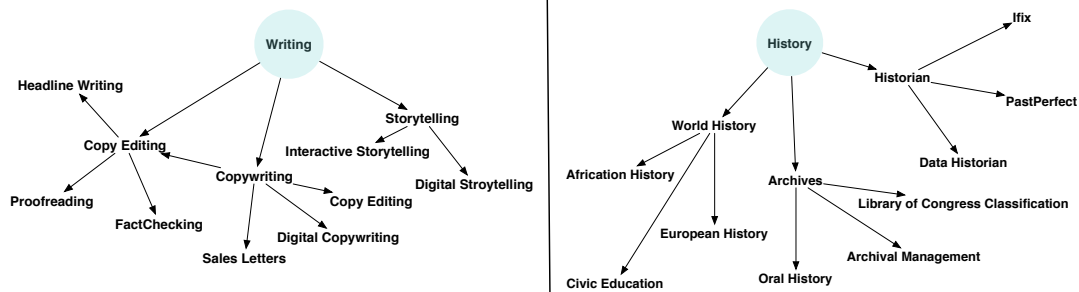
Each article in Wikipedia includes a set of categories tagged by human editors. Category tags are themselves articles with their own category tags. Given an article, one can extract



**Figure 5.3** Feature importance scores produced by the random forest model

a category hierarchy by crawling the category tags until reaching the root category. Since Wikipedia spans many concepts and entities over a variety of topics, many keyword tags have corresponding Wikipedia articles. Our Wikipedia distance feature function  $D_{wiki}$  takes as input a pair of attributes and outputs their distance in the Wikipedia category hierarchy. The closer two attributes are in the Wikipedia category hierarchy the more likely they are to be actual parent child pairs. We infer a direction on category tags because we crawl outward for each attribute, therefore the distance function is asymmetric.

For each attribute  $a$  we denote a set of category tags  $C_a = \{(t_i, \ell_i)\}$  where each category tag  $t_i$  has a corresponding level  $\ell_i$  that represents the number of hops the category is from the given attribute. for instance, for the attribute *Machine Learning*, the category *Artificial Intelligence* is a direct category so it is level 1, and *Computer Science* is a level 2 category because it is direct category of *Artificial Intelligence* but not *Machine Learning*. If a category appears multiple times in  $C_a$  we only include the tag with the smallest distance from  $a$ . We define the Wikipedia distance feature  $D_{wiki}(p, c)$  as the minimum level for which  $p$  appears as a category in  $C_c$ . For inputs where an attribute doesn't have any Wikipedia categories we have the feature output a special value  $-1$  or for the case where the parent attribute does not appear in the child's category set we have the feature output 999 to denote a maximum distance value.



**Figure 5.4** Sample output of DOBBY on two LinkedIn skills, *History* and *Writing*. These examples were generated by expanding out two levels from the root node, choosing the top-3 children for each parent.

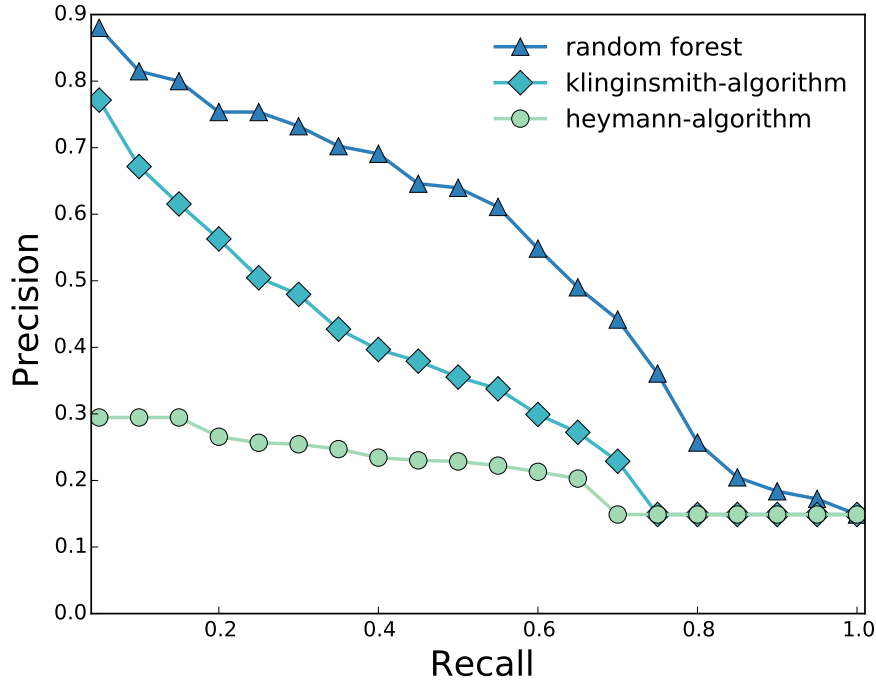
## 5.4 Evaluating DOBBY on LinkedIn Skills

We evaluated DOBBY using data derived from the LinkedIn social network. LinkedIn is the largest professional social network with over 300 million users and has a standardized collection of keyword tags called *Skills*. Skills cover topics ranging from “cooking” and “circus arts” to “computer science” and “marketing” and also vary in specificity, ranging from broad topics like “Mathematics” to the very narrow, such as “support vector machines”.

Skills have been standardized using a system developed by [8] that maps all phrases which refer to the same concept to their canonical term, i.e “data mining algorithms” maps to “data mining”. The skill’s dataset used in this chapter contains 118,314 phrases that map to a total of 39,134 unique skills. In order to compute the wikipedia feature described in section 5.3.3 we needed to obtain a URL for each skill. We obtained URLs for about 70% of the skills by using Amazon’s Mechanical Turk system. This process could also be done automatically by taking the top wikipedia result from a search engine query for the skill, most likely resulting in a noisier feature. We limit the pairs of skills we feed into the parent child classifier by requiring that the skills co-occur at least 10 times. Since many of our features rely on co-occurrence this can remove noise that is generated by pairs of skills that don’t co-occur often enough.

### 5.4.1 Training Data Generation

Acquiring a large set of manually labeled samples for training the parent-child classifier poses a significant challenge. Many skills belong to specific disciplines and can only be labeled by experts knowledgeable about the discipline. In addition, parent-child relationships are rare amongst random pairs of skills, making it expensive to obtain a large and balanced training set. We obtained positive examples of parent-child skills using Quora’s



**Figure 5.5** precision recall curves on hold-out test set

human curated topic hierarchy. Topics on Quora are similar in context to skills on LinkedIn, they are used to tag questions and overlap with many of the skills found on LinkedIn. Quora provides publicly available topic pages that consist of a topic description as well as user labeled parent and child topics on each page. We then mapped the scraped Quora topics to LinkedIn skills via the standardized skills dictionary described in [8].

Negative training examples are not explicitly provided in Quora’s topic hierarchy and therefore were heuristically generated. We generated negative examples by rewiring the skills in the positive examples using two rewiring strategies. The first strategy involved swapping all parent-child pairs in the positive training set, such that if  $A$  is a parent of  $B$  in the positive set, then  $B$  is a parent of  $A$  in the negative set. We generate additional negative examples by taking all pairs of skills that share a parent, and treating these “sibling” skills as negative examples. This second strategy can produce false negatives because it is possible that valid parent-child relationships exist between these automatically generated pairs. We minimize this noise by proposing a method for eliminating such pairs.

Some LinkedIn profiles contain subsumption statements such as “Programming: Java,Python,Ruby” which can be used to extract hierarchical relationships. We built a set of parent-child pairs extracted from these subsumption phrases to filter negative examples. If any of the negative examples extracted from Quora occur as a parent-child pair in this set, then we delete the negative example. While the set of examples we extracted from

the LinkedIn profiles was sufficient for eliminating incorrectly spurious negative pairs, it had to large of a false positive rate to be used for positively labeled data. The final number of negative examples total to 29,000 and were sampled to match the number of positive examples. The final, balanced, training set consists of 8,000 examples.

### **5.4.2 Cross Validation and Feature Importance**

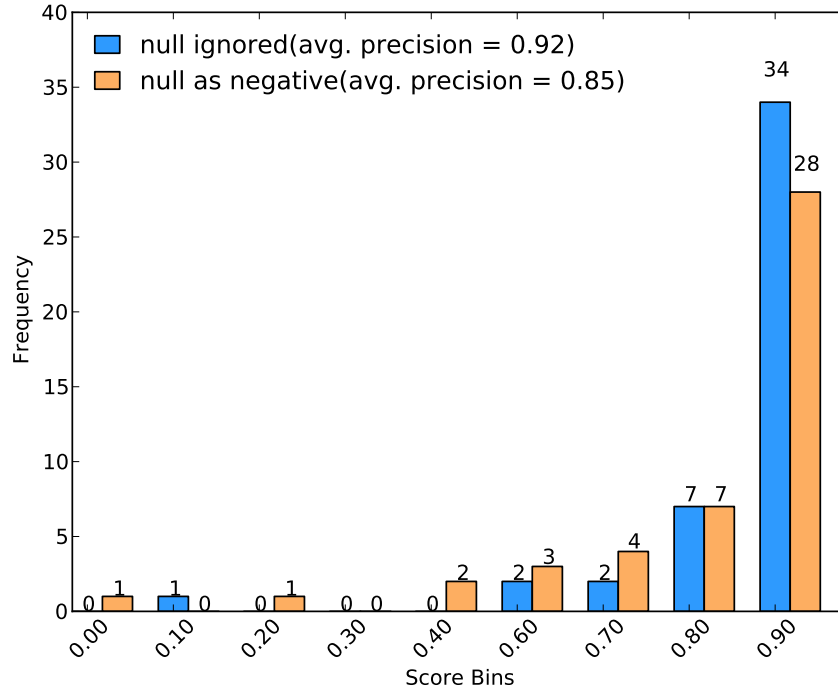
We first analyze the performance of the parent-child classifier using cross validation on the training set. Due to the fact that our features consist of both categorical and real-values we chose to use a random forest model, which has been shown to be one of the best classifiers [47]. We used the Sci-Kit Learn [127] implementation with default hyper parameter settings. Figure 5.2 shows the ROC curve generated by averaging curves generated from a 10-fold cross validation on the training set.

Like many machine learning systems, the features we designed are crucial for its success. Random forest models provide a simple and effective means for measuring relative feature importance. The depth of a feature indicates how discriminative it is with respect to the target variable because it determines the number of examples the feature contributes to. By averaging over the expected number of examples a feature contributes to, we can measure the relative feature importance for each feature. Figure 5.3 shows the relative feature importance for each of the features described in section 5.3. As the plot shows, 6 of the 9 features presented are almost equally important, with both versions of the conditional probability features and the wikipedia distance feature being the most important. The least important features are the entropy and centrality features, which is not too surprising as they only provide a signal about the relative importance and “broadness” of the two skills.

### **5.4.3 Algorithm Comparison**

The models in section 5.4.2 were trained and tested using cross validation on the training data. Since our training data is heuristically derived, we balanced the positive and negative examples in the training set because previous research [168] has shown this to be an effective strategy to counter class imbalance. In reality, we know that parent-child relationships between skills and other keyword tags to be rare; random pairs of skills are much more likely to be siblings of each other or not be related at all. Even with heuristic techniques that don’t nearly capture all of the negative examples, we generated negative examples that outnumbered the positive class 7:1. In order to simulate a more realistic test scenario, we





**Figure 5.6** distribution of precision scores on human evaluated skills

created a test data set that consisted of 10% of both the positive and negative examples, retaining the class imbalance.

