12-2016

Efficient Analytics for Large Implicit Correlation Networks

Raghuveer Mohan
Clemson University, rmohan@g.clemson.edu

Efficient Analytics for Large Implicit Correlation Networks

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Raghuveer Mohan
December 2016

Accepted by:
Dr. Brian Dean, Committee Chair
Dr. Wayne Goddard
Dr. Ilya Safro
Dr. Feng Luo

Abstract

Large "correlation"-type networks have applications in machine learning, data mining and analysis of large biological networks. These networks contain short length-k (on the order of thousands) feature vectors for every node in the network that describe the characteristic or behavior of that node. Edges between nodes are expressed as correlations, partial correlations or coherence between the corresponding feature vectors. If the number of nodes n is too large (on the order of millions), then the dense correlation network with size $n \times n$ is computationally infeasible even to explicitly store in memory, let alone to use for computation of standard network analytics like clustering or centrality. In this dissertation, we provide an algorithmic framework to analyze large correlation networks efficiently in an implicit fashion, without writing down every entry of the network.

Our framework allows us to compute various network analytics like clustering and centrality using spectral techniques (based on linear algebra). It can accommodate several more sophisticated edge weight measurements beyond just correlation, including partial correlation and coherence. Since correlation networks can contain negative edge weights, one of the main innovations of our work is a collection of transformations that re-map node feature vectors so as to produce a non-negative network in which edge weights are approximately the squares of the original weights. We also develop a number of other useful algorithmic tools (e.g., implicit calculation of shrinkage estimators) that play an important role along our implicit computational pipeline.

Dedication

Every thought, emotion and word that formed this dissertation is dedicated to my mother Geetha and my grandmother Ranga.

Acknowledgments

First and foremost, I would like to thank my adviser Dr. Brian C. Dean, who has been the greatest source of inspiration and influence in my life. I thank him for the time and effort he has taken to ensure I would succeed in getting my PhD – by motivating me when I was showing progress, and by being patient when I was not. I appreciate his input into my research and his continuous support and guidance during my entire PhD study. I also thank him for his guidance during the writing of this dissertation, and in carefully reviewing several drafts of this document.

Being the director of the USA Computing Olympiad, Dr. Dean spent many sessions in brainstorming ideas about problems in USACO and other programming contests. These sessions were thoroughly enjoyable and got me very excited about algorithms and data structures. I greatly admire his creative problem solving abilities to not only come up with novel and innovative ideas about research, but also in creating fun and exciting problems for USACO and his algorithms courses, which highlight the use of several algorithmic techniques. I have spent countless hours thinking about these problems, and have used many of them in courses I teach, since they have excellent pedagogical value.

During my time at Clemson, Dr. Dean gave me many opportunities to teach – which ranged from lectures in undergraduate data structures, to graduate algorithms, to more advanced topics in algorithms research. Thanks to his recommendation, I had the opportunity to be the instructor-of-record for a semester long course on undergraduate data structures and algorithms, which I immensely enjoyed, and which enabled me to receive the department's annual teaching assistant award. With the excuse of taking lecture videos for Dr. Dean's courses, I began to observe and study a true master teacher at work. Largely influenced by his teaching, I could develop my own teaching skills, style and philosophy. He has shown me the art form of computer science, and in him I find a master artist who paints some of the most beautiful pictures.

Funding for my PhD came from many sources. I alternated between teaching and research assistantships every semester. I thank Dr. Mark Smotherman for giving me various teaching opportunities and for funding my teaching assistantships through the computer science department. I thank Dr. Dean for funding my research assistantships through his NSF career grant for the first few years, and through his joint collaboration with Dr. Jane Joseph at the Medical University of South Carolina for the latter years. Funding in the summer came largely from Dr. Dean's REU grant, and I profusely thank him for all the funding during my PhD studies.

I thank the rest of my PhD committee – Dr. Feng Luo, Dr. Wayne Goddard and Dr. Ilya Safro for taking their time to carefully review my thesis.

I thank the members of the Applied Algorithms Group, specifically Rommel Jalasutaram, Chad Waters and Matthew Dabbney for the fun we had while pursuing our PhD degrees together. We brainstormed ideas for research, shared PhD experiences, and other academic and non-academic interests, and motivated each other to reach our milestones. These interactions made my PhD experience very memorable.

I thank Dr. Steve Hedetniemi for his weekly meetings, during which we talked about philosophy and spirituality. He kindled my interests in these fields, and I thank him for opening my mind to many possibilities about life and the universe.

I thank all my family and friends for encouraging me and offering emotional support during my PhD. Their motivational talks allowed me to gather confidence to move forward with research. I finally, thank my mother Geetha, for all her love and affection, without which this dissertation would not be possible.

Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Dedication . . . . . **iii**

Acknowledgments . . . . . **iv**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

1 Introduction **1**
1.1 Network Analytics 3
1.2 Dot Product Networks 8
1.3 Applications 11
1.4 Results 14
1.5 Roadmap 15

2 Preliminaries **17**
2.1 Notation 17
2.2 Block Multiplication 17
2.3 The Singular Value Decomposition 18
2.4 Locality Sensitive Hashing 22
2.5 Other Edge Weight Representations 23
2.6 Dimensionality Reduction via Random Projection 30

3 Centrality **35**
3.1 Non-negative Dot Product Networks 36
3.2 Negative Dot Product Networks 40

4 Clustering and Cuts **45**
4.1 Non-Negative Dot Product Networks 45
4.2 Negative Dot Product Networks 59

5 Locality Sensitive Hashing **62**
5.1 Exact Mapping For Squared Correlations 63
5.2 Approximate Mapping Using LSH 65
5.3 Estimation of Coherence 75

6 Conclusions **78**

Bibliography **80**

List of Tables

2.1 Table of implicit calculation of optimal value of the shrinkage parameter γ for four of the common targets provided in [96], where $d_C = \text{trace}(S^2) - 2c\text{sum}(S) + 2(c - v)\text{trace}(S) + \frac{n(n-1)c^2}{2} + nv^2$ and $n_D = c_1\text{sum}(W') + c_2\text{trace}(W^2) - c_1\text{trace}(W_2) - c_2\text{trace}(W')$. 27

List of Figures

4.1 Principal component interpretation of the largest eigenvector of a negative correlation network. 60

5.1 (a) A two dimensional linearly inseparable dataset. (b) The dataset projected into three dimensions along a paraboloid $z = x^2 + y^2$ becomes linearly separable. 64

