Divided We Tweet: Community Detection in Political Networks

Gene Xijie Li, ’19

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Engineering
Department of Engineering
Princeton University

May 16, 2017

Advisor: Professor Emmanuel Abbe
ELE 298

I hereby declare that this Independent Work report represents my own work in accordance with University Regulations.

Gene Li

Abstract

Community detection in networks has many applications in fields such as sociology, biology, and political science, among others. We apply three community detection algorithms: the spectral algorithm, a semi-definite programming algorithm, and an acyclic belief propagation algorithm on the classic political blog dataset from [1] as well as on a new dataset we created of political journalists active on Twitter in 2017. The Twitter dataset is particularly interesting because of the large amount of "centrist" nodes which connect to both left and right-leaning journalists, as well as the strong presence of a much more densely connected extreme right-leaning journalist group. We analyze the performance of these three community algorithms and compare positives and negatives for each.

Acknowledgements

Thank you to StatSocial, for providing the original Twitter journalists dataset that I used for this research as well as answering my queries about the dataset.

To David Luet and Benjamin Hicks from PICSciE, thank you for helping me set up my Nobel cluster account, showing me how to use `tmux`, and helping me run my Twitter scraper.

To Professor Abbe, thank you for advising me this semester and introducing me to your research interests. I really learned a lot about community detection and your work in the field.

To Emma Corless, thank you for proofing my drafts and providing company while I worked on this project.

Contents

1 Introduction

1.1 Social Media and Politics

Social media is extremely prevalent in American society: 62% of all Americans use Facebook, and 20% use Twitter [2]. Americans, especially young Americans, are increasingly turning towards social media such as Facebook, Twitter, and Reddit for political news. Since the current political environment is especially polarized and antagonistic, in-person conversations about politics can often turn into arguments. However, online social media forces users to be exposed to controversial or inflammatory political statements [2]. Thus, as social media users attempt to curate their content and limit exposure to the statements they view as contentious, an "echo chamber" is produced, where users self-segregate into communities that reinforce their own political belief systems while censoring competing belief systems.

Unlike Facebook, which mostly functions as a medium to stay connected with personal acquaintances, friends, and family, Twitter serves as a broader platform for disseminating news and current events. This is because Twitter connections are one-way - anyone using Twitter can follow Donald Trump, Lebron James, or any other public figure who has made their account public without those public figures giving express consent to be followed. Twitter users report that their network contains more public figures and users that they do not know personally [2].

In the 2016 election, Twitter played an important role in disseminating political opinions, inciting debate, and bringing attention to candidates. Tweets, which are capped at 140 characters, cannot contain in depth information about a candidate's platform, but instead serve as soundbites which attract attention of all Americans. Whereas Hillary Clinton's tweets were more curated and produced by campaign staff, Donald Trump's tweets were more aggressive, freewheeling, and unconventional for a candidate (see Fig.1). Many Americans who voted for Trump cited his free-wheeling Tweets as a reason, as they believed that he was not afraid to speak his mind.

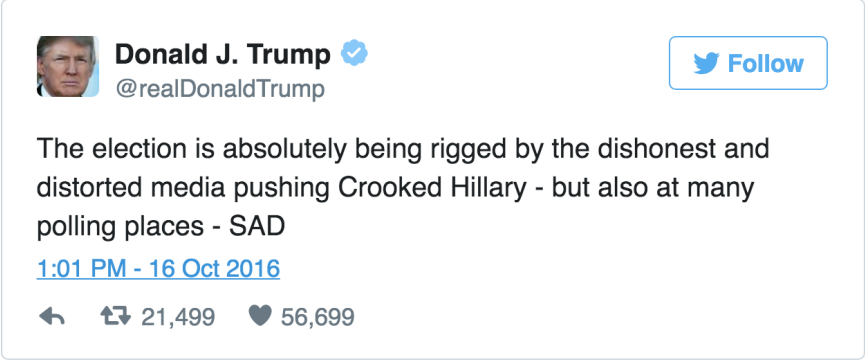


Figure 1: One of Donald Trump's campaign trail tweets

Thus, it is becoming more and more important to understand how political opinions are propagated and amplified by social media, and to understand their effects on the general American electorate.

1.2 Community Detection

Many systems in the real world can be represented as graphs. Generally, a graph $G(V, E)$ is a collection of vertices (nodes) $V = \{v_1, v_2, ...\}$ and edges $E = \{e_1, e_2, ...\}$ that connect vertices in V. For example, in a graph of Hollywood actors, nodes can represent individual actors, and edges can represent friendships. In the field of molecular biology, nodes can represent proteins, and edges represent protein-protein interaction. In social media, nodes can represent individuals, and edges represent followers/friendships. Many more examples exist in sociology, computer science, political science, biology, machine learning, etc. [3].

Graphs representing real systems have both order and disorder. The distribution of edges both globally and locally can be inhomogeneous, specifically groups of nodes may have many edges among them, but few between these groups. This gives rise to a feature of networks called community structure. It is clear that real-world networks can exhibit communities when we consider friendship circles, proteins belonging to a class of molecules, party affiliation, etc.

Thus, community detection, specifically graph clustering, is the identifying of clusters of vertices based on information encoded in the graph only. However, it should be noted that the definition of a "community" is not precise. We can derive different sets of clusters based off of different definitions of a community.

In the context of this report, we focus on communities in political networks, which naturally gives rise to the problem of identifying communities that share similar political beliefs. The aim of this report is to study techniques in community detection applied to different datasets of political social networks.

2 Data and Methods

This section will outline the benchmark blog dataset from [1], a Twitter journalists dataset that we extracted, signal-to-noise-ratio (SNR) calculations on the different datasets, and the various algorithms we implemented.

2.1 Political Blogs Dataset

In 2005, Lada Adamic and Natalie Glance wrote a paper called "The Political Blogosphere and the 2004 U.S. Election: Divided They Blog", where they collected a data set of political weblog URLs from various directories and web-scraped a network of links between blog websites [1]. They manually labeled each weblog as liberal or conservative. This dataset, which is publicly available at http://www-personal.umich.edu/ mejn/netdata/, has been used as a benchmark real world dataset because of the clear separation between liberal and conservative (see Fig. 2).

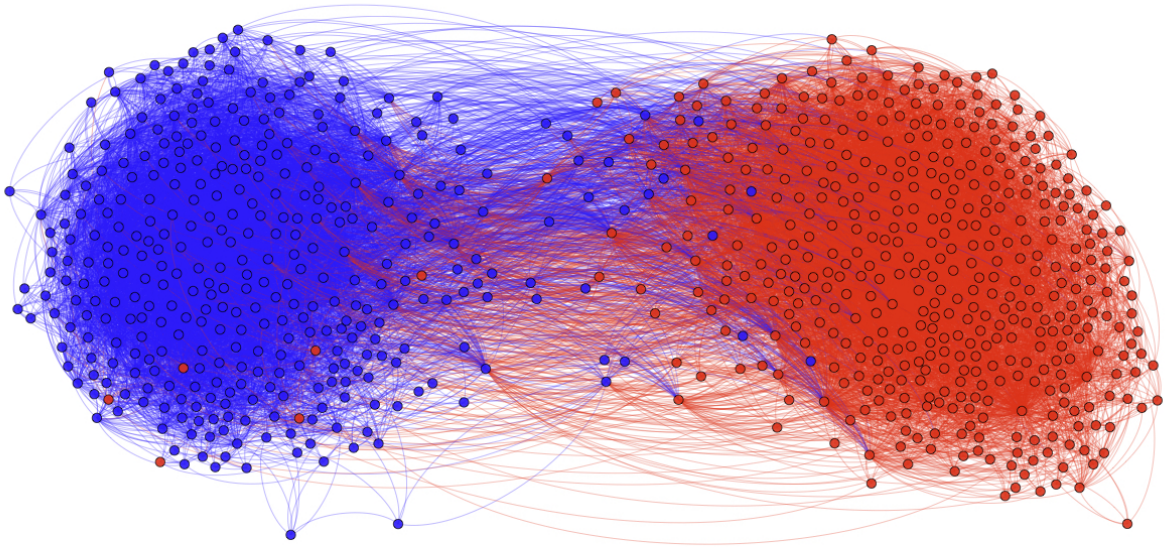


Figure 2: Visualization of Political Blogs Dataset with Gephi. Blue represents liberal, red represents conservative.

The dataset contains 1,494 nodes, of which 759 were liberal and 735 conservative. The authors of [1] were not able to retrieve web pages for some of these blog URLs, and others were not connected. Thus, in this report we analyze the largest connected component of the graph, which contains 1,222 nodes (586 liberal and 636 conservative), with 16,714 edges. 91% of all links between blogs stay within their liberal/conservative community [1].

We refer to [1] for further details about the political blog dataset.

2.2 Political Twitter Journalists Dataset

Using the blogs dataset as inspiration, we wanted to develop a new dataset for the 2016 election that represents the current political atmosphere. While in 2004, weblogs may have been an influential aspect of the election, as 9%

of internet users at that time say they read political blogs “frequently” or “sometimes” [1]; in 2017, many of these websites are now defunct.