We compare against two previous state-of-the-art methods from the related domain of hierarchy construction in tagging systems. The first algorithm, *heyman-baseline*, by [73], creates an ordering of skills using closeness centrality and then greedily adds parent-child edges in hierarchy if the similarity score is above a set threshold. In their original implementation, Heymann *et al.* propose an algorithm that generated a tree, connecting each skill to only its most related parent. In order to fairly compare against our proposed method which builds hierarchies that are not limited to tree structures, we implemented an extension proposed by Heymann *et al.* that allows for each skill to be connected to all parents that have a similarity value above the chosen threshold. The heyman-algorithm requires two user-specified parameters, the first being the threshold value for the edges in the skill similarity network and the other being the similarity threshold when building the hierarchy. The edge threshold of 0.105, was chosen as the mean Jaccard similarity of the positive training examples in the training set used in the previous section. The second parameter is the similarity threshold that is used in the hierarchy generation process, which is left as a free parameter used to generate the precision-recall curve in Figure 5.5. We implemented the heyman-algorithm using both cosine similarity and Jaccard similarity, finally choosing Jaccard similarity because it gave better performance.

The second algorithm we implemented for comparison, *klingsmith-baseline*, proposed in [90] uses the conditional probability to infer hierarchical relationships directly. For each skill pair  $p, c$  they include the edge  $p \rightarrow c$  if  $P(p|c) > \theta$  and  $P(c|p) < P(p|c)$  where  $\theta$  is a user-specified threshold. We evaluate their method by computing the precision-recall curve for  $\theta$  values in the range  $[0, 1]$ .

Figure 5.5 shows the precision-recall curves for our random forest model and both baseline methods. We can first observe that DOBBY substantially outperforms both baselines, showing that even on an imbalanced dataset the random forest classifier performs well. The *klingsmith-baseline* method does the best of the two previous techniques, achieving better performance than the *heyman-baseline* at most recall levels. Most interesting from this plot is that the random forest model significantly outperforms both the baseline techniques and the logistic regression model. The poor performance of the *heyman-baseline* can be contributed to the fact that parent-child edges in the hierarchy can only be constructed between nodes where the parent has a higher centrality in the network. We have shown that centrality is a weak feature in our dataset, and since the *heyman-baseline* enforces an ordering of nodes based on centrality, it misses many parent-child pairs.



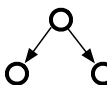

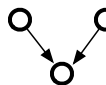
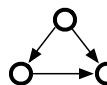
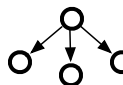
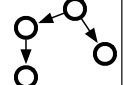
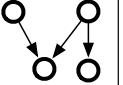
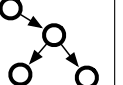
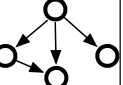
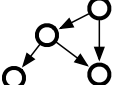
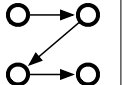
#### 5.4.4 Human Evaluation

In addition to evaluating our system on the Quora topic hierarchy we also ran a user study asking participants to evaluate parent-child skills pairs. Since parent-child pairs are rare, we needed to generate our evaluation set from the output of the random forest classifier. In order to obtain a knowledge graph that could be evaluated without worry that the evaluation examples overlap with the training examples, we generated the graph using a variation of the leave-one-out cross validation strategy. For each skill  $s$ , we predict children using the following process:

1. Remove all training examples that contain  $s$  as either a parent or child
2. Train the parent-child classifier
3. Predict children for  $s$

We chose to evaluate the random forest model with a threshold of 0.71 because it maximized the F1 score on the hold-out set described in the previous section.

Our user evaluation study enrolled 27 volunteer participants, all of whom are graduate students/Faculty and experts in a variety of disciplines including: history, technology, business, and sports. For each participant in the study we asked them to give us a list of

<b>Motif (size: 2)</b>			<b>Motif (size: 3)</b>				
<b>Frequency</b>	98.5%	1.5%	<b>Frequency</b>	74.2%	15.5%	6.4%	2.5%
<b>Motif (size: 4)</b>							
<b>Frequency</b>	55.1%	22.7%	8.5%	3.9%	3.4%	1.1%	0.9%

**Figure 5.7** A table showing motifs that comprise of at least 95% of all motifs of sub-graph size 2,3 and 4

their skills that are related to their domain of expertise and are valid skills in our dataset. Since we needed to obtain two labels we chose participants that overlapped in expertise and asked the second evaluator in each domain if they could evaluate the skill chosen by the first evaluator. For each skill we generated a random sample of 15 child skills predicted by our random forest classifier above the threshold. We then presented the skills to the participant who was asked to mark true, false or “I don’t know” for all 15 pairs. If a skill did not have 15 children above the threshold, then we output all of the predicted children of that skill (12 skills had fewer than 15 children). Through our study we were able to collect evaluations on 46 skills with 2 evaluators for a total of 586 parent-child pairs.

The results from the user-study are presented in Figure 5.6, which shows the distribution of precisions values over the evaluated skills. For this plot, we **ignored** all examples for which two users did not give the exact same answer. Since participants were capable of giving a null answer (option “I don’t know”), we calculated false-positives (FP) in two different ways. For the first way, labeled “null ignored” in Figure 5.6, all examples where both participants gave a null answer are ignored, leaving only the examples labeled “False” by both evaluators as false-positives. The other way, labeled “null as negative” in Figure 5.6, all examples that were labeled null by both evaluators were treated as false-positives in addition to examples labeled “False”. Skills are domain specific and can often be uninterpretable by non-experts, we included the more conservative “null as negative” method, because consensus on negative examples requires that both evaluators be able to detect a spurious skill that is not part their expertise. Results for both methods are good, with the most conservative estimate of average precision at 85%.

**Table 5.1** Table that shows the frequency of loops for sub-graph patterns ranging in size from 2-5. The second column shows the frequency of all loops, and the third column shows the frequency of patterns that contain loops larger than size 2.

pattern size	loop frequency	loop ( <i>size &gt; 2</i> ) frequency
2	1.5%	1.5%
3	1.4%	0.02%
4	1.7%	0.03%
5	2.2%	0.04%

## 5.5 Knowledge Graph Structure

We have shown that our supervised classifier produces superior parent-child pairs as compared to previous rule-based methods. Despite our superior performance on predicting parent-child pairs, an advantage of rule-based methods is that they can be easily constrained to not have loops in their resulting hierarchies. The heymann-algorithm imposes an ordering over nodes in the hierarchy using centrality scores as a constraint to disallow loops. The hierarchy constructed using the klinginsmith-algorithm is guaranteed to be loop free since parent-child pairs  $(x,y)$  only exist if  $P(x|y) > P(y|x)$  and  $P(x) > P(y)$ , therefore imposing an ordering on nodes by the frequency for which nodes appear in the dataset. From our experimental evaluation thus far, we have only evaluated the quality of parent-child pairs and have not analyzed the structure of the knowledge graph.

In order to determine the extent to which our constructed knowledge graph exhibits loops, we used the network motif finding algorithm, FanMod [170]. FanMod enumerates all sub-graphs of a given size and reports the frequency of each type of sub-graph pattern. A sub-graph pattern is defined as a group of subgraphs that are isomorphic to each other, where the frequency of a subgraph pattern is the number of sub-graphs in the isomorphic set. We generated the knowledge graph by predicting all child pairs for each skill in the LinkedIn dataset using the random forest classifier with a threshold of 0.71<sup>1</sup>. Figure 5.7 shows the most frequent sub-graph patterns of size 2,3 and 4, that comprise the 95<sup>th</sup> percentile of all subgraph patterns detected by FanMod. None of the sub-graph patterns shown in the table contain loops and are patterns to be expected in a high quality knowledge-graph.

In addition we calculated the total frequency of sub-graph patterns that contained loops and reported the percentages in column two of table 5.1. The frequency of loops is very small, ranging from a frequency of 1.5% to 2.2%. We found that our classifier would sometimes produce inconsistent parent-child-predictions in the form of predicting reciprocal

<sup>1</sup>The threshold 0.71 maximizes the F1 score of the classifier on the hold-out test set described in the previous section

relationships between skills. We calculated the impact of these reciprocal predictions by also measuring the frequency of patterns with loops greater than size 2, as shown in column 3 of table 5.1. The values in this column show that patterns containing loops of size 2 are indeed the most predominant, consisting of the majority of all patterns containing cycles. While the problem of enumerating all cycles in the graph is computationally hard, we can easily remove reciprocal edges that comprise most of the patterns containing loops. In the next section we propose two heuristics for removing reciprocal edges.

### 5.5.1 Pruning Reciprocal Edges

The output of the parent-child classifier gives probability estimates for all pairs in tags in  $A \times A$ . The ontology induced by this output, is a directed graph with reciprocal edges between all pairs. In order to induce an ontology from these estimates one must choose a suitable threshold and prune all edges below this threshold. The default threshold for a classifier is 0.5, but this is not necessarily optimal, and depending on ones application the tradeoff in precision and recall will inform the choice of threshold. We constructed the knowledge graph used for the sub-graph pattern analysis that optimized the F1 score, which gives equal weight to both precision and recall. Once a threshold is chosen, there is still no guarantee that there will not be reciprocating predictions. More formally, a reciprocal prediction from our parent-child classifier is defined as follows: Let  $\tau$  be the chosen threshold, a reciprocal prediction is one in which for a pair of tags  $a_i, a_j \in A$ ,  $p(a_i, a_j) > \tau$  and  $p(a_j, a_i) > \tau$ .

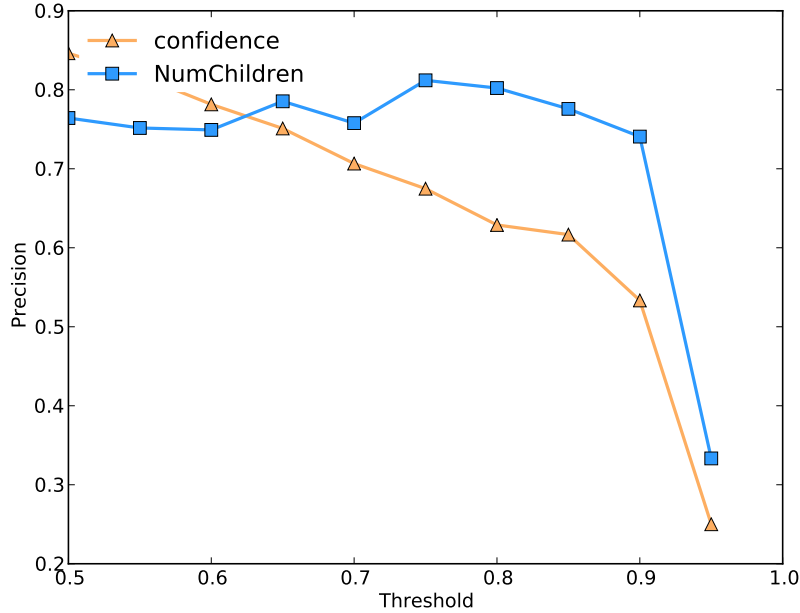
We propose two strategies for eliminating reciprocating predictions. Both strategies use information generated by the classifier to try and decide which attribute in the reciprocating pair is the true parent. Our first pruning heuristic simply chooses the parent as the attribute who is the parent in the edge with higher probability estimate. The second heuristic looks at the number of children that each attribute has above the threshold, and chooses the parent as the attribute with the higher number of children. For both heuristics, if the value is equal for both parent and child, we delete both edges. Below we give formal descriptions of each heuristic.

#### Confidence Pruner

Let tags  $(a_i, a_j)$  and  $(a_j, a_i)$  be edges that participate in a reciprocating prediction. the confidence pruner removes edge  $(a_i, a_j)$  if  $p(a_i, a_j) < p(a_j, a_i)$  else removed  $(a_j, a_i)$ .

#### NumChildren Pruner

Let tags  $(a_i, a_j)$  and  $(a_j, a_i)$  be edges that participate in a reciprocating prediction.



**Figure 5.8** Precision of edge pruning heuristics

Let  $N_{a_i}$  and  $N_{a_j}$  be the number of children for  $a_i$  and  $a_j$  for a selected threshold. The NumChildren pruner removes the edge  $(a_i, a_j)$  if  $N_{a_j} > N_{a_i}$  else removes  $(a_j, a_i)$

We evaluated the pruning rules on the graph of LinkedIn skills constructed using a modified leave-one-out training scheme (see section 5.4.4) which allowed us to use the entire training corpus to test the pruning heuristics. Figure 5.8 shows the performance of both pruning heuristics for the ontology constructed with a random forest parent-child classifier. The x-axis shows represents the edge-threshold of the classifier and the y-axis shows the precision. Both pruning method do about the same for lower threshold values but diverge at higher values. The NumChildren method works best for most threshold values, consistently achieving about 80% precision.