5.2 (a) The regions for which the hash function $h(u) = \xi$ for a specific random vector $\widehat{r}$. (b) The area of the two spherical caps gives the probability that the hashes of both vectors u and v are ξ. 66

5.3 The probability determined by the area of the shaded region gives the equation of the LSH curve. 69

5.4 Squares of correlations in the range $[0, 1]$ described by the curve in red, and the LSH curve defined by $\mathrm{E}\left[h(u) \cdot h(v)\right]$ in blue using functions (a) $f(x)$, a step function; (b) $f(x) = x^4$; and (c) $f(x) = \mathrm{erf}(\frac{x}{\sqrt{2}})^{18}$. We also show $f(x) = \tanh(\frac{x}{\sqrt{2}})^{18}$ in green that approximates erf. For a hash vector $r = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$, the regions in white show positive admissions with the hash function using (d) $f(x)$, a step function; (e) $f(x) = x^4$; and (f) $f(x) = \mathrm{erf}(\frac{x}{\sqrt{2}})^{18}$. 71

5.5 (a) Shapes of the Gaussian error function $\mathrm{erf}(\frac{x}{\sqrt{2}})$ in red and the hyperbolic tangent $\tanh(\frac{x}{\sqrt{2}})$ in blue. (b) Shapes of functions $\mathrm{erf}(\frac{x}{\sqrt{2}})^{18}$ in blue and $\tanh(\frac{x}{\sqrt{2}})^{18}$ in red. The hyperbolic tangent approximates the Gaussian error function. 74

Chapter 1

Introduction

In this dissertation, we study network analytics on certain types of similarity networks that are too large to store explicitly in memory. We develop a framework in which we can compute several analytics on these networks in an implicit fashion, without even writing down all the entries of the network. Such analysis is vital in biomedical fields and in other areas dealing with large similarity networks, such as in data mining and machine learning.

Similarity networks are usually derived from vectors of "features" associated with every node in the network, that represent some aspect of its nature or characteristic. Edges between nodes are weighted by a similarity function on the corresponding feature vectors. For example, in social networks, individuals might each provide a feature vector of desirability of presidential candidates, where the similarity between individuals is determined by the correlation of the corresponding feature vectors. In functional magnetic resonance imaging (fMRI) networks, each region of the brain is associated with a feature vector in the form of a time series measuring blood oxygen consumption over a period of time, which detects neural activations during an experiment. The correlation or coherence between corresponding time series determines the functional connectivity between regions. In these cases and many others, an edge exists between every pair of nodes in the network. For a network with, say, $100,000$ nodes or more, it is computationally stressful even to represent all the pairwise similarities in memory explicitly, let alone run algorithms to compute analytics. Specifically in the field of fMRI coherence analysis, the author of the book "Statistical Analysis of fMRI Data" [12] notes that "The exorbitant computing time this would require makes such calculations highly impractical, but even if all these coherence values were computed, Herculean efforts would be required

to sift through them all and interpret the results. For these reasons, current applications do not attempt to compute all possible coherence values". The current practice in fMRI analysis is to completely evade this issue and only look at coherence or correlations between a few regions of interest. This leads to an important question: can we realistically perform coherence analysis on all the nodes of a large fMRI network without compromising on computational resources? A much-needed improvement in theory is required to tackle this issue, which we address by providing an algorithmic framework to compute several network analytics efficiently, without even expressing the entire network in memory.

We achieve this feat by storing only an "implicit representation" of the network based on a minimal set of data that can be used to derive the contents of the entire network. In particular, we show that certain types of large similarity networks can be expressed as matrix products of smaller matrices, through which it is possible to obtain the eigensystem of the corresponding similarity matrix. The eigensystem can be explicitly stored in memory, since the corresponding similarity matrix is symmetric and highly rank deficient. This allows us to then use spectral techniques (techniques based on eigenvectors and eigenvalues) to compute various network analytics. It is very interesting to work in this space as linear algebraic methods for solving combinatorial problems are becoming increasingly popular. For example, with improvements in solving linear systems [102, 69, 70, 66], Kelner et al. gave the fastest known algorithm for computing approximate maximum s-t flows [30, 65]. Harvey [54] showed how to solve matchings and other related problems using algebraic techniques.

The rest of the chapter is organized as follows: We begin with a very informal discussion of some of the common network analytics. We define them formally and discuss them in detail in Chapters 3 and 4. Note that we discuss results from the relevant literature "along the way" instead of in a dedicated Chapter, as they are easier to explain along with appropriate surrounding mathematical context. We then discuss dot product networks and show how a correlation network can be expressed as a dot product network. We briefly show how to store these large correlation networks implicitly and give a general framework for computing analytics on them. We motivate the importance of studying implicit networks with its applications primarily in biomedical fields. We conclude with a summary of our major results and a road-map for the rest of this document.

1.1 Network Analytics

Network analysis is a field of science that analyzes large volumes of data from network structures, and determines useful and interesting properties of the network. For instance, finding the best search results in the Internet consists of identifying the most important or influential nodes in a network, or document classification in the world wide web involves clustering or grouping of well-related documents. The field of network analysis uses many statistical and computational techniques, to analyze many different types of networks, for example those found in bioinformatics, social networks, financial and transportation networks. For general references on network analytics, see [84, 41, 15]. We study the following analytics that are widely used in practice in network analysis.

1. **Centrality:** These are metrics measured at individual nodes that determine "centrality" or importance within the network. For example, a rumor in a social network might spread more rapidly from a source node having high centrality. Most of the definitions provided in this section assume that the network weights are non-negative. The correlation networks we study (see Section 1.2.1) consist of negative edges, and we typically deal with them by transforming the network into another that approximates the absolute values or squares of the edge weights in expectation. We discuss this in detail in Chapter 5. We consider the following commonly used centrality measurements.

   - **Degree Centrality:** This is a quantitative measure that describes the strength of a node's connection to its neighbors. The degree centrality of each node is the sum of the weights of all its incident edges. That is, if W is the weight matrix of the network, then the degree centrality of a node i, also known as *node strength*, is given by

   $$c_{\text{strength}}(i) = s(i) = \sum_j w_{ij}.$$

   Using the adjacency matrix A gives the number of connections to a node i, which is called *node degree*

   $$c_{\text{degree}}(i) = d(i) = \sum_j a_{ij}.$$

We consider complete graphs in this thesis, for which the degree centrality does not provide useful information, since an edge exists between every pair of nodes. One can also look at various statistical properties of the weight distribution of a node's incident edges. We discuss this in detail in Chapter 3.