We will provide some reasons why we turn to Twitter networks. First let us define some terminology for those unfamiliar with Twitter that will be used throughout this report.

- A tweet is a post on Twitter limited by 140 characters by the original author.
- A follower of X is a person who is able to view the tweets of X.
- A friend of X is someone that X him/herself follows.
- A follower base of X is the set of all followers of X.
- A retweet is when a follower of X forwards X’s tweet to his/her own follower base. Retweets are usually made to bring greater attention to a particular news article, video, or event.
- A public account is an account where anyone else who has a Twitter account can follow. Most celebrities and public figures have public accounts.

We will look at a Twitter political network of public figures for the below reasons:

1. 20% of all Americans use Twitter, and a significant portion of those encounter some sort of political content on Twitter [2].
2. Most political figures, journalists, and bloggers have public accounts, allowing them to reach a massive follower base. Moreover, most political figures, journalists, and bloggers are extremely active on Twitter, sometimes with multiple tweets or retweets in one day.
3. Twitter networks represent a more interesting problem because of the diverse range of perspectives that social media provides. Weblogs from 2004 were usually very partisan and opinion based, catering to a smaller group of readers who themselves were very much affiliated and concerned with politics. However, on Twitter, we observe many nonpartisan journalists who are more concerned with disseminating information, rather than taking and arguing a side. In addition, it is much harder on social media to avoid competing political beliefs, as evidenced by Trump’s astonishing ability to make hundreds of tweets that are seen by millions of liberals and conservatives alike. Moreover, it has been found that most Twitter users who are interested in politics follow others with a wide range of beliefs [2].

The original list of political journalists and bloggers on Twitter comes from StatSocial, a company that uses data analytics to create insights into media influence and branding on social media. On April 16, 2015, they published a

listing of “Twitter’s Most Influential Political Journalists,” which contains 1,963 journalists and bloggers who maintained a strong presence on Twitter. It can be found at https://www.statsocial.com/social-journalists/ [4]. Notably, some of the more popular original bloggers from 2004 have maintained their websites and created Twitter accounts where they tweet out articles from their blogs. To rank these journalists, StatSocial uses a proprietary metric called “social pull”, which measures the “quality and size of the Twitter audience”. For example, a “social pull” of 10x means that the audience of the Twitter account is 10x larger than the audience of an average Twitter account [5]. In addition, the StatSocial dataset provides a “Left/Right” leaning metric that is calculated by subtracting the percentage of followers identifying as Democratic/Green Party by the percentage of followers identifying as Republican/Libertarian on the social web. Thus, this “Left/Right” metric ranges from -100% (completely left-leaning audience) to 100% (completely right-leaning audience). In Fig. 3, we have plotted a histogram of the distribution of “Left/Right” leaning. StatSocial arbitrarily classifies journalists with “Left/Right” leaning falling in range -15% - 15% as “Centrist”.

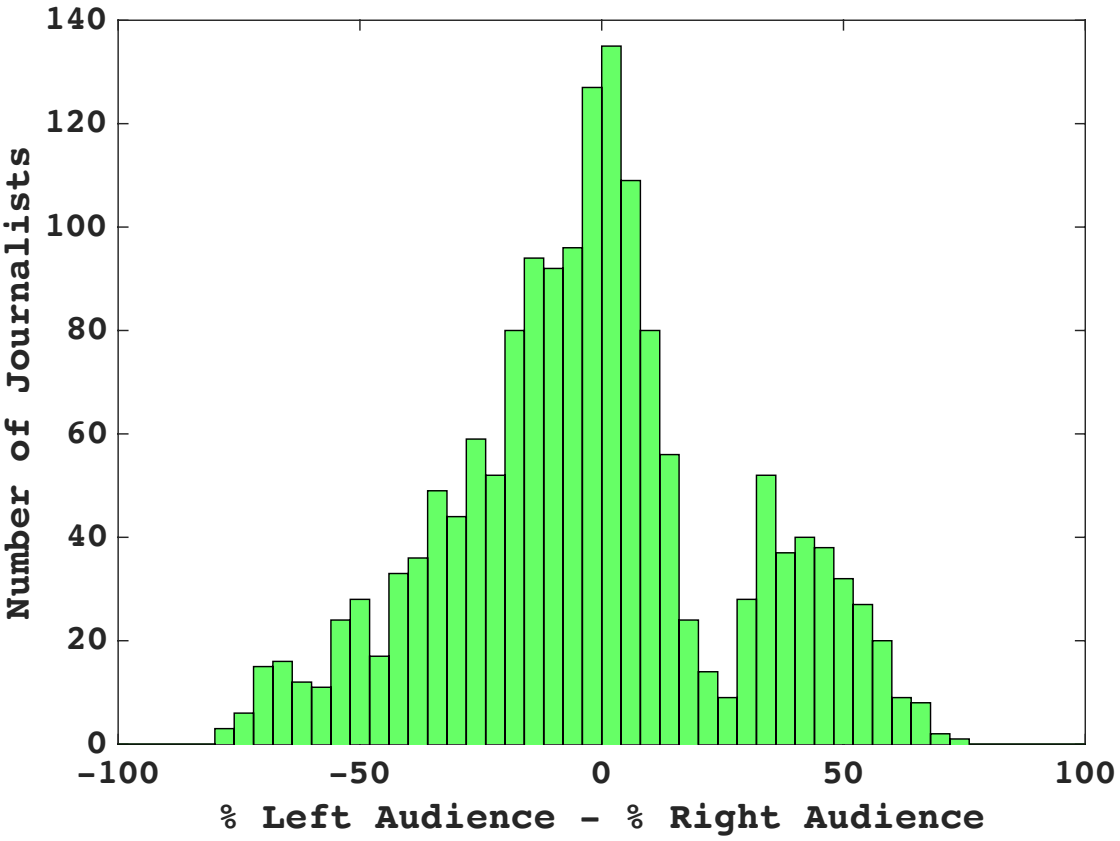


Figure 3: Distribution of Left/Right leaning for Twitter dataset.

Using this list, we build the graph of follower relationships (see Fig. 4). Twitter account information is first updated for the 1,963 journalists using a Python script and the python-twitter library, an easy to use Python wrapper for the Twitter API (found at https://github.com/bear/python-twitter). We manually update our list for several users who had

Rank	Left	Followers	Right	Followers	Centrist	Followers
1	NateSilver538	1035	jaketapper	987	daveweigel	946
2	ezraklein	879	costareports	800	mikeallen	944
3	maddow	674	MajorCBS	618	BuzzFeedBen	937
4	chrislhayes	657	LarrySabato	576	maggieNYT	927
5	ggreenwald	621	HotlineJosh	533	chucktodd	867
6	jdickerson	596	moody	529	jmartNYT	847
7	paulkrugman	594	BretBaeier	522	brianstelter	803
8	DavidCornDC	571	DanaPerino	517	TheFix	802
9	mattyglesias	569	edhenry	506	KFILE	771
10	GStephanopoulos	547	megynkelly	493	GlennThrush	736

Table 1: Most popular Twitter accounts by follower count. "Left", "Right" and "Centrist" classification refers having StatSocial metric of "Left/Right" less than -15, greater than 15, and between -15 and 15.

changed their names since April 16, 2015. In addition, we also deleted some Twitter users who 1) had their account shut down, 2) were unable to be located, 3) made their profile private, thus preventing us from scraping their friend list, 4) were duplicates in the original list, 5) were not journalists with a focus on American news (i.e. international journalists who reported on affairs in other countries). We found that the international journalists mostly formed their own community and were largely classified as left, which was undesirable for our purpose of community detection of American political networks. After this preprocessing, we are left with 1,621 users.

Using this updated list of Twitter journalists, we web-scrape who each journalist follows among the list, thus building up the graph. Even though Twitter relationships are directed (A follows B does not necessarily imply B follows A), we opt to view follower/followee relations as undirected edges. This Twitter graph has 1,615 nodes and 167,806 undirected edges in the largest connected component. Code for scraping user data and the web graph can be found in (5.1).

As seen in Fig. 3, there are a large number of relatively moderate Twitter users, very few extreme left-leaning Twitter users, and a significant amount of extreme right-leaning users. The graph is fairly balanced: if we view all journalists with "Left/Right" metric less than 0 as "Left" and greater than 0 as "Right", there are 894 left-leaning journalists and 721 right-leaning journalists. The most popular "Left", "Right", and "Centrist" Twitter accounts are listed in Table 1.

2.3 Algorithms

To solve community detection for these two datasets, we use three different community detection algorithms, which are outlined below.