## 5.6 Previous Research in Social Tagging

Here we discuss the relationship of the work in this chapter to tagging systems, analysis of unstructured text, and related applications.

**Tagging Systems** — Tagging systems employed by websites such as *Delicious* and *Flickr*, allow users to tag items, *e.g.* webpages and photos, with keywords that describe their content. Research on the construction of tag hierarchies [25, 73, 90, 102, 129, 152] is a well studied and the most related to our problem. The bulk of these related works propose

rule-based algorithms that rely on heuristics such as similarity metrics and mined association rules. Many methods [90, 102, 152] have been proposed that use the conditional probability (calculated from co-occurrence) to infer parent-child pairs in addition to auxiliary rules that prune spurious edges from the constructed hierarchy. Examples of pruning rules include using similarity scores based on TF-IDF vectors of the tagged items to distinguish “sibling” relationships [152]; and the use of existing ontologies such as WordNet [102]. [73] uses the centrality of a tag in a tag-similarity network to determine the direction of edges in their induced hierarchy. DOBBY incorporates many of the rules presented in these works as features, including network centrality, Jaccard similarity, and the conditional probability. Since DOBBY incorporates a supervised learning model, it is more flexible and robust than algorithms that rely on rules. DOBBY also incorporates a post-processing step to ensure that simple loop structures are eliminated.

**Unstructured Text** — Similar to tagging systems, hierarchy construction has also been applied to entities extracted from natural language text [17, 33, 48, 92, 163, 167, 180] for a complete survey see [171]. Wei *et al.* [167] and Dakka *et al.* [33] both propose systems for building faceted taxonomies by using relationships found in Wikipedia and WordNet. Wei *et al.*’s use of Wikipedia is different from our own since they extract hyponym relationships using the link structure of articles and we use Wikipedia categories. Many approaches have also been proposed that utilize supervision from humans [17, 48, 92], utilizing crowd sourcing and semi-supervised methods for building concept hierarchies. Ontology construction from natural language text is similar to our problem in that it is concerned with mining hierarchical relationships. The natural language domain is different since text documents have a linguistic structure and in our domain we are concerned with organizing unstructured keyword attributes. We utilize natural language text to extract co-occurrence based features as well as keyword tags associated with user profile information.

**Applications** — In addition to research on constructing concept hierarchies, there is a related area of research focused on the applications of such hierarchies as well as flat keyword attributes for applications including search [37, 54, 133], user modeling [12, 52, 148] and community detection [116, 144]. Schwarzkopf *et al.* [148] propose a method that used a tag hierarchy to infer high level information about users on websites such as Delicious.com. Gauch *et al.* [54] and Pound *et al.* [133] both propose search systems that enable users to browse their search results using faceted based interfaces. Demeo *et al.* [37] propose a method that uses tag hierarchies to enhance search by expanding queries. Keyword attributes have been shown to be useful in the discovery of communities on social networks and tagging systems. The knowledge graph produced by DOBBY can be used to group skills that share common parent nodes. Such groupings can be leveraged by the community detection

algorithms proposed in [116, 144].

## 5.7 Discussion and Future Work

While DOBBY performs well in most situations, there are exceptions. For example, we observed some skills such as “R” ( the stats software package), that are listed by users with many professions and occur frequently throughout the dataset. From the perspective of classifier “R” seems like a much more general skill then it actually is, therefore having many more children then it should. One proposed method for dealing with this problem is limit the scope and build many graphs that represent specific groups of users. Such groups can be dictated by users professions or communities they belong to. By limiting the tags to a more restrictive set, skills like “R” won’t be as ubiquitous and the classifier will most likely do a better job at inferring edges for these types of skills.

Our parent-child classifier builds the knowledge graph in a bottom-up fashion and as a result there is no global constraint on the topology of the inferred graph. One advantage of our method making independent edge predictions, is that it can be easily parallelized to scale to very large tag vocabularies. In Section 5.5 we proposed heuristics for pruning simple inconsistencies like reciprocal edges but it is possible for our method to infer graphs that have cycles of arbitrary length. In our analysis of cycles we were able to compute frequencies of patterns up to size 5 and showed that the frequency of cycles in patterns was negligibly small, especially after removing reciprocal edges. The number of possible cycles in a graph is exponential, so even if we had pruning rules for arbitrary configurations, pruning all inconsistencies is not feasible. While there may be some applications that require a strict ontology, we anticipate many applications benefiting from our technique, especially since the occurrence of cycles is so rare.

One distinct advantage of our system compared to previous methods, is the possibility for the system to be improved from user input. We built the LinkedIn skills graph using training data derived from Quora, since obtaining labels from unlabeled pairs posed to expensive of a challenge. Now that we have a model that is capable of accurately predicting parent-child pairs, we can have users provide labels for edges in graph. These additional labels could be used to re-train DOBBY and produce a better ontology. This is only possible because DOBBY used a supervised classifier, rather than a rule-based algorithm.

this chapter focused on a methodology for constructing subsumption hierarchies, but another outcome of this research is that we constructed an knowledge graph for all skills on LinkedIn. We anticipate many applications, including news recommendation, search



engines, and ad placement can all benefit from the flexible and robust user modeling enabled by our constructed graph. DOBBY can also be used to construct graphs of keyword tags on other expertise driven social networks, such as Quora.com and StackExchange. These sites could leverage a knowledge graph for building better mechanisms to help match questions with experts respondents.

## **5.8 Conclusion**

Keyword tags have become an important and common feature of expertise driven social web applications. The unstructured nature of tag creation, enables users to generate large and descriptive tag vocabularies but at a cost. The tag vocabularies that result from these tagging systems are unstructured, lacking organization that describes relationships between tags. In this chapter we proposed a system DOBBY, for constructing a knowledge graph of keyword tags. DOBBY uses a random forest based classifier to predict typeOf relationships between tags, constructing a knowledge graph of tags from the bottom up. On our test dataset from the LinkedIn social network, DOBBY performs well, achieving high average precision/recall and substantially out-performing previous methods.

# Chapter 6

## Recovering Hidden Documents from Observed Text

All networks can be represented as adjacency matrices and all matrices can be represented as networks. This duality allows for network analysis to be applied to problems which have associated similarity matrices but, not an inherent network representation. While all similarity matrices can be represented as a network, this representation is only useful when the resulting network has properties that are indicative of networks (e.g small diameter, power-law degree distribution)<sup>1</sup>. The benefit of representing a similarity matrix as a network is that network algorithms can be used to analyze the structure and obtain insights into the underlying data.

Community detection algorithms offer several advantages over clustering algorithms designed for similarity matrices. Most community detection algorithms work without the need to specify the number of clusters, and are designed for sparse data. In this chapter we present an application of network inference for the problem of identifying clusters of bills that exhibit text reuse in a corpus of state legislation. The network is inferred by first computing a similarity matrix based on shared n-grams between documents. The resulting similarity matrix is sparse and therefore we use a community detection algorithm in order to identify groups of bills that exhibit text reuse. The network inference and community detection is a component of a system, LOBBYBACK, that we propose for identifying and summarizing the phenomena of text re-use in state legislation. The work presented in this chapter is based off of work presented in a paper currently under peer review [19].

---

<sup>1</sup>Transformations can be applied to the resulting network in order to obtain a network with the desired network properties. WGCNA, a gene clustering network analysis pipeline, modifies the edge weights of the resulting network in order to obtain a network with a power-law degree distribution.

## 6.1 Problem Overview

Beginning in 2005, a number of states began passing “Stand Your Ground” laws—legal protections for the use of deadly force in self-defense. Within a few years, at least two dozen states implemented a version of the this legislation [53]. Though each state passed its own variant, there is striking similarity in the text of the legislation. While seemingly “viral” the expedient adoption of these laws was not the result of an organic diffusion process, but rather more centralized efforts. An interest group, in particular the American Legislative Exchange Council (ALEC), drafted model legislation (in this case modeled on Florida’s law) and lobbied to have the model law enacted in other states. While the influence of the lobbyists through model laws grows, the hidden nature of their original text (and source) creates a troubling opacity.

Reconstructing such hidden text through analysis of observed—potentially highly mutated—copies poses an interesting and challenging NLP problem. We refer to this as the *Dark Corpora* problem. Since Legislatures are not required to cite the source of the text that goes into a drafted bill, the bills that share text are unknown beforehand. The first problem therein lies in identifying clusters of bills with reused text. Once a cluster is identified, a second challenge is the reconstruction of the original or prototype bill that corresponds to the observed text. The usual circumstances under which a model law is adopted by individual states involves “mutation.” This may be as simple as modifying parameters to the existing policy (e.g., changing the legal limit allowed of medical marijuana possession to 3.0 ounces from 2.5) or can be more substantial, with significant additions or deletions of different conditions of a policy. Interestingly, the need to maintain the objectives of the law creates a pressure to retain a legally meaningful structure and precise language—thus changes need to satisfy the existing laws of the state but carry out the intent of the model. Both subtle changes of this type, and more dramatic ones, are of great interest to political scientists. A specific application, for example, may be predicting likely spots for future modifications as additional states adopt the law. Our challenge is to identify and represent “prototype” sentences that capture the similarity of observed sentences while also capturing the variation.

## 6.2 Proposed Solution

In this chapter we propose LOBBYBACK, a system that automatically identifies clusters of documents that exhibit text reuse and generates “prototypes” that represent a canonical version of text shared between the documents. In order to synthesize the prototypes, LOBBYBACK first extracts clusters of sentences, where each sentence pertains to the same

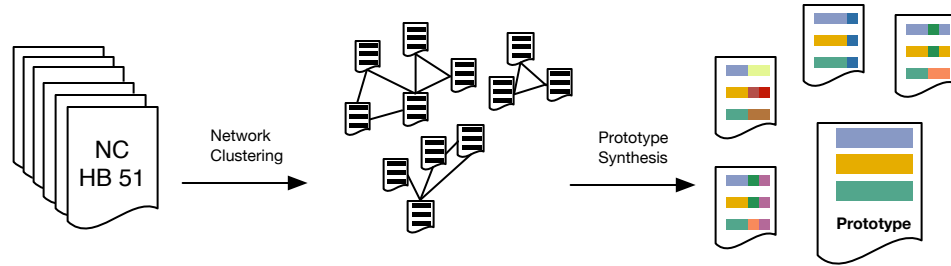
policy but can exhibit variation. LOBBYBACK then uses a greedy multi-sequence alignment algorithm to identify an approximation of the optimal alignment between the sentences. Prototype sentences are synthesized by computing a consensus sentence from the multi-sentence alignment. As sentence variants are critical in understanding the effect of the model legislation, we can not simply generate a single “common” summary sentence as a prototype. Rather, LOBBYBACK creates a data structure that captures this variability in text for display to the end-user.

Policy diffusion is a common phenomenon in state bills [10, 60, 63, 150]. Unlike members of the (Federal) Congress, few state legislators have the expertise, time, and staff to draft legislation. It is far easier for a legislator to adapt existing legislative text than to write a bill from scratch. As a consequence, state legislatures have an increased willingness to adopt legislation drafted by interest groups or by legislators in other states [81]. In addition to states borrowing text from other legislators and lobbyists, another reason why bills can exhibit text reuse is when a new federal law passes and each state needs to modify its existing policy to conform with the new federal law.

The result of legislative copying, whether caused by diffusion between states, influence from a lobby or the passing of a new federal law, is similar: a cluster of bills will share very similar text, often varying only by implementation details of a given policy. The goal in constructing a prototype document—a representation of the “original” text—is to synthesize this document from the modified copies. In the case when the bill cluster was influenced by one external source, such as lobby or passage of a federal bill, the ideal prototype document would capture the language that each bill borrowed from the source document. In the case when there is not one single document that influenced a cluster of bills, the prototype will still give a summary of a concise description of the diffused text between bills, providing fast insight into what text was shared and changed within a bill cluster.