- **Eigenvector Centrality:** This measure is similar to degree centrality, but also looks at quality of neighbors. The eigenvector centrality [19] gives high values not only to nodes that have a large strength, but also to nodes that are connected to important or influential nodes. The eigenvector centrality of a node i is given by

$$c_{\text{eig}}(i) = \alpha \sum_j w_{ij} c_{\text{eig}}(j)$$

where α is a constant of proportionality that will be related to an eigenvalue of W. For non-negative weighted matrices, the well-known Perron-Frobenius theorem states that the leading eigenvector (eigenvector associated with the largest eigenvalue) is non-negative, and therefore provides a good solution to the vector of centralities[1].

There are two other common variations of the eigenvector centrality that can be considered when every node contains a minimum amount of centrality. If we bias the eigenvector centrality with a positive constant α, and we force that each node's centrality to be at least a positive constant β, then the Katz centrality [64] is given by

$$c_{\text{Katz}}(i) = \alpha \sum_j w_{ij} c_{\text{Katz}}(j) + \beta$$

and the widely-known PageRank centrality [21] is given by

$$c_{\text{PageRank}}(i) = \alpha \sum_j w_{ij} \frac{c_{\text{PageRank}}(j)}{s(j)} + \beta.$$

PageRank centrality normalizes the contribution of a neighbor by its strength. The difference between these measures is in the computation of the leading eigenvector of different matrices. We discuss these in detail in Chapter 3 and show how they all can be computed in our implicit framework.

[1] We assume that the largest eigenvalue is unique, which is typically satisfied under reasonable mathematical constraints.

- **Clustering Coefficient:** This measures the extent to which nodes in a graph cluster together. A node has high clustering coefficient if it is connected to a large number of neighbors that are themselves connected to each other. In unweighted networks, the clustering coefficient of a node is defined as the ratio of the total number of triangles containing the current node over the total number of possible triangles that can be formed with it. In other words, it is the ratio of the number of pairs of neighbors that are connected, to the total number of pairs of neighbors. The clustering coefficient of a node i is given by

$$c_{\text{Watts}}(i) = \frac{\sum_{j \neq q} a_{ij} a_{jq} a_{qi}}{d(i)(d(i)-1)}.$$

  The above definition of clustering coefficient was proposed by Watts and Strogatz [112]. We discuss some generalizations [119, 32] for signed and weighted networks in Chapter 3.

2. **Clustering and Cuts:** Clustering and cuts provide measurements of whole-network connectivity. Clustering algorithms provide reasonable ways to partition the entire network into smaller communities or clusters. Cuts provide a partition of the network into two disjoint pieces or clusters. A global minimum cut is a cut that minimizes the sum of the edge weights between two clusters among all possible such partitions in the network.

   - **Ratio and Normalized Cuts:** The global minimum cut is sometimes very trivial; that is, it puts a single node on one side of the cut and the rest of the graph on the other side. The ratio and normalized cut objectives try to find more balanced splits, to prevent such trivial solutions by minimizing the value of a cut relative to the size (number of nodes in each piece of the partition) or volume (sum of node strengths in each piece of the partition) respectively among all cuts in the graph. For a partition of the network into two disjoint sets of nodes C and $\overline{C}$, the value of the cut is given by $\text{cut}(C, \overline{C}) = \sum_{i \in C, j \in \overline{C}} w_{ij}$, its ratio cut, $\text{rcut}(C, \overline{C}) = \frac{\text{cut}(C, \overline{C})}{|C|} + \frac{\text{cut}(C, \overline{C})}{|\overline{C}|}$, and its normalized cut, $\text{ncut}(C, \overline{C}) = \frac{\text{cut}(C, \overline{C})}{\text{vol}(C)} + \frac{\text{cut}(C, \overline{C})}{\text{vol}(\overline{C})}$, where $|C|$ is the size of C, $\text{vol}(C) = \sum_{i \in C} s(i)$ is the volume of C and $s(i)$ is the node strength defined earlier. The ratio and normalized cut objectives find optimal partitions by minimizing the respective cuts over all possible

partitions in a network:

$$\min_C \left\{\text{rcut}(C,\overline{C})\right\} = \min_C \left\{\frac{\text{cut}(C,\overline{C})}{|C|} + \frac{\text{cut}(C,\overline{C})}{|\overline{C}|}\right\}$$
$$\min_C \left\{\text{ncut}(C,\overline{C})\right\} = \min_C \left\{\frac{\text{cut}(C,\overline{C})}{\text{vol}(C)} + \frac{\text{cut}(C,\overline{C})}{\text{vol}(\overline{C})}\right\}.$$

Finding optimal ratio and normalized cuts is NP-Hard. Therefore, we show well-known spectral relaxations of both the ratio and normalized cut objectives in Chapter 4, and show how these can be computed in our implicit framework, giving an effective heuristic to obtain reasonable solutions.

- **Modularity:** This measures the ability of a network to decompose into smaller clusters. In the unweighted case, it tries to find an optimal partition of the network that maximizes the fraction of edges within clusters minus the expected such fraction in a null model in which edges are placed at random so as to preserve the degree or strength of each node in expectation. The modularity of the network [83] is given by

$$\text{modularity} = \frac{1}{2m}\sum_{ij}\left(a_{ij} - \frac{d(i)d(j)}{2m}[C_i = C_j]\right)$$

where m is the number of edges, C_i is the cluster that node i belongs to, and $[C_i = C_j]$ is an "indicator" function that evaluates to 1 if the predicate $C_i = C_j$ is true, 0 otherwise. This can be easily generalized to weighted networks by using W in the definition above:

$$\text{modularity} = \frac{1}{2\text{sum}(W)}\sum_{ij}\left(w_{ij} - \frac{s(i)s(j)}{2\text{sum}(W)}[C_i = C_j]\right)$$

where $\text{sum}(W) = \sum_{i,j} w_{ij}$ is the sum of the weights of all the edges in the network. A partition of the network that has positive modularity is a prospective choice for clustering into two pieces, since the number of edges within clusters exceeds the expected number of edges just on the basis of random chance. Negative values of modularity for every partition indicates the inability of the network to break down into any number of communities. Modularity can also be used as a measure of "connectedness" of the network, that is, how well the nodes in the network are connected with each other. Finding an optimal partition that maximizes modularity is NP-Hard. We therefore discuss heuristics based

on well-known spectral relaxations in detail in Chapter 4.