First, let us define some notation. Let $G(V, E)$ be graph with vertex set $V = \{v_1, ..., v_n\}$. Let $W \in \mathbb{R}^{n \times n}$ be the adjacency matrix where $(W)_{ij} = 1$ iff there is an edge connecting v_i and v_j, $(W)_{ij} = 0$ otherwise. Since we are

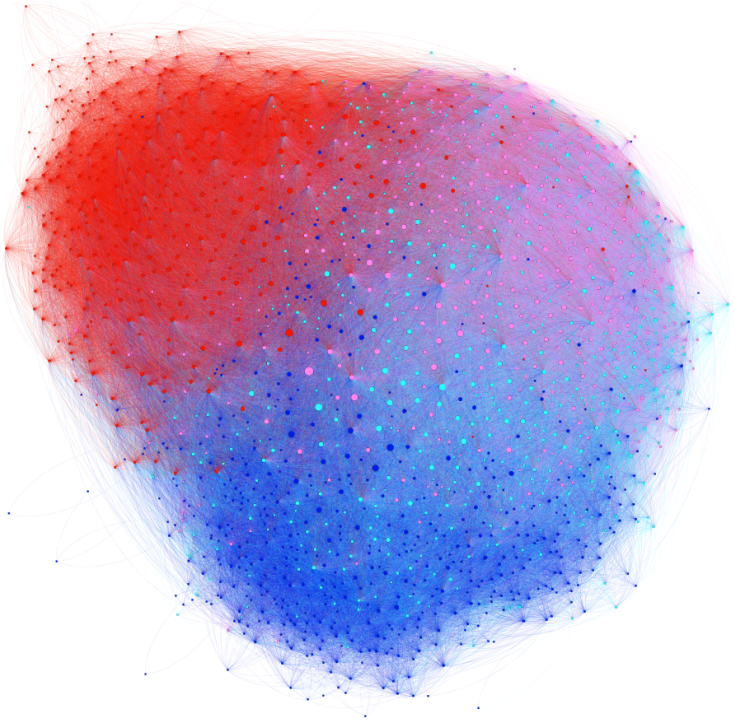


Figure 4: Visualization of Political Journalists Twitter Dataset with Gephi. Blue represents left, red represents right, cyan represents moderate left, and pink represents moderate right.

using an undirected graph, we have the property that W is symmetric. Let us define $D \in \mathbb{R}^{n \times n}$ be the diagonal matrix with values $(D)_{ii} =$ degree of vertex i $\forall i = 1, ..., n$.

2.3.1 Spectral Methods

Spectral clustering methods are one of the most popular clustering algorithms for graph data because of their speed and simplicity. Essentially, spectral clustering uses eigenvalues to reduce the dimension of data of the adjacency matrix. Note that we order eigenvectors by their correspondence to the eigenvalue - thus the statement "first k eigenvectors" refers to the "k eigenvectors with smallest k eigenvalues".

Below we present three spectral clustering algorithms from [6] for completeness.

Algorithm 1 Unnormalized Spectral Clustering

- Compute Laplacian $L = D - W$.
- Compute first k eigenvectors $u_1, ..., u_k$ of L.
- Let $U \in \mathbb{R}^{n \times k}$ contain $u_1, ..., u_k$.
- For $i = 1, ..., n$ let $y_i \in \mathbb{R}^k$ be a vector corresponding to the i-th row of U.
- Cluster (y_i) with k-means algorithm into clusters $C_1, ..., C_k$.

Algorithm 2 Normalized Spectral Clustering due to Shi and Malik (2000)

- Compute Laplacian $L = D - W$.
- Compute first k eigenvectors $u_1, ..., u_k$ of the generalized eigenproblem $Lu = \lambda Du$.
- Let $U \in \mathbb{R}^{n \times k}$ contain $u_1, ..., u_k$.
- For $i = 1, ..., n$ let $y_i \in \mathbb{R}^k$ be a vector corresponding to the i-th row of U.
- Cluster (y_i) with k-means algorithm into clusters $C_1, ..., C_k$.

Algorithm 3 Normalized Spectral Clustering due to Ng, Jordan, and Weiss (2002)

- Compute Laplacian $L_{\text{sym}} = I - D^{-1/2} W D^{-1/2}$.
- Compute first k eigenvectors $u_1, ..., u_k$ of L_{sym}.
- Let $U \in \mathbb{R}^{n \times k}$ contain $u_1, ..., u_k$.
- Form $T \in \mathbb{R}^{n \times k}$ by normalizing rows of U to norm 1.
- For $i = 1, ..., n$ let $y_i \in \mathbb{R}^k$ be a vector corresponding to the i-th row of T.
- Cluster (y_i) with k-means algorithm into clusters $C_1, ..., C_k$.

There are many justifications for why spectral clustering works. The simplest one is that spectral clustering can be viewed as a real value relaxation of the RatioCut and NCut problems, which are NP-hard. For more detail, we refer to [6].

Some comments about spectral clustering and how it is used in this report:

- For our calculations, we will only use Algorithm 3. This is because Algorithm 1 performs poorly on graph datasets with highly inhomogeneous distribution of degree counts, and because Algorithm 2 and Algorithm 3 give identical results on our datasets.

- As noted by [7], sometimes good divisions of the dataset are found by examining the third, not the second eigenvector. Newman applied spectral methods to the blog dataset and notes that the second eigenvector is localized around the high-degree vertices. It is a known fact that for highly inhomogeneous degree distributions, the smallest eigenvectors can be localized around the highest-degree vertices and thus compete with the eigenvectors that encode information about community structure. This phenomenon is perhaps undesirable - for some graphs with extremely inhomogeneous degree distribution, we may have to examine multiple eigenvectors and do guesswork before we come up with a suitable community division. In the Twitter dataset, we do not see this phenomenon occur.

- The last step of the spectral algorithms involves a k-means classification in k-dimensional space. However, in

this report we are seeking a binary classification, so we forgo this step in favor of a simpler approach where we classify into communities based on whether $y_{i,2} > 0$ or $y_{i,2} < 0$. Because the eigenvalues do not exactly form into two distinct clusters, this approach works better (see Fig. 5)

Implementations for spectral algorithms are provided in (5.2).

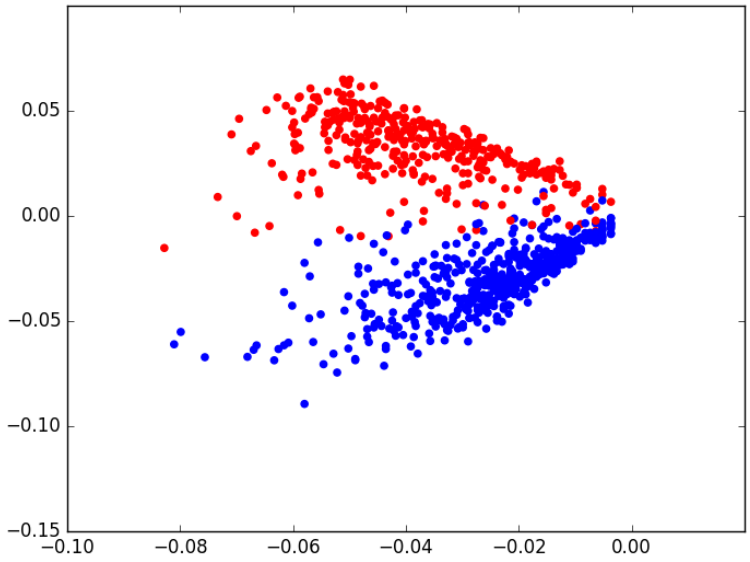


Figure 5: Plots of the first two eigenvectors for Twitter dataset. We see that spectral clustering allows us to separate the left-leaning (blue) from the right-leaning (red) if we classify based on whether $y_{i,2} > 0$ or $y_{i,2} < 0$.

2.3.2 Semi-definite Programming

We present here the semi-definite programming algorithm from [8]. For two-community symmetric case, we want to find a balanced partition of the nodes that has the least number of crossing edges (min-bisection). Thus, we can express this min-bisection problem as a quadratic optimization problem by labeling $\{+1, -1\}$ the two communities. This quadratic optimization problem is defined as:

$$\begin{aligned} \hat{x}_{\text{map}}(g) &= \arg\max_{x} x^t W(g) x \\ \text{s.t. } x &\in \{+1, -1\}^n, x^t 1^n = 0. \end{aligned} \tag{1}$$

(1) is difficult because of the integer constraint $x \in \{+1, -1\}^n$. Spectral clustering can be viewed as a real-value relaxation on this constraint which transforms the problem into an eigenvector problem. However, the SDP algorithm changes this quadratic optimization to linear optimization.