With LOBBYBACK end-users can quickly identify clusters of text reuse to better understand what type of policies are diffused across states. In other applications, sentence prototypes can be used by journalists and researchers to discover previously unknown model legislation and the involvement of lobbying organizations. For example, prototype text can be compared to the language or policy content of interest groups documents and accompanied with qualitative research it can help discover which lobbyists have drafted this legislation.

We evaluated LOBBYBACK on the task of reconstructing 122 known model legislation documents. Our system was able to achieve an average of 0.6 F1 score based on the number of prototype sentences that had high similarity with sentences from the model legislation. We have also run LOBBYBACK on the entire corpus of state legislation (571,000 documents)



**Figure 6.1** Diagram describing processing steps of LOBBYBACK

from openstates.org as an open task. The system identified 4,446 clusters for which we generated prototype documents. LOBBYBACK is novel in fully automating and scaling the pipeline of model-legislation reconstruction. The output of this pipeline captures both the likely “source sentences” but also the variations of those sentences.

LOBBYBACK consists of 3 major components. The first component identifies clusters of bills that have text reuse. Then for each of these bill clusters, LOBBYBACK extracts and clusters the sentences from all documents. For each of the sentence clusters, LOBBYBACK synthesizes prototype sentences in order to capture the similarity and variability of the sentences in the cluster.

### 6.3 State Legislation Corpus

We obtained the entire openstates.org corpus of state legislation, which includes 550,000 bills and 200,000 resolutions for all 50 states. While for some states this corpus includes data since 2007, for the majority of states we have data as early as 2010 . We do not include in our analysis data from Puerto Rico, where the text is in Spanish, and from DC, whose data includes many idiosyncrasies (e.g. correspondence from city commissions introduced as bills). On average, each state introduced 10,524 bills, with an average length of 1205 words .

### 6.4 LOBBYBACK Architecture

LOBBYBACK consists of 2 major components as pictured in Figure 6.1. The first component identifies clusters of bills that have text reuse. Then for each of these bill clusters, LOBBYBACK extracts and clusters the sentences from all documents. For each of the sentence clusters, LOBBYBACK synthesizes prototype sentences in order to capture the similarity

and variability of the sentences in the cluster.

### 6.4.1 Clustering Bills

The first component of LOBBYBACK identifies clusters of bill documents that have text reuse. These represent candidate bills that have been potentially copied from model legislation. There are a number of ways one could identify such clusters through text mining. In our implementation, we have opted to generate a network representation of the bills and then use a network clustering (i.e., “community-finding”) algorithm to generate the bill clusters. In our network representation each node represents a state bill and weighted edges represent the degree to which two bills exhibit substantive text reuse. Since most pairs of bills do not have text reuse, we chose to use a network model because community finding algorithms work well on sparse data and don’t require any parameter choice for the number of clusters. In the context of this work, text reuse occurs when two state bills share:

1. Long passages of text, e.g (sections of bills) that *can* differ in details.
2. passages which contain text of substantive nature to the topic of the bill (i.e., text that is *not* boilerplate).

In addition to text that describe policy, state bills also contain boilerplate text that is common to all bills from a particular state or to a particular topic. Examples of legislative boilerplate include: “Read first time 01/29/16. Referred to Committee on Higher Education” (meta-data describing where the bill is in the legislative process); and “Safety clause. The general assembly hereby finds, determines, and declares . . .” (a standard clause included in nearly all legislation from Colorado, stating the eligibility of a bill to be petitioned with a referendum).

In order to identify pairs of bills that exhibit text reuse, we created an inverted index that contained  $n$ -grams ranging from size 4-8. We used the open-source search engine ElasticSearch to implement the inverted index and computed the similarity between bills using the “More like This” (MLT) query [40]. The MLT query first selects the 50 highest scoring TF\*IDF  $n$ -grams from a given document and uses those to form a search query. The MLT query is able to quickly compute the similarity between documents and since it ranks the query terms by using TF\*IDF the query text is more likely to be substantive rather than boilerplate. By implementing the similarity search using a TF\*IDF cutoff we were able to scale the similarity computation for all pairs of bills while still maintaining our desire to identify reuse of substantive text.

The edges of the bill similarity network are computed by calculating pairwise similarity. Each bill is submitted as input for an MLT query and scored matches are returned by the search engine. Since the MLT query extracts  $n$ -grams only for the query document, the similarity function between two documents  $d_i$  and  $d_j$  is not symmetric. We construct a symmetric bill similarity network by taking the average score of each  $(d_i, d_j)$  and its reciprocal  $(d_j, d_i)$ . A non-existent edge is represented as an edge with score 0. We further reduce the occurrence of false-positive edges by removing all edges with a score lower than 0.1. The resulting network is very sparse, consisting of 35,346 bills that have 1 or more neighbors, 125,401 edges, and 3534 connected components that contain an average of 10 bills.

A specific connected component may contain more than one bill cluster. To isolate these clusters in the bill network we use the InfoMap community detection algorithm [142]. We use the InfoMap algorithm because it has been shown to be one of the best community detection algorithms and it is able to detect clusters at a finer granularity than other methods. Our corpus contains both bills that have passed and those that have not. Bills can often be re-introduced in their entirety after failing the previous year. As we do not want to bias the clusters towards bills that are re-introduced more than others, we filter the clusters such that they only include the earliest bill from each state .

## 6.4.2 Prototype Synthesis

Once we have identified a cluster of bills that have likely emerged from a single “source” we would like to construct a plausible representation of that source. The prototype synthesizer achieves this by constructing a canonical document that captures the similarity and variability of the content in a given bill cluster. The two main steps in prototype synthesis consists of clustering bill sentences and generating prototype sentences from the clusters.

### Sentence Clustering

Most state bills have a common structure, consisting of an introduction that describes the intent of the bill followed by sections that contain the law to be implemented. Each section of a bill is comprised of self-contained policy, usually consisting of a long sentence that describes the policy and the implementation details of that policy. Each document is segmented into these policy sentences using the standard Python NLTK sentence extractor. Sentences are cleaned by removing spurious white space characters, surrounding punctuation and lower-casing each word. Once we have extracted all of the sentences for a given bill cluster, we compute the cosine similarity between all pairs of sentences which are represented using

**Sec. 556. (1) An employer of labor in this state, employing both males and females, who shall not discriminate in any way in the payment of wages as between sexes who are similarly employed.**

**(2) A Person who violates this section is guilty of a misdemeanor . No female shall be assigned any task disproportionate to her strength, nor shall she be employed in any place detrimental to her morals, her health or her potential capacity for motherhood.**

**Any PUNISHABLE AS FOLLOWS:**

**(a) If the person has 1 to 15 employees, a fine of not more than \$500.00**

**(b) If the person has 16 to 50 employees, a fine of not more than \$1,000.00.**

**(c) If the person has more than 50 employees, a fine of not more than \$500.00 \$2,000.00**

**(3) This section does not prohibit the person from being charged with, convicted of, or punished for any other violation of law arising out of violation of this section**

**(4) A difference in wage rates based upon a factor other than sex does not violate this section.**

**Figure 6.2** An sample state bill segment from Michigan SB 343 (2011)

a unigram bag-of-words model. We used a simple unweighted bag of words model because in legal text stop words can be import<sup>2</sup> In this case, we are generating a similarity “matrix” capturing sentence–sentence similarity.

Given the similarity matrix, our next goal is to isolate clusters of variant sentences that likely came from the same source sentence. We elected to use the DBscan [43] algorithm to generate these clusters. The DBscan algorithm provides us with tunable parameters that can isolate better clusters. Specifically, the parameter  $\epsilon$  controls the maximum distance between any two points in the same neighborhood. By varying  $\epsilon$  we are able to control both the number of clusters and the amount of sentence variation within a cluster. A second reason for selecting DBscan is that the algorithm automatically deals with noisy data points, placing all points that are not close enough to other points in a separate cluster labeled “noise.” Since many sentences in a given bill cluster do not contribute to the reused text between bills, the noise cluster is useful for grouping those sentences together rather than having them be outsiders in “good” clusters.

<sup>2</sup>The difference between the words “shall” and “may” for instance is important, the former requires that a specific action be put on a states budget while the later does not while the later does not



## Multi-Sequence Alignment

Once we have sentence clusters we then synthesize a “prototype” sentence from all of the sentences in a given cluster. An ideal prototype “sentence” is one that simultaneously captures the similarity between each sentence in the cluster (the common sentence structures) and the variation between the sentences in a cluster. For a simple pair of (partial) sentences, “The Department of Motor Vehicles retains the right to . . .” and “The Department of Transportation retains the right to . . .”, a prototype might be of the form, “The Department of { Motor Vehicles, Transportation } retains the rights to . . .” Our “sentence” is not strictly a single linear piece of text. Rather, we have a data structure that describes alternative sub-strings and captures variant text.

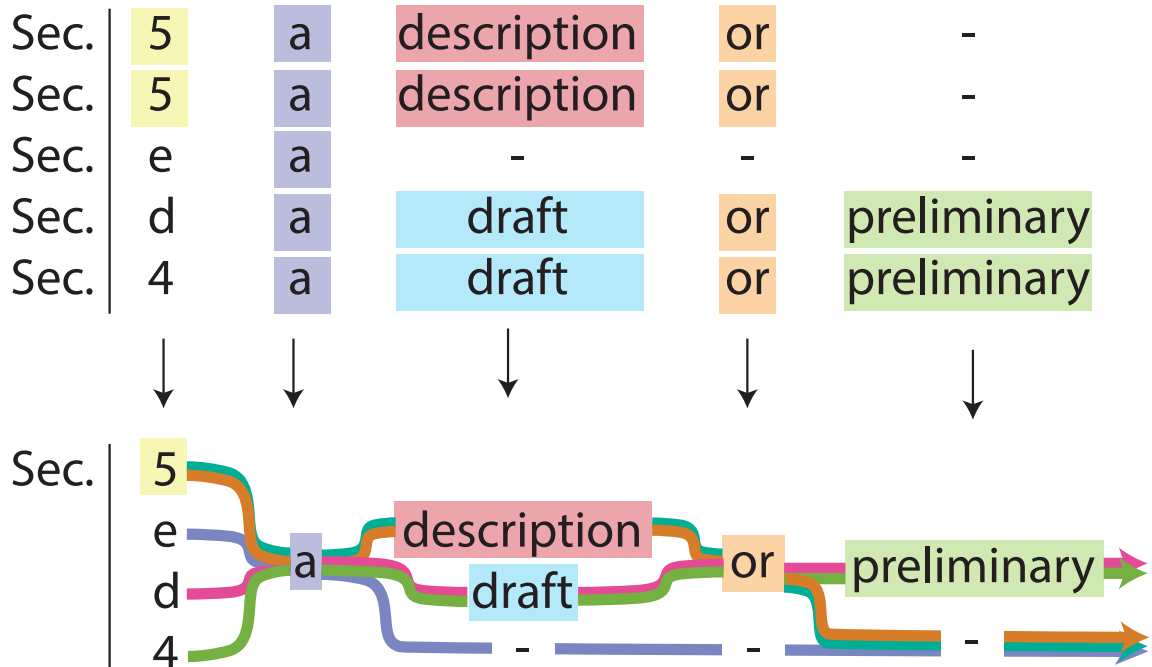
To generate this structure we propose an algorithm that first computes an approximation of the optimal multi-sentence alignment (MSA) in the cluster and then generate a prototype sentence that represents a consensus between the sentences in the MSA.

We generate an MSA using a modified version of the iterative pairwise alignment algorithm described in [62]. Gusfield proposes a greedy strategy that builds a multi-alignment by iteratively applying the NeedlemanWunsch pairwise global alignment algorithm. Needleman-Wunsch is a dynamic-programming algorithm that computes the optimal pairwise alignment by maximizing the alignment score between two sentences. An alignment score is calculated based on three parameters: word matches, word mismatches (when a word appears in one sentence but not the other), and gaps (when the algorithm inserts a space in one of the sentences).