- **Global Clustering Coefficient:** This also measures the ability of a network to decompose into smaller clusters. It is often defined as the average of the local clustering coefficients of all the nodes, though other types of aggregation, such as taking the mediant may also be used. The aggregation is normalized so that the value of global clustering coefficient lies in the range $[0, 1]$. In unweighted networks, a value of 1 indicates that the network is a complete graph; that is, every node is connected to every other node. Using the mediant of local clustering coefficients, the global clustering coefficient is defined as the ratio of three times the number of triangles in a network to the total number of pairs of nodes adjacent to each other

$$\text{gcc} = \frac{3 \sum_{i \neq j \neq q} a_{ij} a_{jq} a_{qi}}{\sum_i \binom{d(i)}{2}}.$$

  Just like local clustering coefficient, this can also be generalized to weighted and signed networks and is discussed in Chapter 3.

- **Eigen Gap:** This is related to a well-studied measure called conductance, which measures how closely nodes are connected in a network, and is also related to the ratio and normalized cut objectives. The conductance ϕ of a cut is the ratio of the value of the cut over the minimum volume of the clusters formed between the cut. It is defined as

$$\phi(C, \overline{C}) = \frac{\text{cut}(C, \overline{C})}{\min\left\{\text{vol}(C), \text{vol}(\overline{C})\right\}}.$$

  The conductance of a cut gives the probability of a random walk to transition from the small side of the cut to the other side in a single step. The conductance of the entire graph ϕ_G is the minimum conductance over all possible cuts;

$$\phi_G = \min_C \left\{\phi(C, \overline{C})\right\}$$

  and measures the tendency of a random walk to stay within one side of the cut. In other words, a graph has high conductance if a random walk tends to jump very often

between clusters, and small conductance if it is more likely to stay trapped within some cluster. The walk matrix of a network provides a probability distribution of taking a single step in a random walk, and can be derived from the weight matrix as WD^{-1}, where D is a diagonal matrix with $d_{ii} = s(i)$. Then the eigen gap is defined as the difference between the largest and the second largest eigenvalues in magnitude of the walk matrix. Computing ϕ_G is NP-Hard, and the eigen gap provides a suitable approximation (see Section 4.1.4). The eigen gap also relates to the mixing time of a Markov chain, which refers to the rate at which a Markov chain converges to its steady state distribution. The larger the value of the eigen gap, the faster the convergence.

1.2 Dot Product Networks

A *similarity network* $G = (V, E, f)$ is a graph with node set V, edge set E, and symmetric similarity function $f\colon V \times V \to \mathrm{R}$. Let the number of nodes $n = |V|$. Each edge $e_{ij} \in E$ also has a weight w_{ij} that is given by $f(i, j)$. The $n \times n$ matrix W contains all the pairwise edge weights.

The similarity function often operates on feature vectors associated with each node in the network. For length-k feature vectors b_i and b_j associated with nodes i and j respectively, the dot product $b_i \cdot b_j$, defined as $\sum_{l=1}^{k} b_i(l)b_j(l)$, where $b_i(l)$ is the l^{th} entry of b_i, offers a measure of similarity. A *dot product network* is a similarity network in which f is a dot product of the corresponding feature vectors. Note that a dot product network is symmetric, i.e. $f(i, j) = f(j, i)$. Also note that a dot product can be negative. If all edge weights in W are non-negative, then we say it is a *non-negative dot product network*. Dot product networks are common in many settings. For example, in a social network, a feature vector of an individual might represent the extent of his or her opinions on k issues, and the dot product gives the degree to which two individuals agree.

If B is a matrix with feature vectors along its rows (i.e., row i is the feature vector b_i of i), then W can be expressed as a simple matrix product $W = BB^T$, since $w_{ij} = b_i \cdot b_j$. For a large network (e.g., with $n \approx 100,000$ and $k \approx 1000$) the matrix W has size n^2 and can be computationally stressful to store in memory explicitly. It also takes $\Theta(n^2k)$ time to compute all the entries of W in the most straightforward fashion. However, matrix B with dimensions $n \times k$ is a substantially smaller matrix that implicitly encodes all the information of W.

Dot product networks were studied as a model for social networks [97, 117, 118, 86, 59].

Bailey [13] showed how dot product networks can be used in several applications in ecology. Several graph-theoretic results on structural properties of such networks were obtained by [92, 93, 42, 63, 74, 11].

1.2.1 Correlation Networks

Correlations measure the extent to which two feature vectors fluctuate together. In this section, we discuss correlation networks and show how to express them as dot product networks.

Statistical covariance measures the linear dependence of two random variables. For random variables I and J that represent an underlying probability distribution, the covariance between I and J is given as

$$\text{cov}(I, J) = \text{E}[(I - \text{E}[I])(J - \text{E}[J])]$$

where $\text{E}[I]$ is the expected value of I. Sometimes we acquire only k independently drawn samples or observations of the probability distribution represented by the random variable I. If the feature vector b_i contains these k samples, then the sample mean is given by

$$\overline{b_i} = \frac{1}{k} \sum_{l=1}^{k} b_i(l)$$

and the sample covariance[2]

$$s_{ij} = \frac{1}{(k-1)} \sum_{l=1}^{k} (b_i(l) - \overline{b_i})(b_j(l) - \overline{b_j}).$$

If both b_i and b_j are normalized so that they are centered to mean zero, then

$$s_{ij} = \frac{b_i \cdot b_j}{(k-1)}.$$

If matrix B contains all the rows normalized to mean zero, then the covariance matrix $S = \dfrac{BB^T}{(k-1)}$ becomes a dot product network.

Positive values of covariance indicate that the two vectors peak and dip at about the same

[2]The normalizing factor $k-1$, sometimes known as Bessel's correction, is used in the denominator instead of k as this gives an unbiased estimate of the covariance when the actual mean is unknown.

time, and negative values indicate that while one peaks, the other dips. The magnitude of covariance gives a reasonable measure of the degree of this linear relationship between the two variables. As this magnitude approaches zero, the random variables tend to be linearly independent, but could still be dependent through a non-linear process.