Note that we have

$$x^t W(g)x = tr(x^t W(g)x) = tr(W(g)xx^t), \tag{2}$$

so let us define $X := xx^t$, so we can rewrite (1) as:

$$\begin{gathered}\hat{X}_{\text{map}}(g) = \arg\max_X \text{tr}(W(g)X) \\ \text{s.t. } X \succeq 0; X_{ii} = 1 \; \forall i = 1, ..., n; \text{rank} X = 1; X1^n = 0\end{gathered} \tag{3}$$

Our SDP relaxation removes the rank$X = 1$ constraint in (2) and we get:

$$\begin{gathered}\hat{X}_{\text{sdp}}(g) = \arg\max_X \text{tr}(W(g)X) \\ \text{s.t. } X \succeq 0; X_{ii} = 1 \; \forall i = 1, ..., n; X1^n = 0\end{gathered} \tag{4}$$

Alternatively, let us define $B(g)$ as $B(g)_{ij} = 1$ if there is edge between v_i and v_j, -1 if there is no edge between v_i and v_j. This gives us another SDP:

$$\begin{gathered}\hat{X}_{\text{SDP}}(g) = \arg\max_X \text{tr}(B(g)X) \\ \text{s.t. } X \succeq 0; X_{ii} = 1 \; \forall i = 1, ..., n\end{gathered} \tag{5}$$

Thus our community detection algorithm solves (5), giving us a matrix $X \in \mathbb{R}^{n \times n}$ This matrix will not be rank 1 for real datasets because we removed that constraint, however we expect it will have low rank. Our classification step assumes the matrix X has close to rank 2. We then plot the first two dimensions of X, which will be points around the unit circle. We look for a line crossing (0,0) that partitions the points into two clusters C_1 and C_2 with the fewest crossing edges between C_1 and C_2.

The implementation for this algorithm can be found in (5.3).

2.3.3 Acyclic Belief Propagation

We present here the belief propagation algorithm from [9].

First let us introduce the stochastic block model, a popular random graph model for community detection.

Definition 1. *Let n be the number of vertices, k be number of communities, $p = (p_1, ..., p_k)$ probabilities representing vertices assignment to each community, and $\hat{W} \in \mathbb{R}^{k \times k}$ be a matrix with $(\hat{W})_{i,j}$ = probability of a connection from vertex in community i and community j. We draw (X,G) under $SBM(n, p, \hat{W})$ if X is a n-dimensional random vector drawn i.i.d. under p and G is a n-vertex undirected graph where vertices i and j are connected independently with probability $(\hat{W})_{X_i, X_j}$.*

Definition 2. *(X,G) is drawn under symmetric SSBM(n, k, A, B) if* $p = \{1/k\}^k$ *and* $\hat{W}$ *takes on values A on diagonal and B off diagonal.*

In this report, we implemented the 2-community symmetric case of the ABP algorithm. The ABP algorithm uses a random assignment of communities to each vertex and then improves on the belief for each vertex using information about the communities of its neighbors.

First of all, we define r to be a parameter of the length of cycles to detect in the preliminary step and m as the number of iterations to calculate.

1. We construct a graph G' where each edge v_1, v_2 is replaced by two directed edges $(v_1, \vec{v_2})$ and $(v_2, \vec{v_1})$.

2. For each directed edge $(v_1, \vec{v_2})$ we calculate whether this particular edge is in a cycle of length smaller or equal to r. If it is, we store the two neighbors v_3 and v_4, where v_3 is next to v_1 and v_4 is next to v_2.

3. We initialize $y^{(0)}$ as a random vector $\in \mathbb{R}^n$, whose elements are drawn i.i.d $\sim N(0, 1)$. This gives us our first belief in which communities each vertex is in.

4. Define matrix M with dimension $2|E(G)| \times m$, where we map from a directed edge (v_i, v_j) to an index in the array. Thus, the columns represent $y^{(t)}$, to be computed for $t = 1, ..., m$.

5. We initialize vector $y^{(1)} \in \mathbb{R}^{2*|E(G)|}$ as $y^{(1)}_{(v_1,v_2)} = y^{(0)}_{v_1}$, where we map each directed edge in G' to an index in $y^{(1)}$.

6. To iterate, for $2 \leq t \leq m$, we define $y^{(t)} \in \mathbb{R}^{2*|E(G)|}$ and set

$$y^{(t)}_{(v_1,v_2)} = \sum_{v_3:(v_3,v_1)\in E(G), v_3 \neq v_2} y^{(t-1)}_{(v_3,v_1)}$$

   which conceptually, amplifies beliefs for an edge based on beliefs of the other edges adjacent to that edge.

7. To take into account the possibility that we get "stuck" in a loop in the amplification step, for each directed edge $(v_1, \vec{v_2})$ that belongs to a cycle C of length $r' \leq r$ (which we found in step 2), set

$$y^{(t)}_{(v_1,v_2)} = \sum_{v_3:(v_3,v_1)\in E(G), v_3 \neq v_2} y^{(t-1)}_{(v_3,v_1)} - \sum_{v_4:(v_2,v_4)\in E(G), v_4 \in C} y^{(t-r'+1)}_{(v_2,v_4)}.$$

8. Redefine for all $v_2 \in G$ by summing up all the rows that correspond to an edge ending at v_2:

$$y_{v_2} = \sum_{v_1 \in V(G):(v_1,v_2)\in E(G)} y_{(v_1,v_2)},$$

thus redefining M to be this new $n \times m$ matrix.

9. To remove the eigenvector corresponding to the first component, multiply M by M_1 which has ones on the main diagonal and $-\lambda_1$ above the diagonal, where λ_1 is the ratio of the mean of $y^{(m)}$ and the mean of $y^{(m-1)}$ (the last and second to last columns of matrix M).

10. The last column of the previous step is the vector $y^{(m)}$ We classify each v_2 into C_1 if $y_{v_2}^{(m)} \leq 0$, into C_2 if $y_{v_2}^{(m)} > 0$.

For details on the intuition and steps of the full ABP algorithm, we refer to [9]. Our implementation can be found in (5.4).

2.4 Accuracy Calculation

Accuracy for our algorithms is defined as a Hamming distance [10]:

Definition 3. *For two vectors x, $\hat{x} \in \{0,1\}^n$ where x is the base truth and $\hat{x}$ is the predicted label, the two-community accuracy is defined as:*

$$A(x, \hat{x}) = \max\left(\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{x_i = \hat{x}_i}, \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{x_i \neq \hat{x}_i}\right).$$

2.5 SNR Calculation

This definition comes from [10].

Definition 4. *In the exact recovery regime $SBM(n, p, \hat{W})$, let $Q = \frac{n}{\log n}\hat{W}$. Then the signal-to-noise ratio is defined as*

$$\textit{SNR} = \lambda_2^2/\lambda_1,$$

where λ_1 is the largest eigenvalue of Q and λ_2 is the second largest eigenvalue of Q.

If this SNR > 1, we are able to exactly recover communities in poly-time ($\mathbb{P}\{A(x, \hat{x}) = 1\} = 1 - o(1)$) [9].

	L_{sym}	L_{rw}	$L_{sym,r}$	SDP	ABP
Error(%):	4.66	4.66	5.24	4.66	5.07

Table 2: Comparison of algorithms on benchmark blog dataset.

3 Results and Discussion

3.1 Political Blog Dataset

We first apply our algorithms to the political blog dataset. First, let us calculate the SNR for this dataset. Since the graph has 1222 nodes, we let $n = 1222$. Letting the first community be liberal and the second community be conservative, we find estimates for p and W:

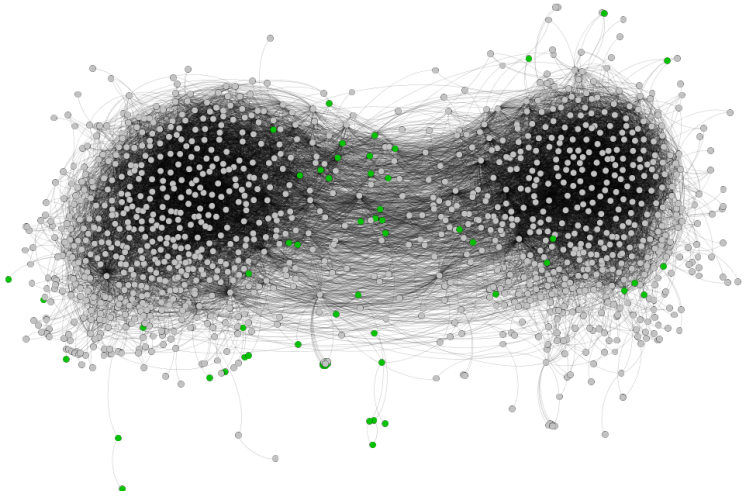
$$p = [0.48, 0.52], W = \begin{bmatrix} 0.0426 & 0.0042 \\ 0.0042 & 0.0388 \end{bmatrix}$$

Thus, we can calculate the SNR for this dataset in the exact recovery regime to be ≈ 2.5389.