The algorithm we use to generate an MSA is described involves the following steps:

1. Construct an edit-distance matrix for all pairs of sentences
2. Construct an initial alignment between the two sentences with the smallest edit distance
3. Repeat this step  $k$  times:
  - (a) Select the sentence that has the smallest average edit distance to the current MSA.
  - (b) Add the chosen sentence to the MSA by aligning it to the existing MSA.

The algorithm stops after the alignment has reached a size that is determined as a free parameter  $k$  (decided by the user but can also be chosen to be the total number of sentences in the cluster). Since the algorithm follows an order based on the edit distance between the



**Figure 6.3** A visualization of a multi-sentence alignment and the resulting prototype sentence.

current MSA and the next sentence to be added, the larger the MSA the more variation we are allowing in the prototype sentence.

### Synthesizing Prototype Sentences

We synthesize a prototype sentence by finding a *consensus* sentence from all of the aligned sentences in the MSA for a given cluster. We achieve this by going through each “column” of the MSA and using the following rules to decide which token will be used in the prototype. A token can be either a word in one of the sentences or a “space” that was inserted during the alignment process.

1. If there is a token that occurs in the majority of alignments ( $> 50\%$ ), that token is chosen
2. If no token appears in a majority, a special variable token is constructed that displays all of the possible tokens in each sentence. For example the 1<sup>st</sup> and 3<sup>rd</sup> columns of the example in Figure .
3. If a space is the majority token chosen then it is shown as a variable token with the second most common token

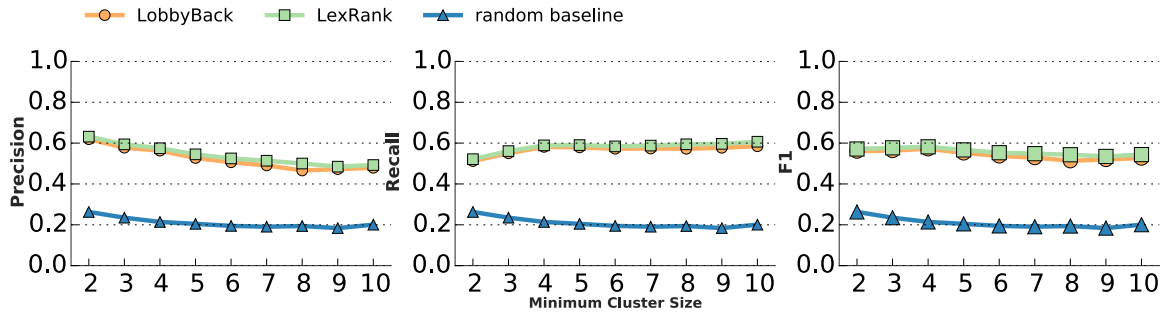


Figure 6.4 Precision, Recall, and F1 scores of LOBBYBACK and baselines

## 6.5 Evaluation

We provide two experiments that evaluate both the sentence clustering and prototype synthesis components of LOBBYBACK.

### 6.5.1 Model Legislation Corpus

In order to test the effectiveness of LOBBYBACK in recreating a model bill, we first identified a set of known model bills. Our model legislation corpus consists of 1846 model bills that we found by searching on Google using the keywords "model law", "model policy" and "model legislation". Most of the corpus is comprised of model bills from the conservative lobby group, American Legislative Exchange Council (ALEC), its liberal counterpart, the state innovative exchange (SIO) and a non-partisan group the Uniform Law Commission (ULC). The rest of the documents come from smaller interest groups that focus on specific issues.

Using the clusters we previously described (Section 6.4.1), we found the most similar cluster to each model bill. This was done by first computing the set of neighbors for a model bill using the same procedure used in creating the bill similarity network. We then matched a bill cluster to the model legislation by finding the bill cluster that had the highest Jaccard similarity with the neighbor set of a model bill. does this make more sense now? Each test example in our evaluation data set consists of model bill and its corresponding bill cluster. The total number of model legislation documents that had matches in the state bill corpus was 360 documents.

Once we have an evaluation data set comprised of model bill/cluster pairs our goal is to compare the prototype sentences we infer for a cluster to the model bill that matches that cluster. Since we don't have ground truth on which sentences from the model bill match sentences in the documents that comprise the cluster we need to infer such labels. In order

to identify which sentences from the model legislation actually get re-used in the bill cluster, we take the following steps:

1. Extract all sentences from each of the bills in a cluster and the sentences in the corresponding model legislation.
2. Compute the pairwise cosine similarity between bill sentences and each of the model bill sentences using the same unigram bag-of-words model described in Section 6.4.2
3. Compute the “oracle” matching  $M^*$  using the Munkres algorithm [115]

The Munkres algorithm gives the best possible one-to-one matching between the sentences in the model legislation and the sentences in the bill clusters. There are some sentences in the model bill that are never used in actual state legislation (e.g., sentences that describe the intent of a law or instructions of how to implement a model policy). Therefore we label model bill sentences in  $M^*$  that match a bill sentence with a score greater than 0.85 as true matches ( $S^*$ )<sup>3</sup>. The final set of 122 evaluation examples consists of all model legislation/bill cluster pairs where more than 50% of model bill sentences have true matches.

## 6.5.2 Baselines

While no specific baseline exists for our problem, we implemented two alternatives to test against. The first, Random-Baseline, was implemented simply to show the performance of randomly constructing prototype documents. The second, LexRank-Baseline, implements a popular extractive summarization method.

**Random-Baseline** – The *random-baseline* randomly samples sentences from a given bill cluster. The number of sentences it samples is equal to the number of sentences in the optimal matching  $|M^*|$

**LexRank-Baseline** – The LexRank baseline uses the exact same clustering algorithm as LOBBYBACK except instead of synthesizing prototype sentences, it uses the LexRank algorithm [42] to pick the most salient sentence from each of the sentence clusters.

## 6.5.3 Evaluating Sentence Clusters

We first evaluate the quality of the sentence clusters using the optimal matching  $M^*$  described above. For each test example in the evaluation set we generate a prototype document

---

<sup>3</sup>A threshold of 0.85 was found effective in prior work [53] and we observed good separation between matching sentences and non-matches in our data-set.

using LOBBYBACK and each of the baselines described above. We then compute a matching  $M$  between the prototype sentences and the model bill sentences (using the same procedure described in 6.5.1), where  $S$  is the set of sentences in the prototype and  $S_{0.85}$  is the set of sentences that match with a score greater than 0.85. We compute precision,  $P$ , as  $P = |S_{0.85}|/|S|$ , and recall,  $R$ , as  $R = |S|/|S^*|$ .

Figure 6.4 shows precision, recall and F1 scores for both baselines and LOBBYBACK. Each curve is generated by averaging the precision/recall/F1 scores computed for each of the examples in the test set. The  $x$ -axis represents the minimum bill cluster size of the test examples for which the score is computed. For example, a minimum cluster size would average over all test examples with at least 2 bills in a cluster. LOBBYBACK relies on the fact that if text was borrowed from a model bill, then it would have been borrowed by many of the bills in the cluster. By analyzing how LOBBYBACK performs with respect to the minimum cluster size, we can determine how much evidence LOBBYBACK needs in order to construct clusters that correspond to sentences in the model bills. The performance of LOBBYBACK and the LexRank baseline substantially improves over the random baseline, while the difference between LOBBYBACK and LexRank is negligible. Since our cut-off similarity is 0.85, all sentences above the threshold are treated as true positives, making the distinction between the LexRank baseline and system small. LOBBYBACK performs a little worse than LexRank for large cluster sizes because it is penalized for having space and variable tokens which don't occur in model bills. Space and variable tokens occur more frequently in prototype sentences in larger clusters because there is more variation in the sentence clusters.

## Evaluating Sentence Synthesis

The experiment in the previous section evaluated the quality of the sentence clusters by treating all matching sentences with a similarity greater than 0.85 as true positives. Here we provide an evaluation of the synthesized sentences that LOBBYBACK generates and compare them to the LexRank baseline, which chooses the most salient sentence from each cluster. We evaluate the quality of the synthesized prototype sentences by computing the word-based edit-distance between the prototype sentence with its corresponding model bill sentence in  $S$  for each test example.

Since the prototypes contain variable and space tokens which do not occur in the model bill sentences we modify the standard edit distance algorithm by *not* penalizing space tokens and allowing for any of the tokens that comprise a variable to be positive matches. In addition, we remove punctuation and lowercase all words in all of the sentences, regard-

Method	$\epsilon$	Min Cluster Size			
		2	4	6	8
LOBBYBACK	0.1	<b>24.4</b>	<b>20.4</b>	<b>18.2</b>	<b>17.2</b>
LexRank	0.1	25.4	22.5	20.3	19.4
LOBBYBACK	0.15	<b>25.5</b>	<b>21.6</b>	<b>19.4</b>	<b>17.9</b>
LexRank	0.15	27.3	25.6	25.0	24.1

**Table 6.1** Mean edit distance scores for LOBBYBACK and LexRank

less of method. We generate the results in Table 6.1 by averaging the edit distance for a configuration of LOBBYBACK or LexRank over sentence clusters produced for each test example. LOBBYBACK was configured to run with the number of iterations set to the size of the sentence cluster.

We compared both the performance of LOBBYBACK and LexRank for DBscan  $\epsilon$  values of 0.1 and 0.15 as well as computing the average edit distance for different minimum sizes of cluster values. As the table shows, LOBBYBACK obtains a lower edit distance than LexRank in every configuration and as the size of the clusters increase the gap between the two increases. The goal of LOBBYBACK is not to be a better summarization algorithm than LexRank. By comparing to LexRank and showing that the edit distances are smaller on average, we can conclude that the prototype sentences created by LOBBYBACK are capturing the text that is “central” or similar within a given cluster. In addition, the prototype sentences produced by LOBBYBACK are superior because they also capture and describe in a succinct way, the variability of the sentences within a cluster.

## 6.6 Discussion

One assumption that we made about the nature of state adoption of model legislation is that the legislatures make modifications that largely preserve the model language in an effort to preserve policy. However, we currently do not consider cases in which a legislature has intentionally obscured the text while still retaining the same meaning. While not as frequent as common text reuse, Fernandez *et al.* [72] observed that some legislatures almost completely changed the text while reusing the concepts. One area of future work would be to try and extend LOBBYBACK to be more robust to these cases. One potential way of extending LOBBYBACK would be to allow for a more flexible representation of text, such as word vector embeddings. The embeddings could be used to allow for more flexibility when computing similarity between sentences and could even be used to extend the multi-sentence alignment to include a penalty based on the distance between two words in the embedding

space.

We have shown that LOBBYBACK performs well on reconstructing model legislation from automatically generated bill clusters. However, there are a number of improvements that can refine part of the pipeline. A potential change, but one that is more computationally costly, would be to use a deeper parsing of the sentences that we extract from the documents. We used a simple unigram model when computing sentence similarities because we wanted to ensure that stop words were included—due to their importance in legal text. We suspect that by using a parser we could weight the similarity of noun-phrases for instance, yielding a better similarity matrix and potentially higher precision/recall.

## 6.7 Previous Work

While no specific system or technique has focused on the problem of legislative document reconstruction, we find related work in a number of domains. Multi-document summarization (MDS), for example, can be used to partially model the underlying problem—generating a representative document from multiple sources. Extractive MDS, in particular, is promising in that representative sentences are identified.

Early work in extractive summarization include greedy approaches such as that proposed by Carbonell *et al.* The algorithm uses an objective function which trades off between relevance and redundancy [24]. Global optimization techniques attempt to generate “summaries” (selected sets of sentences or utterances) that maximize an objective based on informativeness, redundancy and/or length of summary. These have shown superior performance to greedy algorithms [57, 177]. Approaches based on neural networks have recently been proposed for ranking candidate sentences [23]. Graph based methods, such as LexRank [42], have also proven effective for MDS. Extensions to this approach combine sentence ranking with clustering in order to minimize redundancy [22, 136, 162]. The C-LexRank algorithm [136], in particular, uses this combination and inspired our high level design.