Note that the unit of covariance is the product of the units of I and J. This is a bit cumbersome in comparing covariances that have different units. Also, the range of covariance is in $(-\infty, +\infty)$. To reduce the range of these values and to make it unit-less for the purposes of comparison, we define Pearson's correlation as normalized covariance

$$\text{corr}(I, J) = \frac{\text{cov}(I, J)}{\sigma_i \sigma_j}.$$

Here σ_i is the standard deviation of I, defined as $\sigma_i = \sqrt{\text{E}[(I - \text{E}[I])^2]}$. The sample correlation between I and J provides a good estimate of the correlation between I and J and is given as

$$w_{ij} = \frac{s_{ij}}{\sigma_i \sigma_j}$$

where σ_i is the sample standard deviation of I, defined as $\sqrt{\frac{\sum_{l=1}^{k} (b_i(l) - \overline{b_i})^2}{k-1}}$. If b_i is normalized to mean zero, then $\sigma_i = \sqrt{\frac{\sum_{l=1}^{k} b_i(l)^2}{k-1}} = \frac{|b_i|}{\sqrt{k-1}}$, where $|b_i|$ is the Euclidean length of the vector b_i. The sample correlation then becomes

$$\begin{aligned} w_{ij} &= \frac{s_{ij}}{|b_i||b_j|}(k-1) \\ &= \frac{b_i \cdot b_j}{|b_i||b_j|}. \end{aligned}$$

If vectors b_i and b_j are normalized to mean zero and unit variance, then $w_{ij} = b_i \cdot b_j$. If the matrix B contains all rows normalized to mean zero and unit variance (e.g., $\overline{b_i} = 0$, $|b_i| = 1$), then the correlation matrix is simply given by $W = BB^T$. A *correlation network* is then defined as a dot-product network, where $f(i, j)$ is the sample correlation between nodes i and j. Pearson's correlation is unit-less and is in the range $[-1, +1]$. The magnitude and sign of the correlation reflects the behavior of covariance.

Note that the correlation network is a *signed* weighted network. Most of the current algorithms to compute analytics are defined mainly for non-negative networks. In situations where only non-negative weights are allowed, many approaches have been used to make correlation networks non-negative, including squaring or taking absolute values of edge weights or zeroing negative edge weights. While the latter approach completely disregards negative correlations, the former two approaches treat both negative and positive correlations equally. This captures trends between feature vectors that are slightly out of phase with each other. We show in Chapter 5 how to transform a correlation network into another dot product network that preserves the absolute values or squares of corresponding correlations in expectation. This non-negative dot product network can then be used in our implicit computations.

Another approach is to use $1 + \text{corr}(\cdot)$. This approach simply moves the range $[-1, +1]$ to the range $[0, 2]$, and therefore treats negative correlations as less important. This can be easily achieved in our dot product framework by appending a 1 to the end of each feature vector.

Analysis of networks is also studied using *partial correlations*, which measure similarities between two variables while controlling for a third variable. Sample partial correlations find correlations between two vectors after first subtracting out projections of other vectors. It is also interesting to look at correlations in the frequency domain by considering coherence between two time series signals. Coherence measures the consistency in phase difference at a given frequency between the corresponding time-series signals. We discuss these similarity measurements in detail in later chapters.

1.3 Applications

Dot product networks are used extensively in data mining and machine learning. Such analysis has applications in machine learning classification, learning, prediction and recommendation. Dot product networks also occur naturally in many social and biological contexts. Below, we discuss briefly their applications in computational neuroscience and genomics, and also the general applicability of converting many other similarity networks into dot product networks through the technique of locality sensitive hashing.

1.3.1 Computational Neuroscience

An fMRI experiment studies neural activations in the brain, through excitation of the response of a subject to several types of tests, such as touching of the arm, flashing of lights or even under a steady resting state. The neural activations correspond to increase in blood flow though that region. Each region of the brain emits a time series of the amount of oxygen consumption during that period. By taking correlations of the time series data, we obtain a functional connectivity network.

At high resolutions where each region is a voxel of size 3 x 3mm, there could be of the order of $100,000$ nodes in an entire brain. The entire weighted correlation matrix would be extremely large to store in memory explicitly, and very expensive to compute. The current practice is to analyze (using either correlations or coherence) only small regions of interest, and therefore improving the efficiency of computing various analytics is required. Computing analytics like centrality and clustering measurements on functional brain networks of patients that have autism and comparing them against a control group of non-autistic patients could provide insight into regions of the brain that are highly active for autistic patients. This could lead to a better understanding and treatment of autistic patients.

1.3.2 Genomics

Correlation networks are widely used in computing similarities between genes in a gene-gene interaction network. A high-throughput experiment over a gene micro-array can measure the expressions of n genes in a cell. The number of human genes n in a cell is approximately $21,000$. If k observations are made on the genes over a period of time, or if k different experiments are made on the cell (say, subjecting to different environmental factors or experimental conditions), then each gene can be represented by a length-k feature vector of expressions. The correlation between two genes now gives a similarity measurement, sometimes called the co-expression relationship. Studying network analytics in such gene-gene interaction networks could lead to useful biological insights on how gene function changes with disease.

We can also measure the genotype of k individuals with a specific disease, and compare them against k' healthy individuals who act as a control group, to determine the genomic origin of the disease. Measurements are made at n different SNPs (single nucleotide polymorphisms), which

are specific sites in the genome where there is substantial genetic variation (n can be in the order of millions). A SNP network is essentially a dot product network formed between n nodes that represent different SNPs and length k feature vectors of measurements of those SNPs on k different individuals. Analytic properties that change between a SNP network computed from the cases and a network computed from the controls might illustrate SNPs associated with the disease in question.

1.3.3 Locality Sensitive Hashing

A locality-sensitive hash function hashes input elements into "buckets" (often just the set of two values $\{-1, +1\}$ or $\{0, 1\}$) such that similar elements often end up in the same bucket. It is usually used to reduce the dimensionality of high-dimensional data, and has applications in various clustering algorithms, data mining, machine learning, and audio and video fingerprinting.

Suppose objects such as strings, audio, video, etc. admit a locality-sensitive hash function. That is, it is possible to hash objects to elements to $\{-1, +1\}$, such that $\Pr[h(i) = h(j)] \approx w_{ij}$, where $w_{ij} \in [0, 1]$ is the similarity between objects i and j, and h is a randomly-parameterized function. By selecting a set of locality-sensitive hash functions $h_1 \dots h_k$, we can form a new feature vector $b_i = [h_1(i) \dots h_k(i)]$, such that $\mathrm{E}[b_i \cdot b_j] = 2kw_{ij} - k$. Thus, domains supporting locality-sensitive hash functions can be easily converted into the familiar framework of a dot product network. In Chapter 5, we use locality-sensitive hashing to transform any negative correlation network into another dot product network that approximately preserves the absolute values or squares of the corresponding correlations in expectation. This allows us to compute several analytics that are solely defined for non-negative networks in an implicit fashion.