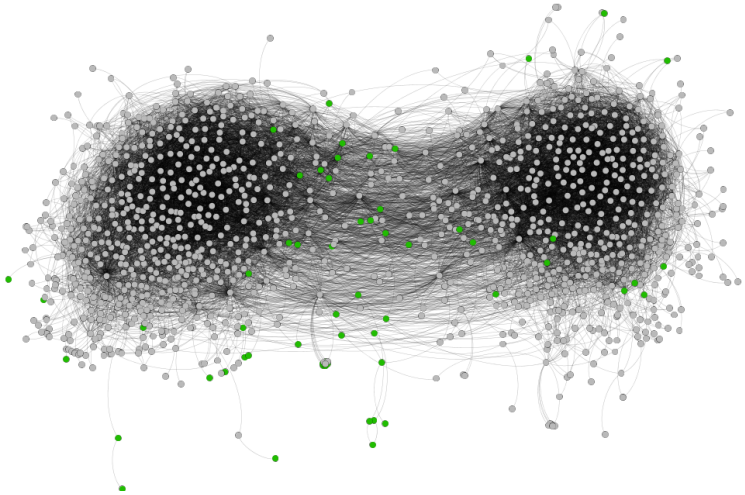
Since this SNR is above 1, we can be assured that our algorithms will perform reasonably well in recovering the communities. Indeed, they do, according to Table 2. Gephi visualizations of the nodes that are wrongly clustered can be found in Fig. 6. We see generally that the algorithms, as expected, misclassify the nodes that have similar left and right-leaning connections, i.e. the nodes located in-between the two community structures in the visualizations.

When we performed spectral clustering, we verified as in [7] that community structure is found by looking at the third eigenvector rather than the second eigenvector. We can see this clearly from Fig. 7. When implemented, we find that L_{sym} and L_{rw} normalize spectral clustering algorithms give identical results, however the unnormalized spectral clustering algorithm fails to produce any good results. We conjecture that this may be because of the highly inhomogeneous degree distribution, as we are able to recover clusters with the unnormalized spectral cluster algorithm with a simple graph drawn from a SBM. In general, spectral clustering is a fast algorithm for medium sized datasets like the ones we are using: our code in Python takes mere milliseconds to run. This is because spectral clustering algorithms only rely on simple linear algebra, and can be run with any standard linear algebra package such as `numpy`.

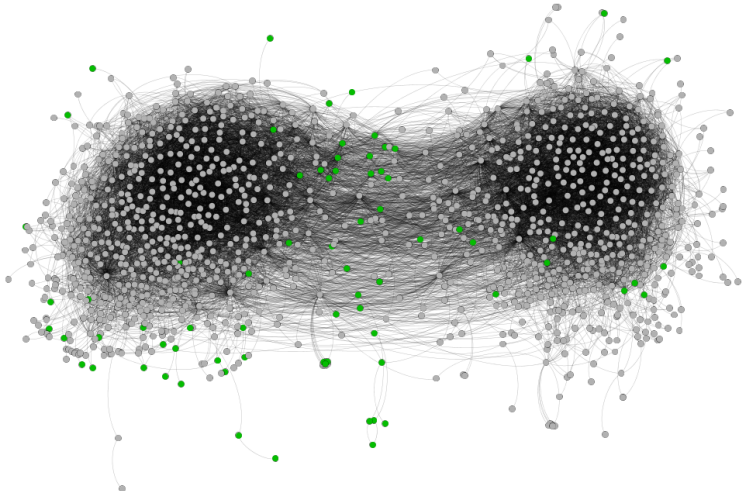
Next we turn to the SDP algorithm. For the blog dataset, the SDP algorithm performs identically to spectral clustering in terms of accuracy. However, the weakness of the SDP algorithm is that solving the SDP problem as-is requires a huge amount of memory and time. For this graph with $n = 1222$, I was unable to run on my laptop and had to resort to running the code on the Princeton Nobel cluster. CVX solver, the Matlab package I was using, reported that solving this SDP required 747,253 variables. For even larger SDP problems, we simply do not have the memory resources to solve. Thus, the SDP algorithm unfortunately does not scale well with with size.



(a) Wrongly classified nodes in spectral clustering

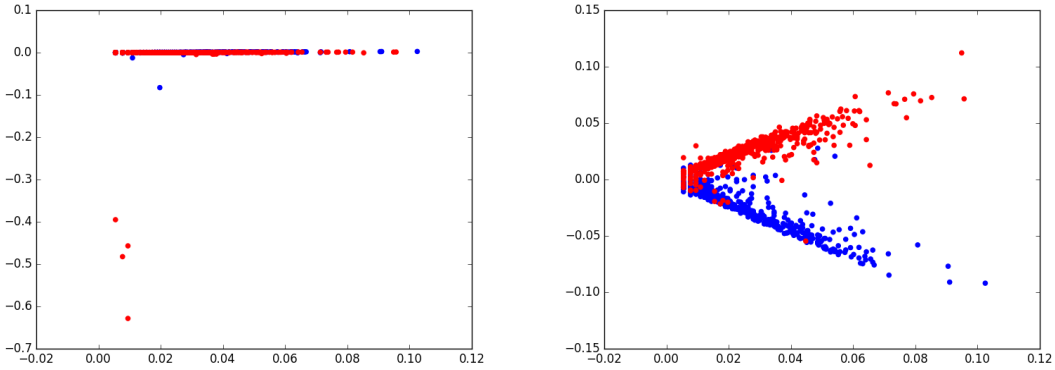


(b) Wrongly classified nodes in SDP algorithm clustering



(c) Wrongly classified nodes in ABP algorithm clustering

Figure 6: Visualizations of the error for all three algorithms in green. To orient the viewer, the left community is liberal, and the right community is conservative.



(a) First eigenvector vs second eigenvector

(b) First eigenvector vs third eigenvector

Figure 7: Eigenvector plots of L_{sym} spectral clustering algorithm. Blue represents liberals, and red represents conservatives.

The ABP algorithm does not perform as well as the the spectral algorithms and the SDP algorithms, but we implemented the "vanilla" version of the ABP algorithm for the two-community symmetric case, so it is possible a more robust version of the ABP algorithm can beat the spectral and SDP algorithms. The advantage of the ABP algorithm is that we do not have to go hunting for clusters by looking at different eigenvectors, as we often have to do for spectral clustering. According to [9], the ABP algorithm runs in $O(n\log(n))$, where n is the number of edges. Since the graph has a huge number of edges (16,714 to be exact), the ABP algorithm does take some time to run ($\approx$ 10 minutes). Again, it is conceivable a more robust version of the algorithm can be run much faster. If we view the convergence of the algorithm to detecting communities, we see from Fig. 8 that the by $m = 9$ the error converges to $\approx 5.07\%$.

3.2 Political Journalists Twitter Dataset

By looking at Fig. 4, it is clear that the Twitter dataset poses a greater challenge for community detection because of the large presence of "centrist" nodes which connect relatively equally to the left-leaning and the right-leaning nodes. The Gephi graph visualization software is not able to separate the graph into two communities. Moreover, it is not so clear whether the centrist nodes form their own community, separate from the left and right-leaning journalists, or whether they serve as a "bridge" between the two communities.

When we apply our algorithms to the full dataset of Twitter journalists, they fail to recover the communities in the conventional left-right way as defined by the "Left/Right" metric. Instead, it seems like the algorithms separate the large group of extreme right-leaning Twitter journalists ("Left/Right" metric > 15 in Fig. 3). This roughly corresponds to the large group colored red in Fig. 4. Indeed, if we only consider extreme leaning journalists (defined as having "Left/Right" metric < -15 or > 15), we find that that group of extreme right-leaning Twitter journalists has much