Though related, it is important to note that the objectives of summarization (informativeness, reduced redundancy, etc.) are not entirely consistent with our task. For example, using the ngram co-occurrence based ROUGE score would not be sufficient at evaluating LOBBYBACK. Our goal is to accurately reconstruct entire sentences of a hidden document given observed mutations of that document. Additionally, our goal is not simply to find a representative sentence that may reflect the original document, but to capture the similarity *and* variability of the text within a given “sentence cluster.”

Within the political science and legal studies communities research has focused on *manual* approaches to both understanding how model legislation impacts law and how policy ideas diffuse between bill text. As these studies are time consuming, there is no large-scale or broad analysis of legislative materials. Rather, researchers have limited their workload by focusing on a single topic (e.g. abortion [126] and crime [88]) or a single lobbying group (e.g. ALEC [80]). Similarly, those studying policy diffusion across US states have also limited their analysis to a few topics (e.g., same-sex marriage [63]).

Recent attempts to automate the analysis of model legislation has had similar problems, as most researchers have limited their analysis to one interest group or a few relevant topics [72, 81]. Hertel-Fernandez *et al.* proposed a supervised model in which they train on hand labeled examples of state bills that borrow text and/or concepts from ALEC bills. The problem they focus on is different from ours, the motivation behind LOBBYBACK is that there exists lots of model legislation which we don't have access to and the goal is to try and reconstruct these documents without labeled training data. Jansa *et al.* propose a technique for inferring a network of policy diffusion for manually labeled clusters of bills. Both Jansa *et al.* and Hertel-Fernandez *et al.* propose techniques that only look at the problem of inferring whether two bills exhibit text reuse but unlike LOBBYBACK they do not attempt to infer whether specific policies (sentences) in the documents are similar/different.

## 6.8 Conclusion

In this chapter we present LOBBYBACK, a system to reconstruct the “dark corpora” that is comprised of model bills which are copied (and modified) by resource constrained state legislatures. A critical component of LOBBYBACK is the inference of a state bill network in which the link structure reflects text reuse between bills. A community detection algorithm is then applied to this network in order to extract clusters of text reuse in a large state legislation corpus. LOBBYBACK then generates prototype sentences that summarize the similarity and variation of the copied text for each bill cluster.



# Chapter 7

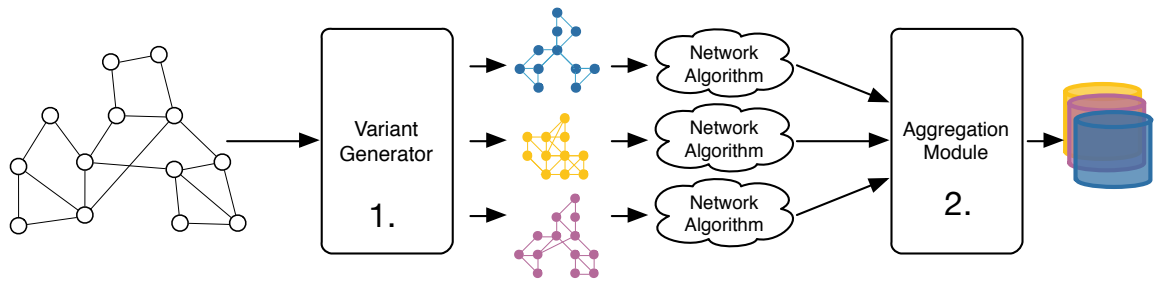
## Conclusion

This dissertation describes the challenges associated with applying social network analysis on incomplete network structures. Incomplete networks can be described as networks that have any combination of the following scenarios: *missing edges*, *missing semantics*, and *missing structure*. Each of the four systems we propose, tackles a problem in which a major component provides a solution to applying network analysis to an incomplete network that falls into one or more of these scenarios. In addition, Each of these four systems provides a novel solution that solves a real world data science problem in disciplines ranging from information retrieval to computational social science.

### 7.1 Contributions and Future Directions

**EDGEBOOST**, proposed in Chapter 3 is a meta-algorithm and framework for improving community detection on both networks that exhibit *missing edges* and *missing structure*. For networks with *missing edges*, **EDGEBOOST** applies a non-deterministic link prediction process to create many imputed variants of an input network. Each of the variants are then clustered with a user defined community detection algorithm producing partitions which are aggregated together in order to form a final higher quality partition. We also extended **EDGEBOOST** to handle networks with *missing structure* by replacing the link prediction module with a threshold enumeration module. The threshold enumeration module takes as input a similarity matrix and prunes the matrix at each of the desired threshold values. The resulting networks are then processed the same way as in the standard **EDGEBOOST** pipeline.

We have shown the efficacy of applying the **EDGEBOOST** framework to the task of community detection and believe that it can also be extended to boost the performance of other network algorithms. The framework that we propose in **EDGEBOOST** can be thought of in a more general sense as pictured in Figure 7.1. This general framework consists of three steps:



**Figure 7.1** Extension of EDGEBOOST framework to other types of network algorithms

1. Generation of a set of *variant* networks based on the input
2. Apply network algorithm to each variant
3. Aggregate output from each variant via an aggregation function

Any instantiation of this framework involves the selection of variant generator and aggregation function. We have proposed two variant generators: link-prediction based sampling, and threshold enumeration that can be re-used in other instantiations. The aggregation algorithm we proposed is specific to community detection but for other types of network algorithms such as centrality computation, an aggregation function as simple as averaging the scores of the nodes may be sufficient.

**BUTTERWORTH** addresses the problem of extracting topically cohesive clusters from an online social network in Chapter 4. The first component of **BUTTERWORTH** is to extract a user’s topic(s) of interest by clustering her ego network. **BUTTERWORTH** uses link prediction to both impute *missing edges* and *missing semantics* in the ego-network. We use community detection to extract topically cohesive clusters from the imputed network. Once extracted, these clusters are used to automatically train ranking models that rank a user’s social feed by the corresponding topic. **BUTTERWORTH** presents an end-to-end solution to the problem of ranking social feeds by a user’s topic(s) of interest, all without requiring any user supervision.

While presented as a component of **BUTTERWORTH**, the use of link prediction–based on the textual content of nodes–to improve down stream network analysis (e.g community detection) can be used in many other applications. Often times, network structure is sparse and *missing semantics* but many networks have nodes with associated textual content, such as: citation, patent and email networks. Link prediction based on the textual content of nodes can lead to more focused and more accurate analysis.

**DOBBY**, presented in Chapter 5 is a system for constructing a knowledge graph of user-defined keyword tags. By computing features based on statistics such as tag co-occurrence

and wikipedia topic information, and training on a labeled set of subsumption edges. DOBBY is able to infer novel hypernym relationships between tags. We used DOBBY to construct a knowledge graph of all skills on the most popular professional social network, LinkedIn. The resulting knowledge graph has many potential applications such as improving ad targeting and query expansion in search applications. Through DOBBY we have shown the efficacy of using a supervised model to predict links in a network, based on features computed from heterogenous data sources describing the nodes in various contexts.

**LOBBYBACK**, presented in Chapter 6, is a system for extracting and automatically summarizing the phenomena of text reuse in state legislation. One critical component of LOBBYBACK is the inference of a network in which the structure represents text reuse between state bills. Community detection is then applied to this network in order to obtain clusters of bills that exhibit text reuse. LOBBYBACK then uses the clusters to construct “prototype” documents that represent the conical representation of the text shared between the documents. We applied LOBBYBACK to the task of reconstructing model legislation written by lobbyists that is known to have influenced clusters of documents in the corpus. One of the main contributions of LOBBYBACK is a novel application of network inference, in which we show the efficacy of using community detection on a dataset which does not have an inherent network structure.

The thesis of this dissertation is that the incorporation of imputation and inference components can improve network analysis pipelines. In proposing each of the four systems: EDGEBOOST, BUTTERWORTH, DOBBY, and LOBBYBACK we demonstrate that by integrating imputation and inference into network analysis pipelines, we can improve the efficacy of the pipelines or enable the application of network analysis to solve new problems. Each of these systems targets specific technical challenges but the imputation and inference techniques that are used can be extended and re-purposed for use in other pipelines to solve new problems.

This dissertation stresses the importance of the network analysis pipeline, instead of just isolating the network algorithm as the only way to improve network analysis. The systems in this dissertation use existing network algorithms, but, by incorporating imputation and inference into the pipelines we were able to improve the quality of the overall analysis. In proposing better pipelines, we have shown that structure of the network a given algorithm is can have as much of an effect as to the choice of the algorithm itself. With systems like BUTTERWORTH, we have also shown that by imputing missing information, network algorithms like community detection can be adapted to new problems such as topic extraction.

### **7.1.1 Summary**

This dissertation has explored the use of network analysis on incomplete structures. We have proposed a comprehensive way of characterizing the scenarios in which networks can be incomplete. Each of the four novel systems, proposed in this dissertation, solve a real data science challenge that involves a network that is incomplete via one of many of our proposed scenarios. In solving these four challenges we have shown that the imputation of missing information in incomplete networks is a crucial component for network analysis on real world data.

# Bibliography

- [1] A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar. 1953.
- [2] M. K. 0002 and J. Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *SDM*, pages 47–58. SIAM / Omnipress, 2011.
- [3] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211 – 230, 2003.
- [4] L. A. Adamic and N. Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery*, LinkKDD '05, pages 36–43, New York, NY, USA, 2005. ACM.
- [5] R. Aldecoa and I. Marin. Exploring the limits of community detection strategies in complex networks. *Sci. Rep.*, 2013.
- [6] R. Aldecoa and I. Marn. Deciphering network community structure by surprise. *PLoS ONE*, 6(9):e24195, 09 2011.
- [7] A. Arenas, A. Fernandez, and S. Gomez. Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics*, 2008.
- [8] M. Bastian, M. Hayes, W. Vaughan, S. Shah, P. Skomoroch, H. Kim, S. Uryasev, and C. Lloyd. LinkedIn skills: Large-scale topic extraction and inference. RecSys '14, 2014.
- [9] M. S. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, and E. H. Chi. Eddi: interactive topic-based browsing of social status streams. In *UIST'10*, pages 303–312, 2010.
- [10] F. S. Berry and W. D. Berry. State lottery adoptions as policy innovations: An event history analysis. *The American Political Science Review*, pages 395–415, 1990.
- [11] P. Bhattacharyya, A. Garg, and S. Wu. Analysis of user keyword similarity in online social networks. *Social Network Analysis and Mining*, 1(3):143–158, 2011.
- [12] P. Bhattacharyya, A. Garg, and S. F. Wu. Social network model based on keyword categorization. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining*, ASONAM '09, pages 170–175, Washington, DC, USA, 2009. IEEE Computer Society.

- [13] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.
- [14] L. M. S. A. V. B. Borate BR, Chesler EJ. Comparison of threshold selection methods for microarray gene co-expression matrices. *BMC Res Notes*, 2009.
- [15] S. P. Borgatti, K. M. Carley, and D. Krackhardt. On the robustness of centrality measures under conditions of imperfect data. *Social Networks*, 28(2):124 – 136, 2006.
- [16] J. Boyd-Graber, J. Chang, S. Gerrish, C. Wang, and D. Blei. Reading tea leaves: How humans interpret topic models. In *NIPS'09*, 2009.
- [17] J. Bragg, Mausam, and D. S. Weld. Crowdsourcing multi-label classification for taxonomy creation. In B. Hartman and E. Horvitz, editors, *HCOMP*. AAAI, 2013.
- [18] M. Burgess, E. Adar, and M. Cafarella. Link-prediction enhanced consensus clustering for complex networks. *PLoS ONE*, 2016.
- [19] M. Burgess, E. Giraudy, and E. Adar. Reconstructing hidden documents from observed text. Under Review.
- [20] M. Burgess, A. Mazzia, E. Adar, and M. Cafarella. Leveraging noisy lists for social feed ranking. International AAAI Conference on Weblogs and Social Media, 2013.
- [21] M. Burgess, P. Skomoroch, and E. Adar. Building a knowledge graph of tags for expertise driven web applications. Under Review.
- [22] X. Cai and W. Li. Ranking through clustering: An integrated approach to multi-document summarization. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(7):1424–1433, July 2013.
- [23] Z. Cao, F. Wei, L. Dong, S. Li, and M. Zhou. Ranking with recursive neural networks and its application to multi-document summarization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2153–2159, 2015.
- [24] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for re-ordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '98*, pages 335–336, New York, NY, USA, 1998. ACM.
- [25] L. D. Caro, K. S. Candan, and M. L. Sapino. Using tagflake for condensing navigable tag hierarchies from tag clouds. In Y. Li, B. Liu, and S. Sarawagi, editors, *KDD*, pages 1069–1072. ACM, 2008.