1.3.4 Other Applications

Analysis of dot product networks is also used in other engineering and science domains. For example, in civil engineering, vibrational data emits a time series signal which can be used to analyze and assess earthquakes. In chemistry, oscillation patterns of chemical molecules and atoms can be used in the study and analysis of the phase consistency between them by looking at a coherence network.

Another interesting application is to analyze large time-series data at different window intervals. Given a long time series or feature vector data, we have n different length-k snippets or

windows of the time series data. If each of these snippets form the rows of a matrix B, then the correlation network $W = BB^T$ gives the similarity between two different time series windows. Analyzing such networks may be useful in identifying motifs that show up as clusters in the correlation network.

1.4 Results

This thesis makes the following contributions.

1. Large correlation-type networks can be implicitly represented as dot product networks $W = BB^T$, where B is a data matrix of size $n \times k$ with $n > k$. Typically n and k are in the order of millions and thousands respectively. They are symmetric and highly rank deficient, and therefore can be expressed using their singular value decomposition $W = \sum_{i}^{k} \lambda_i u_i u_i^T$, where the λ_is are the eigenvalues of W, and the u_is are the corresponding eigenvectors of W. The entire spectrum can be computed in only $O(nk^{\omega-1})$ time, where ω is the exponent of matrix multiplication.

2. The power method can be used to compute the top few eigenvalues or eigenvectors of W and other matrices that can be derived from the spectrum of W. This allows us to compute various analytics either precisely or approximately, like eigenvector, Katz and PageRank centralities, clustering coefficient, ratio and normalized cuts, modularity and eigengap. For correlation networks that contain negative edges, we give a generalization of the degree centrality that can be computed implicitly. Also, we show how principal component analysis and its mathematical equivalent, correlation clustering can both be computed implicitly.

3. We show how other similarity measurements can be expressed implicitly. Specifically, the optimal value of a shrinkage estimator γ for several popular targets T can be computed in only $O(nk^{\omega-1})$ time. This can then be used to obtain the spectrum of the biased correlation matrix $W^* = \gamma T + (1-\gamma)W$. Partial correlations and partial coherence can also be expressed as a sum of a diagonal matrix and a matrix for which the spectrum can be computed in $O(nk^{\omega-1})$ time. The matrix of all pairwise coherence for a specific frequency can also be expressed as a dot product network. The power method can then be used to compute various analytics.

4. Using mathematical manipulation similar to that found in a common "kernel trick", we show how to obtain a non-negative dot product network in $O(k^2)$ dimensions that preserves the squares of correlations exactly. This new network can be used to compute various analytics which are exclusively defined for non-negative networks. This also allows us to express a matrix of pairwise magnitude squared coherence implicitly.

5. By reducing dimensionality using random projections, we show how to use locality sensitive hashing to not only reduce dimensionality, but also to produce a non-negative dot product network that approximately preserves squares of dot products in expectation. The absolute values of the dot products can also be similarly preserved. The error between the actual values and the expected values after the mapping can be reduced by using more dimensions.

1.5 Roadmap

We provide all the necessary mathematical foundations in Chapter 2. Specifically, we show how to multiply a dot product network implicitly with another vector or matrix. We show how to obtain the singular value decomposition of the weight matrix in $O(nk^{\omega-1})$ time. We discuss the power method for computing the top few eigenvalues and eigenvectors in an implicit fashion. We also discuss how to express both the biased correlation estimate using a shrinkage parameter, and partial correlations as dot product networks. We conclude Chapter 2 with a discussion of dimensionality reduction via random projections. We discuss various centrality measures for both negative and non-negative dot product networks in Chapter 3. We discuss degree centrality, eigenvector centrality and clustering coefficients for non-negative networks and some generalizations for negative networks. In Chapter 4, we discuss various clustering and cut algorithms that can be computed implicitly. Specifically, we discuss ratio and normalized cuts, and modularity for non-negative dot product networks. We also discuss two mathematically equivalent measures – correlation clustering and principal component analysis in our implicit setting. In Chapter 5, we show how to obtain a non-negative dot product network that preserves squares of correlations exactly. We also show how to get a mapping using locality sensitive hash functions that allows us to get another non-negative dot product network in fewer dimensions that approximately preserves squares of correlations in expectation. We also discuss how a coherence or partial coherence network can be expressed as a dot product network, which provides an alternative way to represent edge weights. We finally

provide our conclusions and discuss future work in Chapter 6.

Chapter 2

Preliminaries

2.1 Notation

As is the general convention, small letters refer to column vectors, and capital letters to matrices. We use B to refer to a data matrix where each row b_i is a feature vector (expressed as a column vector if referred to individually), W the weight matrix of a network and A the adjacency matrix. We however, for the sake of presentation, reserve the notation u_i or v_i as column vectors of matrices U and V that are the singular value decompositions of $W = BB^T$ and B^TB respectively. The element at the i^{th} row and j^{th} column of B is written b_{ij} or B_{ij}. The transpose of B is written B^T and the inverse B^{-1}. B is an orthonormal matrix if $BB^T = B^TB = I$, the identity matrix. B^k denotes multiplying a square matrix B with itself k times, and $B^{(k)}$ denotes the element-wise power operation, raising each element in B to the power k.

If the rows of B have mean zero and unit variance ($\overline{b_i} = 0$, $|b_i| = 1$), then $W = BB^T$ is a correlation network. We use w_{ij}^k to refer to the k^{th} power of w_{ij}, and $(w^k)_{ij}$ or W_{ij}^k to the element in row i and column j of w^k. Note that a correlation network is likely to have negative edge weights.

2.2 Block Multiplication

Matrix multiplication is a basic building block in our algorithms. Multiplying two $k \times k$ matrices takes $O(k^\omega)$ time, where ω is defined as the best exponent for matrix multiplication. The most recent value $\omega = 2.3728639$, is due to Le Gall [44] and any improvement in this value also

improves the efficiency of our algorithms. The result of the matrix product $B^T B$ is a $k \times k$ matrix that takes $O(nk^2)$ time to compute in a straightforward fashion. However, we can improve the speed by performing "block" multiplication: divide B into $\frac{n}{k}$ blocks $B_1 \ldots B_{\frac{n}{k}}$, each of size $k \times k$. We can compute $B_x^T B_x$ for any block $x = 1 \ldots \frac{n}{k}$ in $O(k^\omega)$ time, and then add all the results together. The total time for computing $B^T B$ is then $O(\frac{n}{k} k^\omega) = O(nk^{\omega-1})$.