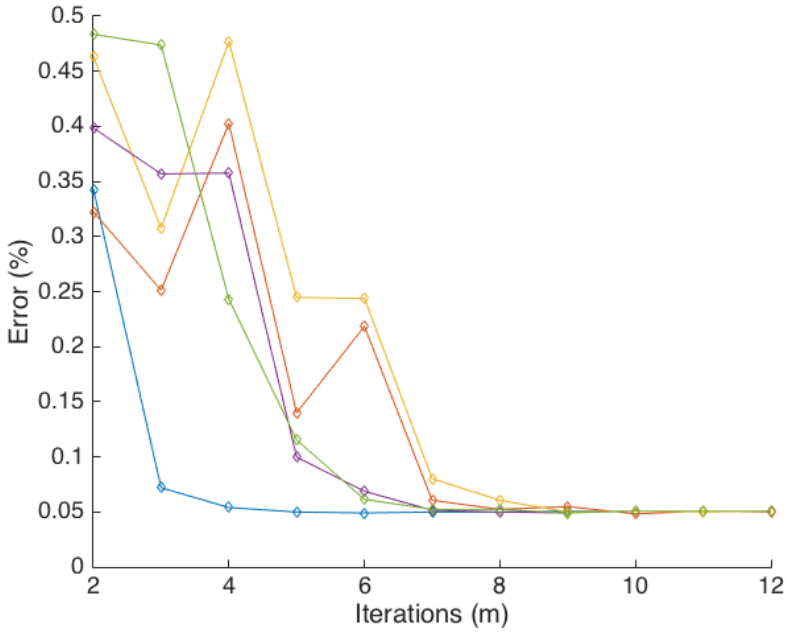


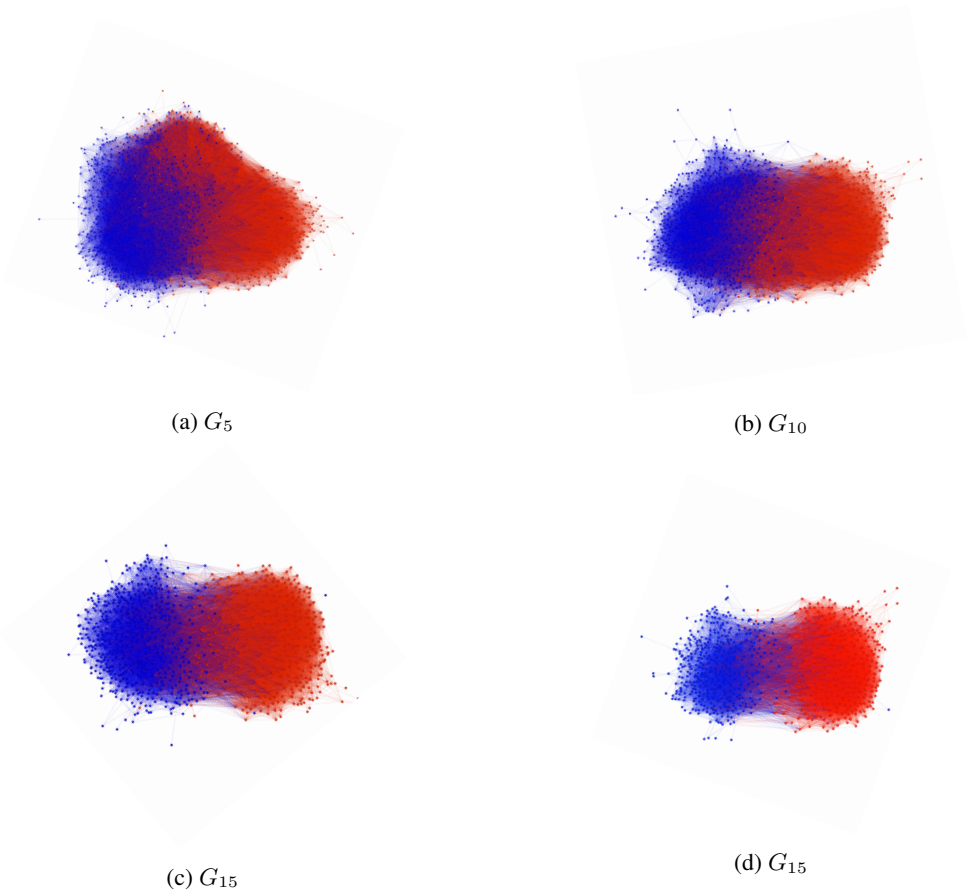
Figure 8: We plot 5 trials of the error of the ABP algorithm on the blog dataset.

higher edge density than the other nodes (≈ 0.294, when the edge density for the extreme left-leaning journalists is ≈ 0.095 and the edge density between the extreme left-leaning and extreme right-leaning is ≈ 0.0419). Our algorithms' resulting performance on this dataset is not so much an indictment of the flaws of the algorithms. Rather, we can clearly observe the complexity of the graph dataset and how simple 2-community clustering may not be the appropriate model for this dataset. We see in this model we are actually separating between extreme right-leaning and the rest of the moderates + extreme left journalists/bloggers.

Nevertheless, we will proceed with 2-community clustering by selectively removing the "centrist" nodes from the graph, as these "centrist" nodes are giving us the most problems in applying the algorithms. This is not ideal, after all, the "centrist" nodes are major components of the Twitter graph. However, for the sake of comparing 2-community clustering to our benchmark political blogs dataset, we will opt to build truncated datasets G_k that remove all nodes with "Left/Right" metric in the range $(-k, k)$ for various k. Future work with this dataset should be focused on understanding the community structure of the graph with all the "centrist" nodes included.

Gephi visualization of the graphs for various G_k are shown in Fig. 9. Note that as we remove "centrist" nodes, Gephi's visualization is able to separate the communities more clearly, which bodes well for the performance of our 2-community clustering algorithms.

SNR calculations in the exact-recovery regime are shown in Fig. 10. From this figure, we can see that these G_k generally have SNR > 1. Note that G_0 corresponds to the original dataset. At this time, we do not have a good explanation for why the SNR peaks at around G_{15}, or why it decreases the more we "polarize" the communities



(a) G_5

(b) G_{10}

(c) G_{15}

(d) G_{15}

Figure 9: Gephi visualizations for various G_k

(increase k above 15).

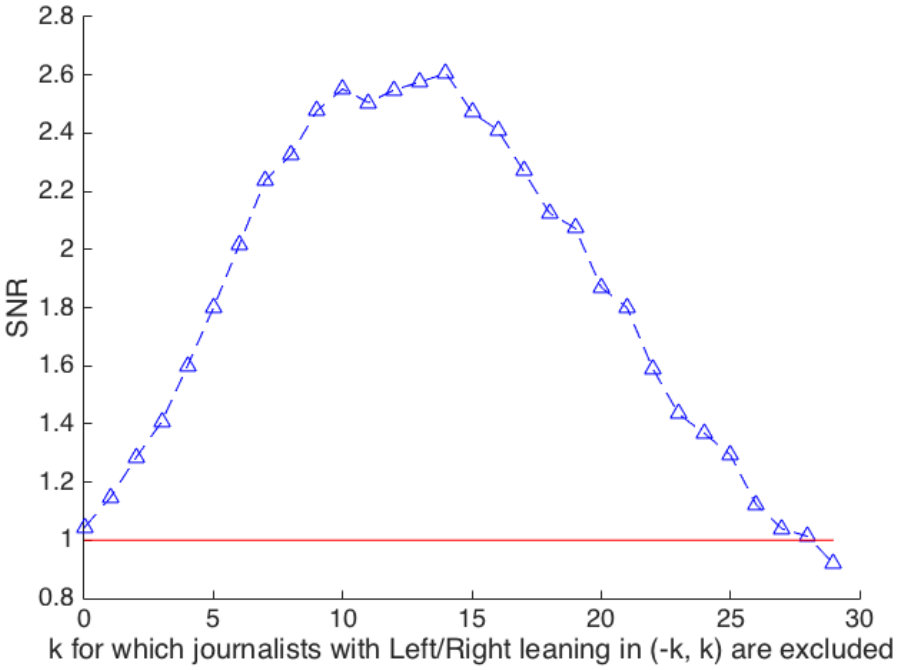


Figure 10: SNR for various G_k

Results of applying each algorithm to various G_k can be found in Fig. 11.

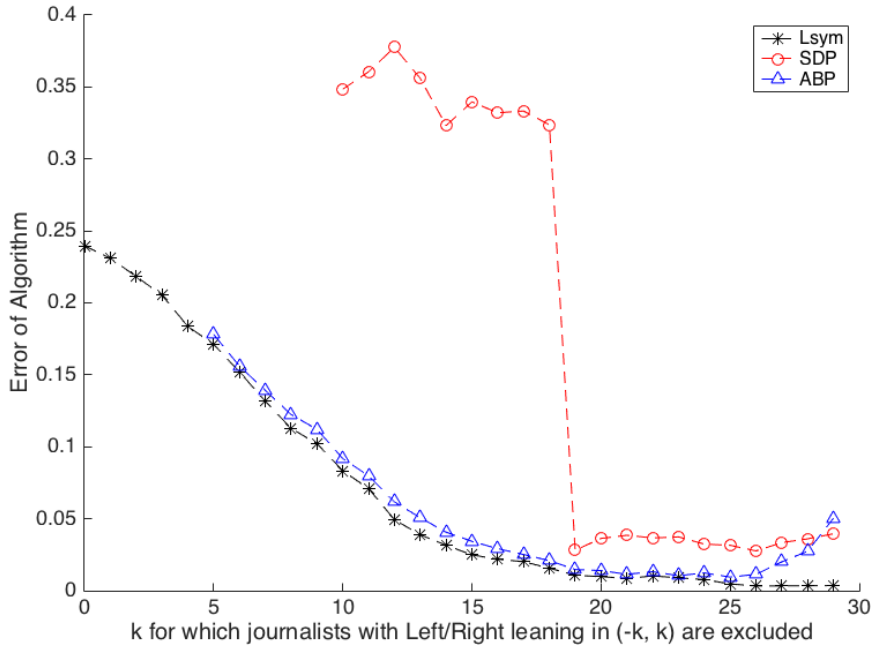


Figure 11: Error of each algorithm for various G_k

From Fig. 11, we can see that the spectral algorithm of L_{sym} performs slightly better than the ABP algorithm. Unlike in the blog dataset, the community structure is always located in the second eigenvectors, so we did not have to examine multiple eigenvectors to come up with a clustering. Again, L_{sym} and L_{rw} perform identically, so our results only include L_{sym}.

For the large dataset of the Twitter graph, SDP does not perform well. It is unable to be run even on the Princeton Nobel Cluster for $k < 10$ because of memory constraints. Moreover, the algorithm fails to recover clusters for all G_k where $k < 19$. The reason why it is unable to recover the clusters for $k < 19$ is unknown. This experiment further validates the issues with SDP, namely on larger datasets on the order of $n = 1000$.