- [26] M. Chen, A. Bahulkar, K. Kuzmin, and B. K. Szymanski. *Complex Networks VII: Proceedings of the 7th Workshop on Complex Networks CompleNet 2016*, chapter Improving Network Community Structure with Link Prediction Ranking, pages 145–158. Springer International Publishing, Cham, 2016.
- [27] S. S. Choi, S. H. Cha, and C. Tappert. A Survey of Binary Similarity and Distance Measures. *Journal on Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.
- [28] P. Choudhary and U. Singh. Article: A survey on social network analysis for counter-terrorism. *International Journal of Computer Applications*, 112(9):24–29, February 2015. Full text available.
- [29] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, Dec 2004.
- [30] M. D. Conover, J. Ratkiewicz, M. Francisco, B. Goncalves, F. Menczer, and A. Flammini. Political polarization on twitter. *ICWSM*, 2011.
- [31] R. Dahimene, C. Mouza, and M. Scholl. Efficient filtering in micro-blogging systems: We won’t get flooded again. In A. Ailamaki and S. Bowers, editors, *Scientific and Statistical Database Management*, volume 7338 of *Lecture Notes in Computer Science*, pages 168–176. Springer Berlin Heidelberg, 2012.
- [32] J. Dahlin and P. Svenson. Ensemble approaches for improving community detection methods. *ArXiv e-prints*, Sept. 2013.
- [33] W. Dakka and P. G. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE ’08*, pages 466–475, Washington, DC, USA, 2008. IEEE Computer Society.
- [34] D. Darmon, E. Omodei, and J. Garland. Followers are not enough: A multifaceted approach to community detection in online social networks. *PLoS ONE*, 10(8):1–20, 08 2015.
- [35] A. Das Sarma, A. Das Sarma, S. Gollapudi, and R. Panigrahy. Ranking mechanisms in Twitter-like forums. In *WSDM’10*, pages 21–30, 2010.
- [36] E. David and K. Jon. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010.
- [37] P. De Meo, G. Quattrone, and D. Ursino. A query expansion and user profile enrichment approach to improve the performance of recommender systems operating on a folksonomy. *User Modeling and User-Adapted Interaction*, 20(1):41–86, Feb. 2010.
- [38] J.-C. Delvenne, S. N. Yaliraki, and M. Barahona. Stability of graph communities across time scales. *Proceedings of the National Academy of Sciences*, 2010.

- [39] S. Dudoit and J. Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 2003.
- [40] Elastic. Elasticsearch reference, 2016.
- [41] G. Erkan and D. R. Radev. Lexrank: Graph-based centrality as salience in text summarization. *Journal of Artificial Intelligence Research (JAIR)*, 2004.
- [42] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, pages 457–479, 2004.
- [43] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pages 226–231, 1996.
- [44] L. Fang, A. Fabrikant, and K. LeFevre. Look who I found: Understanding the effects of sharing curated friend groups. In *WebSci'12*, pages 137–146, 2012.
- [45] X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, 2004.
- [46] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.
- [47] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.
- [48] B. Fortuna, M. Grobelnik, and D. Mladenic. Semi-automatic Data-driven Ontology Construction System. In *Proceedings of the 9th international multi-conference information society IS-2006*, Ljubljana, Slovenia.
- [49] S. Fortunato and M. Barthlemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 2007.
- [50] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):355–369, March 2007.
- [51] T. L. Frantz, M. Cataldo, and K. M. Carley. Robustness of centrality measures under uncertainty: Examining the role of network topology. *Computational and Mathematical Organization Theory*, 15(4):303–328, 2009.
- [52] A. Garg, P. Bhattacharyya, C. U. Martel, and S. F. Wu. Information flow and search in unstructured keyword based social networks. In *CSE (4)*, pages 1074–1081. IEEE Computer Society, 2009.



- [53] K. N. Garrett and J. M. Jansa. Interest group influence in policy diffusion networks. *State Politics & Policy Quarterly*, 15(3):387–417, 2015.
- [54] S. Gauch, J. Chaffee, and A. Pretschner. Ontology-based personalized search and browsing. *Web Intelli. and Agent Sys.*, 1(3-4):219–234, Dec. 2003.
- [55] D. Gfeller, J.-C. Chappelier, and P. De Los Rios. Finding instabilities in the community structure of complex networks. *Phys. Rev. E*, 2005.
- [56] R. Ghaemi, M. N. Sulaiman, H. Ibrahim, and N. Mustapha. A survey: Clustering ensembles techniques, 2009.
- [57] D. Gillick, K. Riedhammer, B. Favre, and D. Hakkani-Tur. A global optimization framework for meeting summarization. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '09*, pages 4769–4772, Washington, DC, USA, 2009. IEEE Computer Society.
- [58] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.
- [59] W. H. Goma and A. A. Fahmy. Article: A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, April 2013. Full text available.
- [60] V. Gray. Innovation in the states: A diffusion study. *American political science review*, 67(04):1174–1185, 1973.
- [61] B. Guc. Information Filtering on Micro-blogging Services. In *Master's Thesis*. Swiss Federal Institute of Technology Zürich, 2010.
- [62] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.
- [63] D. P. Haider-Markel. Policy diffusion as a geographical expansion of the scope of political conflict: Same-sex marriage bans in the 1990s. *State Politics & Policy Quarterly*, 1(1):5–26, 2001.
- [64] S.-K. Han. The other ride of paul revere: The brokerage role in the making of the american revolution. *Mobilization: An International Quarterly*, 14(2):143–162, 2009.
- [65] J. A. Hanley and B. J. Mcneil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.
- [66] S. Harispe, S. Ranwez, S. Janaqi, and J. Montmain. Semantic measures for the comparison of units of language, concepts or entities from text and knowledge base analysis. *CoRR*, 2013.
- [67] M. Hasan and M. Zaki. A survey of link prediction in social networks. In C. C. Aggarwal, editor, *Social Network Data Analytics*, pages 243–275. Springer US, 2011.

- [68] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. In *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006.
- [69] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web, WWW '02*, pages 517–526, New York, NY, USA, 2002. ACM.
- [70] K. Healy, 2013.
- [71] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004.
- [72] A. Hertel-Fernandez and K. Kashin. Capturing business power across the states with text reuse. In *annual conference of the Midwest Political Science Association, Chicago, April*, pages 16–19, 2015.
- [73] P. Heymann and H. Garcia-Molina. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical Report 2006-10, Computer Science Department, April 2006.
- [74] L. Hong, R. Bekkerman, J. Adler, and B. Davison. Learning to rank social update streams. In *SIGIR'12*, Portland, Oregon, 2012.
- [75] L. Hong, G. Convertino, B. Suh, E. H. Chi, and S. Kairam. FeedWinnower: layering structures over collections of information streams. In *CHI '10*, pages 947–950, 2010.
- [76] L. Hong and B. Davison. Empirical study of topic modeling in Twitter. In *Proceedings of the First Workshop on Social Media Analytics*, pages 80–88. ACM, 2010.
- [77] Y. Hu, Y. Nie, H. Yang, J. Cheng, Y. Fan, and Z. Di. Measuring the significance of community structure in complex networks. *Physical Review E*, 2010.
- [78] H. Huang, B. Jedynek, and J. S. Bader. Where have all the interactions gone? estimating the coverage of two-hybrid protein interaction maps. *PLoS Computational Biology*, 3(11), 2007.
- [79] P. Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.
- [80] M. Jackman. Alecs influence over lawmaking in state legislatures, December 6 2013.
- [81] J. M. Jansa, E. R. Hansen, and V. H. Gray. Copy and paste lawmaking: The diffusion of policy language across american state legislatures. 2015.
- [82] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '02*, pages 538–543, New York, NY, USA, 2002. ACM.

- [83] C. Jha, Rahul Finegan-Dollak, B. King, R. Coke, and D. R. Radev. Content models for survey generation: A factoid-based evaluation. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics*, 2015.
- [84] R. Jha, A. Abu-Jbara, and D. R. Radev. A system for summarizing scientific topics starting from keywords. In *Proceedings of The Association for Computational Linguistics (short paper)*, 2013.
- [85] R. Jha, R. Coke, and D. R. Radev. Surveyor: A system for generating coherent survey articles for scientific topics. In *Proceedings of the Twenty-Ninth AAAI Conference*, 2015.
- [86] P. F. Jonsson, T. Cavanna, D. Zicha, and P. A. Bates. Cluster analysis of networks generated through homology: automatic identification of important protein communities involved in cancer metastasis. *BMC bioinformatics*, 7, 2006.
- [87] B. Karrer, E. Levina, and M. E. J. Newman. Robustness of community structure in networks. *Phys. Rev. E*, 2008.
- [88] S. L. Kent and J. T. Carmichael. Legislative responses to wrongful conviction: Do partisan principals and advocacy efforts influence state-level criminal justice policy? *Social science research*, 52:147–160, 2015.
- [89] D. Kim, Y. Jo, I.-C. Moon, and A. Oh. Analysis of Twitter lists as a potential source for discovering latent characteristics of users. In *Workshop on Microblogging at the ACM Conference on Human Factors in Computer Systems. (CHI 2010)*, 2010.
- [90] J. Klinginsmith, M. Mahoui, Y. Wu, and J. F. Jones. Discovering domain specific concepts within user-generated taxonomies. In Y. Saygin, J. X. Yu, H. Kargupta, W. W. 0010, S. Ranka, P. S. Yu, and X. Wu, editors, *ICDM Workshops*, pages 19–24. IEEE Computer Society, 2009.
- [91] G. Kossinets. Effects of missing data in social networks. *Social Networks*, 28(3):247 – 268, 2006.
- [92] Z. Kozareva and E. Hovy. A semi-supervised method to learn and construct taxonomies using the web. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP '10*, pages 1110–1118, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
- [93] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis, 2009.
- [94] A. Lancichinetti and S. Fortunato. Consensus clustering in complex networks. *Scientific Reports*, 2012.
- [95] A. Lancichinetti, S. Fortunato, and J. Kertsz. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

- [96] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 2008.
- [97] A. Lancichinetti, F. Radicchi, and J. J. Ramasco. Statistical significance of communities in networks. *Physical Review E* 81, 046110, 2010.
- [98] E. Leicht, P. Holme, and M. Newman. Vertex similarity in networks. *Phys. Rev. E*, 73:026120, Feb 2006.
- [99] Z. Li, S. Zhang, R.-S. Wang, X.-S. Zhang, and L. Chen. Quantitative function for community detection. *Phys. Rev. E*, 2008.
- [100] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 2007.
- [101] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10*, pages 243–252, New York, NY, USA, 2010. ACM.
- [102] H. Lin, J. Davis, and Y. Zhou. An integrated approach to extracting ontological structures from folksonomies. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications, ESWC 2009 Heraklion*, pages 654–668, Berlin, Heidelberg, 2009. Springer-Verlag.
- [103] J. Lin, R. Snow, and W. Morgan. Smoothing techniques for adaptive online language models: topic tracking in tweet streams. In *KDD'11*, pages 422–429. ACM, 2011.
- [104] W. Lin, X. Kong, P. S. Yu, Q. Wu, Y. Jia, and C. Li. Community detection in incomplete information networks. In *Proceedings of the 21st International Conference on World Wide Web, WWW '12*, pages 341–350, New York, NY, USA, 2012. ACM.
- [105] L. Lü, C.-H. Jin, and T. Zhou. Similarity index based on local paths for link prediction of complex networks. *Phys. Rev. E*, 2009.
- [106] L. Lu and T. Zhou. Link prediction in complex networks: A survey. *Physica A*, 2011.
- [107] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
- [108] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *NIPS*, 2012.
- [109] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.
- [110] A. Mirshahvalad, O. H. Beauchesne, E. Archambault, and M. Rosvall. Mapping change in large networks. *PLoS ONE*, 2010.