Using a similar approach, the multiplication of a large $n \times n$ matrix W with a smaller $n \times k$ matrix B can be computed by dividing the rows of W into $k \times n$ blocks, multiplying each block with B in $O(nk^{\omega-1})$ time, to each form a $k \times k$ block of the resulting matrix. Since there are $\frac{n}{k}$ such blocks, the total time is $O(n^2 k^{\omega-2})$.

According to well-known block decomposition methods, solving an $n \times n$ linear system or inverting an $n \times n$ matrix both can be done in $O(n^\omega)$ time.

2.2.1 Multiplying by a vector

It is easy to multiply the weighted correlation matrix $W = BB^T$ by a length n vector x, by multiplying the two $n \times k$ matrices from the inside out. That is, we parenthesize the expression $Wx = (BB^T)x = (B(B^T x))$ so as to only multiply a smaller $n \times k$ matrix by a vector, instead of computing the large BB^T matrix. The total time to compute Wx is therefore only $O(nk)$.

2.3 The Singular Value Decomposition

An eigenvector v of an $n \times n$ matrix W satisfies the equation $Wv = \lambda v$, where λ is called its corresponding eigenvalue. The set of all eigenvectors along with their eigenvalues is called an eigensystem. Although we rarely use this approach, it is worth noting that the eigenvalues can be computed by finding roots of the characteristic polynomial $\det(W - \lambda I) = 0$. The eigenvectors would then be the solutions to the characteristic equation $(W - \lambda I)x = 0$. Though the problem of polynomial root finding cannot be solved within any fixed running time as a function of n, modern eigensystem solvers use sophisticated numerical techniques to iteratively converge to an approximation, and typically compute the complete set of eigenvalues and eigenvectors in $O(n^3)$ time. For more information on linear system solvers, refer to textbooks on linear algebra and numerical analysis [105, 37].

The singular value decomposition (SVD) allows us to represent an $n \times k$ matrix B as a

product of three other matrices $U\Sigma V^T$, where each column u_i of the $n \times n$ matrix U is an eigenvector of $W = BB^T$, each column v_i of the $k \times k$ matrix V is an eigenvector of $B^T B$, and Σ is a $n \times k$ diagonal matrix with singular values $\sigma_1 \ldots \sigma_k$. Since BB^T and $B^T B$ are real symmetric matrices, the u_is and the v_is are all real and orthonormal, and are called the left and right singular vectors of B respectively.

We often write a matrix as a sum of outer products of rank-1 matrices using its eigendecomposition:

$$B = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

and

$$W = BB^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V \Sigma^T U^T = U\Sigma I \Sigma U^T = U\Sigma^2 U^T = \sum_{i=1}^{k} \sigma_i^2 u_i u_i^T.$$

Similarly

$$B^T B = \sum_{i=1}^{k} \sigma_i^2 v_i v_i^T.$$

Note that BB^T and $B^T B$ share the same eigenvalues $\sigma_1^2 \ldots \sigma_k^2$. Since $O(n^3)$ is not a reasonable time to compute the eigensystem of a large $n \times n$ matrix $W = BB^T$, we compute the SVD of W as follows:

1. Compute $B^T B$ in $O(nk^{\omega-1})$ time.
2. Compute the eigensystem of $B^T B$ in $O(k^3)$ time, which gives both Σ and V.
3. From the SVD, we know that $BV = U\Sigma$, therefore we can compute BV in $O(nk^{\omega-1})$ time, and then rescale its columns according to Σ^{-1} to obtain $U = BV\Sigma^{-1}$.

As long as $n > k^{4-\omega}$ (a reasonable assumption), the entire process is not bottlenecked by the time to compute the eigensystem of $B^T B$ in Step 2, and therefore takes $O(nk^{\omega-1})$ total time and $O(nk)$ space.

Note that any individual entry of $W = BB^T$ can still be computed by taking the dot product of corresponding feature vectors $b_i \cdot b_j$. Another way to compute an entry of W is using

its eigendecomposition $\sum_{i=1}^{k} \sigma_i^2 u_i u_i^T$ (each column u_i is a singular vector of B, see above). The entry w_{xy} in W is $\left(\sum_{i=1}^{k} \sigma_i^2 u_i u_i^T\right)_{xy} = \sum_{i=1}^{k} \sigma_i^2 u_i(x) u_i^T(y)$. It takes $O(k)$ time to compute an individual entry of W in both cases. The eigendecomposition technique to compute individual entries becomes particularly useful when computing entries of W^n, for some power n, or W^{-1}. Since we spend at least $O(k^3)$ time to compute the SVD of W, we do not compromise the running time of our algorithms when we explicitly compute at most $O(k^2)$ of the entries in W or any other matrix that can be derived from the eigendecomposition of W.

It is also easy to multiply W by a vector x using the eigendecomposition, since $W = \left(\sum_{i=1}^{k} \sigma_i^2 u_i u_i^T\right) x = \sum_{i=1}^{k} \sigma_i^2 \left(u_i \left(u_i^T x\right)\right)$. Each parenthesized dot product takes $O(n)$ time and therefore the total time to compute Wx is $O(nk)$ time, matching the time via parenthezation $B(B^T x)$. The matrix product WX, where X is a $n \times k$ matrix can be computed in $O(nk^2)$ time, since each column of the resultant matrix takes $O(nk)$ time to compute.

We now list some of the important properties of an eigensystem of a $n \times n$ real and symmetric matrix $W = BB^T$ without proof. The proof can be found in any introductory textbook on linear algebra [105, 37].

1. W is rank deficient and therefore singular. Its rank is k and it contains only k non-zero eigenvalues.

2. The eigenvectors of W are linearly independent.

3. If λ is an eigenvalue of W, then $\alpha\lambda$ is an eigenvalue of αW.

4. If λ is an eigenvalue of W, then $\lambda + \alpha$ is the eigenvalue of $W + \alpha I$.

5. If λ is an eigenvalue of W, then λ^α is an eigenvalue of W^α.

6. If λ is an eigenvalue of W, then λ^{-1} is an eigenvalue of W^{-1}.

7. The trace of W, written trace(W), defined as the sum of its main diagonal, is equal to the sum of the eigenvalues.

8. The product of the eigenvalues is equal to the determinant of W, which is 0 since W is singular.

9. The principal eigenvector of a matrix is the eigenvector associated with the eigenvalue with the largest magnitude. This eigenvalue is sometimes called the dominant or principal eigenvalue.