The ABP compares well with the spectral clustering algorithms. We can see that as we include more centrist users (graphs G_k with lower k), both spectral and the ABP algorithms perform more poorly in clustering, which is to be expected.

It is not exactly clear why, even when SNR is decreasing in the $k = (15, 30)$ range, we are still getting more accurate clustering for almost all the algorithms.

4 Conclusions and Future Directions

We developed a new graph dataset of political journalists on Twitter that parallels the political blogs dataset in 2005 that is widely used as a benchmark. We apply three community detection algorithms, namely the spectral algorithm, a semi-definite programming algorithm, and an acyclic belief propagation algorithm, on both the political blog dataset and our political Twitter journalist dataset. We find that the Twitter political journalists dataset is much more complex than the blogs dataset because of the strong presence of "centrist" nodes which connect to both left-leaning journalists and right-leaning journalists, as well as the fact that there exists a strong extreme right-leaning journalists group which has high interior edge density. Overall, the spectral algorithm seems to outperform the other two; however, with the spectral algorithm we sometimes have to "hunt" for the correct eigenvector for clustering, as in the blog dataset the correct eigenvector is the third, rather than the second. The SDP algorithm suffers from memory constraints for medium sized graphs ($n \approx 1000$).

Our work leaves many unanswered questions which we would like to address in the future. We would like to further analyze the measure of SNR on real datasets, since currently it is mostly used on understanding theoretic limits for recovery in the SBM. In terms of the algorithms that we used, we would like to refine the SDP approach so that it is is not memory limited, perhaps developing better heuristics to solve the SDP problem. We would also like to implement the more robust version of the ABP algorithm and apply it to more real world datasets. In addition, perhaps we can build a more realistic model of graphs that takes into account a continuous "gradient" of connectivity instead of discrete communities, which we observe in the Twitter dataset as different nodes have different values of "Left/Right" metric along a scale that impact how they connect within the graph.

5 Appendix

5.1 Web Scraping Code

userdatascraper.py

This code collects the initial Twitter data about each Twitter journalist. The ID numbers collected are used subsequently to identify each Twitter account.

```
import pandas as pd
import twitter
import json
import time

# read in StatSocial lists of top left, right and centrist journalists
left = pd.read_csv('../data/topleft.csv')
right = pd.read_csv('../data/topright.csv')
centrist = pd.read_csv('../data/topcentrist.csv')

# consumerkey, consumersecret, accesstoken, accesssecret are supplied when
# you register for the twitter API.
api = twitter.Api(consumer_key = consumerkey,
    consumer_secret = consumersecret,
    access_token_key = accesstoken,
    access_token_secret = accesssecret,
    sleep_on_rate_limit = True)

# filter of columns from twitter JSON file that we want
subset_keys = ['id', 'screen_name', 'description', 'followers_count',
               'friends_count', 'favourites_count', 'statuses_count']
# filter of columsn from csv files that we want.
csv_keys = ['Pull', 'Left', 'Right', 'Left/Right']

index = 0 # instantiate index for inserting into dataframe.
df = pd.DataFrame() #instantiate new dataframe.

for screen_name in left['Username']:
    if (pd.isnull(screen_name)): break
    try:
        usr = api.GetUser(screen_name = screen_name)
        usrdict = usr.AsDict()
        usrdict = dict((k, usrdict[k]) for k in subset_keys if k in usrdict)
        usrdict['Label'] = 'Left' # add classification column
        # add some data to the Dataframe row from original csv file.
        for key in csv_keys:
            usrdict[key] = left[left['Username'] == screen_name][key].iloc[0]
        dfrow = pd.DataFrame(usrdict, index = [index])
        # add row to dataframe.
        df = pd.concat([df, dfrow], axis = 0)
        index = index + 1
    except:
        print("Oops! ", screen_name, " data was not saved")

for screen_name in right['Username']:
    if (pd.isnull(screen_name)): break
    try:
        usr = api.GetUser(screen_name = screen_name)
        usrdict = usr.AsDict()
        usrdict = dict((k, usrdict[k]) for k in subset_keys if k in usrdict)
        usrdict['Label'] = 'Right'
        for key in csv_keys:
            usrdict[key] = right[right['Username'] == screen_name][key].iloc[0]
        dfrow = pd.DataFrame(usrdict, index = [index])
        df = pd.concat([df, dfrow], axis = 0)
        index = index + 1
    except:
        print("Oops! ", screen_name, " data was not saved")

for screen_name in centrist['Username']:
    if (pd.isnull(screen_name)): break
    try:
        usr = api.GetUser(screen_name = screen_name)
```

```
        usrdict = usr.AsDict()
        usrdict = dict((k, usrdict[k]) for k in subset_keys if k in usrdict)
        usrdict['Label'] = 'Centrist'
        for key in csv_keys:
            usrdict[key] = centrist[centrist['Username'] == screen_name][key].iloc[0]
        dfrow = pd.DataFrame(usrdict, index = [index])
        df = pd.concat([df, dfrow], axis = 0)
        index = index + 1
    except:
        print("Oops! ", screen_name, " data was not saved")

# write data to csv file.
df.to_csv('../data/politicaljournalistst witter.csv')
```

graphscraper.py

Using the IDs found with userdatascraper.py, we query the Twitter API for the friends of each ID.

```
import pandas as pd
import twitter
import json
import time

api = twitter.Api(consumer_key = consumerkey,
    consumer_secret = consumersecret,
    access_token_key = accesstoken,
    access_token_secret = accesssecret,
    sleep_on_rate_limit = True)

# read in csv files.
journalists = pd.read_csv('politicaljournalistst witter.csv')
graphnodes = open("graphnodes.txt", "w")
ids = set() # hash the IDs to support fast lookup.

for id in journalists['id']:
    if(pd.isnull(id)): break
    ids.add(id)
    graphnodes.write(str(id) + "\n")

graphnodes.close()
graphedges = open("graphedges.txt", "w")
# error log of ids which we can't retrieve edges for.
grapherror = open("grapherror.txt", "w")

# for each user id, get a list of their friend IDs.
count = 1
for id in ids:
    cursor = -1
    while True:
        try:
            nc, pc, data = api.GetFriendIDsPaged(user_id = id, cursor = cursor, count = 5000)
            cursor = nc
            flagger = 0
            for friend in data:
                # for each friend ID, record connection between original user ID and friend ID.
                if (friend in ids):
                    flagger = 1
                    graphedges.write(str(id) + " " + str(friend) + "\n")
            if (flagger == 0):
                grapherror.write("error: " + str (id))
            if (cursor == 0):
                break
        except:
            grapherror.write("error: " + str (id))
            break
    count = count + 1
    if (count % 100 == 0):
        print(count, " done")

graphedges.close()
grapherror.close()
```

5.2 Spectral Algorithms

Unnormalized Spectral Clustering

```
# W is adjacency matrix, k is number of communities to find.
def spectral(W, k):
    D = np.diag(np.sum(W, axis=1))
    L = D - W
    Leval, Levec = scipy.linalg.eig(L)
    # sort eigenvalues and eigenvectors by size of eigenvalue.
    idx = Leval.argsort()
    Leval = Leval[idx]
    Levec = Levec[:,idx]

    return [1 if i > 0 else 0 for i in Levec[:,1]]
```

Normalized Spectral Clustering according to Shi and Malik

```
# W is adjacency matrix, k is number of communities to find.
def spectralrw(W, k):
    D = np.diag(np.sum(W, axis=1))
    L = D - W
    Dneg1 = scipy.linalg.inv(D)
    Lrw = Dneg1.dot(L)
    Lrweval, Lrwevec = scipy.linalg.eig(Lrw)
    idx = Lrweval.argsort()
    Lrweval = Lrweval[idx]
    Lrwevec = Lrwevec[:,idx]

    return [1 if i > 0 else 0 for i in Lrwevec[:,1]]
```

Normalized Spectral Clustering according to Ng, Jordan, and Weiss

```
# W is adjacency matrix, k is number of communities to find.
def spectralsym(W, k):
    D = np.diag(np.sum(W, axis=1))
    L = D - W
    DnegRoot = scipy.linalg.fractional_matrix_power(D, -1/2)
    Lsym = DnegRoot.dot(L).dot(DnegRoot)
    Lsymeval, Lsymevec = scipy.linalg.eig(Lsym)
    idx = Lsymeval.argsort()
    Lsymeval = Lsymeval[idx]
    Lsymevec = Lsymevec[:,idx]

    # normalize
    T = Lsymevec[:,0:k]
    norm = np.sqrt((T * T).sum(axis=1))
    T / norm.reshape(len(norm),1)

    return [1 if i > 0 else 0 for i in T[:,1]]
```

5.3 Semi-definite Programming Algorithm

For our semi-definite programming solver, we use CVX, a Matlab software for convex programming created by Michael C. Grant and Stephen P. Boyd. CVX is available at http://cvxr.com/cvx/.