- [111] A. Mirshahvalad, J. Lindholm, M. Derln, and M. Rosvall. Significant communities in large sparse networks. *PLoS ONE*, 2012.
- [112] S. Mohammad, B. Dorr, M. Egan, A. Hassan, P. Muthukrishan, V. Qazvinian, D. Radev, and D. Zajic. Using citations to generate surveys of scientific paradigms. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 584–592, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
- [113] S. Monti, P. Tamayo, J. Mesirov, and T. Golub. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Mach. Learn.*, 52(1-2):91–118, July 2003.
- [114] F. Morstatter, Jürgen Pfeffer, H. Liu, and K. M. Carley. Is the sample good enough? comparing data from twitter’s streaming api with twitter’s firehose. 2013.
- [115] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, March 1957.
- [116] V. Nair and S. Dua. Folksonomy-based ad hoc community detection in online social networks. *Social Netw. Analys. Mining*, 2(4):305–328, 2012.
- [117] A. Nenkova and R. Passonneau. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (HLT-NAACL '04)*, 2004.
- [118] M. Newman. Political books network.
- [119] M. Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.
- [120] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.
- [121] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.
- [122] G. K. Orman and V. Labatut. A comparison of community detection algorithms on artificial networks. In *Discovery Science*, pages 242–256. Springer Berlin Heidelberg, 2009.
- [123] T. Paek, M. Gamon, S. Counts, D. M. Chickering, and A. Dhesi. Predicting the importance of newsfeed posts and social network friends. In *AAAI'10*, pages 1419–1424, 2010.
- [124] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1999.

- [125] A. Pal and S. Counts. Identifying topical authorities in microblogs. In *WSDM'11*, pages 45–54, 2011.
- [126] D. J. Patton. The effect of united states supreme court intervention on the innovation and diffusion of post-roe abortion policies in the american states-university of kentucky. *National Catholic Bioethics Quarterly*, 2004.
- [127] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.
- [128] L. M. Perkins AD. Threshold selection in gene co-expression networks using spectral graph theory techniques. *BMC Bioinformatics*, 2009.
- [129] A. Plangprasopchok, K. Lerman, and L. Getoor. Growing a tree in the forest: Constructing folksonomies by integrating structured metadata. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10*, pages 949–958, New York, NY, USA, 2010. ACM.
- [130] J. Platig, E. Ott, and M. Girvan. Robustness of network measures to link errors. *Phys. Rev. E*, 88:062812, Dec 2013.
- [131] P. Pons and M. Latapy. Computing communities in large networks using random walks. In *Proceedings of the 20th International Conference on Computer and Information Sciences, ISCIS'05*, 2005.
- [132] P. Pons and M. Latapy. Computing communities in large networks using random walks. In *Computer and Information Sciences - ISCIS 2005*, volume 3733 of *Lecture Notes in Computer Science*. 2005.
- [133] J. Pound, S. Paparizos, and P. Tsaparas. Facet discovery for structured web search: A query-log mining approach. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11*, pages 169–180, New York, NY, USA, 2011. ACM.
- [134] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd International Conference on World Wide Web, WWW '14*. ACM, 2014.
- [135] V. Qazvinian and D. R. Radev. Scientific paper summarization using citation summary networks. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING-08)*, Manchester, UK, 2008.
- [136] V. Qazvinian and D. R. Radev. Scientific paper summarization using citation summary networks. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 689–696, Manchester, UK, August 2008. Coling 2008 Organizing Committee.

- [137] V. Qazvinian, D. R. Radev, S. M. Mohammad, B. Dorr, D. Zajic, M. Whidby, and T. Moon. Generating extractive summaries of scientific paradigms. *J. Artif. Int. Res.*, 46(1):165–201, Jan. 2013.
- [138] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3), 2007.
- [139] D. Ramage, S. Dumais, and D. Liebling. Characterizing microblogs with topic models. In *WDSM'10*, 2010.
- [140] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A. L. Barabási. Hierarchical organization of modularity in metabolic networks. *Science (New York, N.Y.)*, 297(5586):1551–1555, Aug. 2002.
- [141] P. Ronhovde and Z. Nussinov. Local resolution-limit-free potts model for community detection. *Phys. Rev. E*, 81:046114, Apr 2010.
- [142] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 2008.
- [143] E. Sadikov, M. Medina, J. Leskovec, and H. Garcia-Molina. Correcting for missing data in information cascades. In *Proceedings of the fourth ACM international conference on Web search and data mining, WSDM '11*, pages 55–64, New York, NY, USA, Feb. 2011. ACM Press.
- [144] S. Salem, S. Banitaan, I. Aljarah, J. E. Brewer, and R. Alroobi. Discovering communities in social networks using topology and attributes. In X. wen Chen, T. S. Dillon, H. Ishbuchi, J. Pei, H. Wang, and M. A. Wani, editors, *ICMLA (1)*, pages 40–43. IEEE Computer Society, 2011.
- [145] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.
- [146] M. Sanderson and B. Croft. *Deriving concept hierarchies from text*. 1999.
- [147] P. Sarkar, D. Chakrabarti, and A. W. Moore. Theoretical justification of popular link prediction heuristics. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three, IJCAI'11*, pages 2722–2727. AAAI Press, 2011.
- [148] E. Schwarzkopf, D. Heckmann, D. Dengler, and A. Kröner. Mining the structure of tag spaces for user modeling. In R. S. Baker, J. E. Beck, B. Berendt, A. Kröner, E. Menasalvas, and S. Weibelzahl, editors, *Proceedings of the UM 2007 workshop on Data Mining for User Modeling (DMUM07)*, pages 30–31, Corfu, Greece, June 2007. Extended abstract.
- [149] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, Jan. 2001.

- [150] C. R. Shipan and C. Volden. The mechanisms of policy diffusion. *American journal of political science*, 52(4):840–857, 2008.
- [151] S. Sina, A. Rosenfeld, and S. Kraus. Solving the missing node problem using structure and attribute information. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 744–751, Aug 2013.
- [152] G. Solskinnsbakk and J. A. Gulla. A hybrid approach to constructing tag hierarchies. In *Proceedings of the 2010 International Conference on On the Move to Meaningful Internet Systems: Part II, OTM’10*, pages 975–982, Berlin, Heidelberg, 2010. Springer-Verlag.
- [153] W.-M. Song and B. Zhang. Multiscale embedded gene co-expression network analysis. *PLoS Comput Biol*, 11(11):1–35, 11 2015.
- [154] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biol. Skr.*, 5:1–34, 1948.
- [155] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas. Short text classification in Twitter to improve information filtering. In *SIGIR ’10*, pages 841–842, 2010.
- [156] A. Strehl and J. Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, Mar. 2003.
- [157] V. A. Traag, R. Aldecoa, and J.-C. Delvenne. Detecting communities using asymptotical surprise. *Phys. Rev. E*, 92:022816, Aug 2015.
- [158] V. A. Traag, G. Krings, and P. Van Dooren. Significant Scales in Community Structure. *Scientific Reports*, 2013.
- [159] C.-Y. Tseng, Y.-J. Chen, and M.-S. Chen. Socfeedviewer: A novel visualization technique for social news feeds summarization on social network services. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 616 –617, june 2012.
- [160] I. Uysal and W. B. Croft. User oriented tweet ranking: a filtering approach to microblogs. In *CIKM ’11*, pages 2261–2264, 2011.
- [161] C. Wagner, V. Liao, P. Pirolli, L. Nelson, and M. Strohmaier. It’s not in their tweets: Modeling topical expertise of Twitter users. *SocialCom ’12*, 2012.
- [162] X. Wan and J. Yang. Multi-document summarization using cluster-based link analysis. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’08*, pages 299–306, New York, NY, USA, 2008. ACM.



- [163] C. Wang, M. Danilevsky, N. Desai, Y. Zhang, P. Nguyen, T. Taula, and J. Han. A phrase mining framework for recursive construction of a topical hierarchy. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 437–445, New York, NY, USA, 2013. ACM.
- [164] C. Wang, V. Satuluri, and S. Parthasarathy. Local probabilistic models for link prediction. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 322–331, Oct 2007.
- [165] D. Wang, D. Pedreschi, C. Song, F. Giannotti, and A.-L. Barabasi. Human mobility, social ties, and link prediction. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1100–1108, New York, NY, USA, 2011. ACM.
- [166] P. Wang, B. Xu, Y. Wu, and X. Zhou. Link Prediction in Social Networks: the State-of-the-Art. *ArXiv e-prints*, Nov. 2014.
- [167] B. Wei, J. Liu, J. Ma, Q. Zheng, W. Zhang, and B. Feng. Dft-extractor: A system to extract domain-specific faceted taxonomies from wikipedia. In *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 277–280, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
- [168] Q. Wei and R. L. Dunbrack, Jr. The role of balanced training and testing data sets for binary classifiers in bioinformatics. *PLoS ONE*, 2013.
- [169] J. Weng, E.-P. Lim, J. Jiang, and Q. He. Twiterrank: Finding topic-sensitive influential twitterers. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 261–270, New York, NY, USA, 2010. ACM.
- [170] S. Wernicke and F. Rasche. Fanmod: A tool for fast network motif detection. *Bioinformatics*, 2006.
- [171] W. Wong, W. Liu, and M. Bennamoun. Ontology learning from text: A look back and into the future. *ACM Comput. Surv.*, 44(4):20:1–20:36, Sept. 2012.
- [172] J. Xiang, X. Hu, X. Zhang, J. Fan, X. Zeng, G. Fu, K. Deng, and K. Hu. Multi-resolution modularity methods and their limitations in community detection. *The European Physical Journal B*, 2012.
- [173] Z. Xu, R. Lu, L. Xiang, and Q. Yang. Discovering user interest on Twitter with a modified author-topic model. In *WI-IAT'11*, volume 1, pages 422–429, aug. 2011.
- [174] B. Yan and S. Gregory. Finding missing edges and communities in incomplete networks. Sept. 2011.
- [175] B. Yan and S. Gregory. Detecting community structure in networks using edge prediction methods. *CoRR*, abs/1201.3466, 2012.

- [176] Y. Yang, N. Chawla, P. Basu, B. Prabhala, and T. La Porta. Link prediction in human mobility networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 380–387, Aug 2013.
- [177] W.-T. Yih, J. Goodman, L. Vanderwende, and H. Suzuki. Multi-Document Summarization by Maximizing Informative Content-Words. 2007.
- [178] J. Yu, P. Tadepalli, and L. Getoor. Chance-constrained programs for link prediction.
- [179] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 1977.
- [180] E. Zavitsanos, G. Paliouras, G. A. Vouros, and S. Petridis. Discovering subsumption hierarchies of ontology concepts from text corpora. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, WI '07*, pages 402–408, Washington, DC, USA, 2007. IEEE Computer Society.
- [181] B. Zhang, S. Horvath, B. Zhang, and S. Horvath. A general framework for weighted gene coexpression network analysis. In *Statistical Applications in Genetics and Molecular Biology 4: Article 17*, 2005.
- [182] T. Zhou, L. Lü, and Y. Zhang. Predicting missing links via local information. *The European Physical Journal B-Condensed Matter and Complex Systems*, 71(4):623–630, 2009.