The industrial solver CVX uses is MOSEK, a high performance optimization software package. It can be found at https://mosek.com/.

sdp.m

```
matrix = ... % some csv file where we stored the adjacency matrix
labels = ... % some csv file where we stored the labels.
A = csvread(matrix); % read in the adjacency matrix
Y = csvread(labels); % read in true labels
n = size(A, 1);
B = -(A==0) + A; % calculate B.

% call cvx software.
cvx_begin sdp
  variable X(n,n) symmetric
  X == semidefinite(n);
  diag(X) == 1
  maximize(trace(B*X));
cvx_end

% approximate min-bisection by "turning" a bisection line from 91 degrees to 179 degrees
deg = [91:179];
rad = degtorad(deg);
slope = tan(rad);

% initialize minimum crossing edges to maximum values.
mince = sum(sum(A));
% initialize accuracy to 1.
maxxacc = 1;

% change our partition passing through (0,0) to find min-bisection
for i = 1:length(deg)
  % this is our labeling for this partition.
  label = X(:,2) > slope(i)*X(:,1);
  [crossedge, acc] = getNumCrossingEdges(label, A, Y);
  if (mince > numCrossEdge)
    mince = numCrossEdge;
    maxacc = acc;
  end
end

% maxacc is our accuracy.
```

getNumCrossingEdges.m

```
function [ crossedge, acc ] = getNumCrossingEdges( labels, A, Y )
  n = size(A, 1);
  crossedge = 0;
  % iterate through all edges and count all that cross.
  for i = 1:1:n
    for j = i+1:1:n
              if (A(i, j) && labels(i) ~= labels(j))
                  crossedge = crossedge + 1;
              end
    end
  end

  % accuracy calculation
  acc = max(sum(Y == labels)/n, sum(Y ~= labels)/n);
end
```

5.4 Acyclic Belief Propagation Algorithm

abp.m

```
% Acyclic Belief Propagation Algorithm
% 2 Community Symmetric Case

matrix = ... % some csv file where we stored the adjacency matrix
labels = ... % some csv file where we stored the labels.
A = csvread(matrix); % read in the adjacency matrix
Y = csvread(labels); % read in true labels

% remove all self-loops
for i = 1:length(A)
   if (A(i, i) == 1)
       A(i, i) = 0;
   end
end

r = 6; % parameter for non-backtracking walks.
m = 20; % parameter for length of walks to propagate values.

numE = sum(sum(A))/2; % sum of A divided by 2 because edges double counted.
numV = length(A);

% mapping of each edge (v1, v2) to an index.
edgeIndexed = zeros(numE, 2);
index = 1;
for i = 1:numV
    for j = i+1:numV
        if (A(i,j) == 1)
            edgeIndexed(index,:) = [i, j];
            index = index + 1;
        end
    end
end

e2 = [edgeIndexed(:,2), edgeIndexed(:,1)];
% edge Indexed is 2|E(G)| x 2 now
edgeIndexed = [edgeIndexed; e2];

% store the indices in a matrix, so we can support fast lookup for indices given e1 and e2.
edgemat = zeros(numV);
for i = 1:2*numE
    e = edgeIndexed(i,:);
    edgemat(e(1), e(2)) = i;
end

% Compute list D of length |E(G)|, where ith element represents vector.
% if ith edge v1-v2 is NOT in cycle length <= r, vector has elements 0.
% if ith edge v1-v2 is IN cycle length <= r.
% vector has length 4: [1, length(C), neighbors v1/v2]

% Matrix storing this information.
D = zeros(2*numE, 4);

for i = 1:numE
    D(i,:) = calculateCycle(edgeIndexed(i,1), edgeIndexed(i,2), A, r);
    D(i+numE,:) = [D(i,1), D(i,2), D(i,4), D(i,3)];
end

% initialize y0, drawn IID from N(0,1)
y0 = randn(numV,1);

M = NaN(2*numE, m);
for t = 1:m
   for i = 1:2*numE
       e = edgeIndexed(i,:);
       M = walklengtht(A, e(1), e(2), t, r, edgemat, D, y0, M);
   end
end

M2 = zeros(numV, m);
for row = 1:2*numE
    edge = edgeIndexed(row,:);
    v2 = edge(2);
```

```
    M2(v2,:) = M2(v2,:) + M(row,:);
end

M = M2;

% Compensation step. Get rids of component proportional to first
% eigenvector.

lambda1 = mean(M(:,m))/mean(M(:,m-1));

M1 = diag(-lambda1 .* ones(m-1,1), 1) + eye(m);
M = M * M1;

% Classification
c2 = median(M(:,m));
class = M(:,m) > c2;

acc = max(sum(Y == class)/numV, sum(Y ~= class)/numV);
```

calculateCycle.m

```
function [ cycle ] = calculateCycle(v1, v2, A, r)
    % row to return.
    cycle = zeros(1,4);

    % delete edge from A.
    A(v1, v2) = 0;
    A(v2, v1) = 0;

    % run BFS on v1 with max of r-1 steps
    % see if you can reach v2 within r steps.
    % enqueue v1
    visited = zeros(length(A),1);
    edgeFrom = zeros(length(A),1);
    queue = [v1];
    for i = 1:1:r-1
        % find adj to the current queue.
        visited(queue) = 1;
        totaladj = [];
        for j = 1:1:length(queue)
            adj = find(A(queue(j),:) == 1);
            adj = adj(visited(adj) == 0);
            edgeFrom(adj) = queue(j);
            if (any(adj==v2))
                % calculate cycle.
                cyclevertices = [];
                v = v2;
                while (v ~= 0)
                    cyclevertices = [v, cyclevertices];
                    v = edgeFrom(v);
                end
                cycle = [1, length(cyclevertices), cyclevertices(2), cyclevertices(length(
    cyclevertices) - 1)];
                return
            end
            totaladj = [totaladj, adj];

        end
        queue = totaladj(visited(totaladj) == 0);
    end
end
```

walklengtht.m

```
function [ M ] = walklengtht( A, v1, v2, t, r, edgemat, D, y0, M)
%walklengtht
%   Stores into Matrix of dimension 2 * |E(G)| x m,
%   where (i, j) is y(j) (v1, v2), where v1, v2 = i.
%   number of r-nonbacktracking walks of length t ending at directed edge
%   v1, v2. Otherwise it is NaN.

    numE = sum(sum(A))/2;

    % initial step y(1) (v1,v2) = y(0) v1.
    if (t == 1)
```

```
        % find edge index.
        % set M[edge index, 1] = y0[v1].
        k = edgemat(v1,v2);
        M(k, 1) = y0(v1);
    % when t == 2, sum all edges going to v1, except for v1-v2 edge.
    elseif (t == 2)
        sm = sum(y0(A(v1,:) == 1)) - y0(v2);
        k = edgemat(v1,v2);
        M(k, 2) = sm;
    else
        adj = find(A(v1,:) == 1);
        sm = 0;
        % calculate first part of sum:
        for i = 1:length(adj)
            v3 = adj(i);
            k = edgemat(v3, v1);
            if (isnan(M(k, t-1)))
                M = walklengtht(A, v3, v1, t-1, r, edgemat, D, y0, M);
            end
                sm = sm + M(k, t-1);
        end

        % subtract out the v2->v1 edge
        k = edgemat(v2,v1);
        if (isnan(M(k, t-1)))
            M = walklengtht(A, v2, v1, t-1, r, edgemat, D, y0, M);
        end
        sm = sm - M(k, t-1);

        % adjust for r - non backtracking step
        % get the appropriate row from the cycles list.
        k = edgemat(v1,v2);
        Drow = D(k,:);

        % if v1->v2 is in a cycle.
        if(Drow(1,1) == 1)
            % length of the cycle.
            r2 = Drow(2);
            if (t > r2-1 && r2 < r + 1)
                % find v4.
                v4 = Drow(4);
                k = edgemat(v2, v4);
                if (isnan(M(k, t-r2+1)))
                    M = walklengtht(A, v2, v4, t-r2+1, r, edgemat, D, y0, M);
                end
                sm = sm-M(k, t-r2+1);
            end
        end

        k = edgemat(v1,v2);
        M(k, t) = sm;

    end
end
```

References

[1] Lada A. Adamic and Natalie 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.

[2] Maeve Duggan and Aaron Smith. The political environment on social media, 2016.

[3] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, February 2010.

[4] StatSocial. The most influential political journalists and bloggers in social media, 2015.

[5] StatSocial. What is social pull?, 2012.

[6] U. von Luxburg. A Tutorial on Spectral Clustering. *ArXiv e-prints*, November 2007.

[7] M. E. J. Newman. Spectral methods for community detection and graph partitioning. *Phys. Rev. E*, 88:042822, Oct 2013.

[8] E. Abbe, A. S. Bandeira, and G. Hall. Exact recovery in the stochastic block model. *IEEE Transactions on Information Theory*, 62(1):471–487, Jan 2016.

[9] E. Abbe and C. Sandon. Detection in the stochastic block model with multiple clusters: proof of the achievability conjectures, acyclic BP, and the information-computation gap. *ArXiv e-prints*, December 2015.

[10] E. Abbe. Community detection and stochastic block models: recent developments. *ArXiv e-prints*, March 2017.