10. For all unit-length vectors x, the eigenvector associated with the largest eigenvalue is the solution to quadratic form $\max_{|x|=1} \left\{ \sum_{i,j} w_{ij} x_i x_j \right\} = \max_{|x|=1} \left\{ x^T W x \right\}$. Similarly the eigenvector associated with the smallest eigenvalue is the solution to $\min_{|x|=1} \left\{ x^T W x \right\}$.

2.3.1 The Power Method

The power method [80] finds an approximation of the the dominant eigenvalue and its corresponding eigenvector. Though the SVD algorithm above computes all eigenvalues and eigenvectors, the power method is a very common technique in linear algebra, that is widely used in many applications such as Google's Pagerank [89], Twitter's "who to follow" system [51], and many other advanced iterative linear algebra algorithms, such as inverse iteration [90] and Arnoldi iteration [7].

We start with an initial guess v_0 for the eigenvector, and in every iteration improve it by computing

$$v_{i+1} = \frac{W v_i}{|W v_i|}.$$

The above process converges to an eigenvector associated with the dominant (largest magnitude) eigenvalue if v_o contains a non-zero component in the direction of the eigenvector associated with the dominant eigenvalue, which is almost surely true if v_0 is chosen randomly.

Since $W = BB^T$, the above multiplication Wv_i can be easily computed by parenthezation $B(B^T v_i))$. This allows us to run the power method in an implicit fashion, even without writing down all the entries of W.

To compute the eigenvector associated with the second dominant eigenvalue, the power method can be continued with the projection of the principal eigenvector subtracted from the computation above. The new computation converges to the eigenvector associated with the second dominant value. This process can be further continued to find all eigenvectors, by simply subtracting the projections of each successively dominant eigenvector found. However, it is not good at finding higher-order eigenvectors due to numerical instability.

2.4 Locality Sensitive Hashing

Locality sensitive hashing (LSH) is a technique that hashes similar objects together. This technique was proposed by Indyk and Motwani [57] and has become a basic primitive in large scale data processing. There are many applications of this technique in databases [46], information retrieval, pattern recognition, dynamic closest pairs [57], fast clustering [55], finding approximate neighbors [57, 6] and computational biology [23, 24]. The general idea is to hash objects using several hash functions, such that there are likely to be collisions between objects that are highly similar. Formally, a family of hash functions H is a locality sensitive hash function if $\Pr_{h\in \mathrm{H}}[h(U) = h(V)] \approx f(u, v)$ for objects U and V and for the similarity function f. The codomain of the hash functions is often just $\{0, 1\}$. A similar definition found in [57, 46] is that a (r, R, p, P)-LSH for a similarity function f is a probability distribution over a set H of hash functions such that

- $f(u, v) \geq R \Rightarrow \Pr_{h\in \mathrm{H}}[h(u) = h(v)] \geq P$
- $f(u, v) < r \Rightarrow \Pr_{h\in \mathrm{H}}[h(u) = h(v)] < p$

To illustrate this idea, consider points in Hamming space $\{0, 1\}^d$. The similarity function f is defined in terms of the Hamming distance, which is the fraction of positions in which the two points differ. Formally, $f(u, v) = 1 - \frac{1}{d}\sum_{i=1}^{d} |(u(i) - v(i))|$. A family of hash functions H can be selected, where $h(u)$ is a projection of u along one of the coordinate axis, i.e., $h(u) = u(i)$, for a randomly selected dimension $i \in \{1, \ldots, d\}$. Let D be a uniform random variable associated with choosing a dimension, so $D = i$ is the event that the random variable takes the value i, or the event that we pick the i^{th} dimension. Then

$$\begin{aligned}
\Pr_{h\in \mathrm{H}}[h(u) \neq h(v)] &= \sum_{i=1}^{d} \Pr[h_i(u) \neq h_i(v) | D = i] \Pr[D = i] \\
&= \sum_{i=1}^{d} \Pr[h_i(u) \neq h_i(v) | D = i] \frac{1}{d} \\
&= \frac{1}{d} \sum_{i=1}^{d} |u(i) - v(i)| \\
&= 1 - f(u, v).
\end{aligned}$$

As another example, consider the Jaccard similarity (see [22, 23]) for computing the simi-

larity between sets. The Jaccard similarity between two sets $U, V \subseteq S$ is defined as

$$J(U,V) = \frac{|U \cap V|}{|U \cup V|}.$$

A family of hash function can be selected by taking the minimum of a set from a random permutation of all the elements in S, i.e., $h_\pi(U) = \{\min \pi(u) \mid u \in U\}$. It can be easily seen that $\Pr[h(U) = h(V)]$ is the Jaccard similarity, since any element in $U \cup V$ has an equally likely chance to be the minimum, and $h_\pi(u) = h_\pi(v)$ only for the elements in $U \cap V$.

We will revisit this topic in Chapter 5 when we design such functions to obtain a non-negative dot product network that approximates the absolute values or squares of the correlations in expectation.

2.5 Other Edge Weight Representations

In this section, we look at shrinkage estimators, which offer better estimates to the true values of covariance and correlation between random variables. We also look at partial correlation between random variables as a means of expressing relationships between nodes, and show how we can express them implicitly as dot product networks. Later, in Chapter 5, we show how to express coherence as a dot product network.

2.5.1 Shrinkage Estimators for Correlations

In Chapter 1 we saw that Pearson's correlation measures linear relationships between random variables. As mentioned earlier, we acquire only k independent samples of the probability distribution represented by a random variable. With only these k samples, we need to estimate mean and standard deviations that are close to their true values, and hence estimate correlations between random variables close to their true values. Shrinkage estimators offer a way to parameterize the data in order to reduce the mean squared error of the estimated values.

The sample covariance or correlation matrix offers very little structure for "large n and small k data". It is for one thing highly rank deficient, and though it produces better estimates as k tends to infinity, it is unsuitable for small values of k (see [96] for further elaboration). A solution to this called shrinkage [103] involves taking a weighted average of a high-structured and

less variable estimate called a target, and the raw sample covariance or correlation matrix. For a sample covariance or correlation matrix W and a target matrix T, the shrinkage estimator is given by

$$W^* = \gamma T + (1 - \gamma)W$$

where $\gamma \in [0, 1]$ is called a shrinkage parameter and allows us to control the amount of structure we introduce. When $\gamma = 1$, the shrinkage estimator equals the target matrix, and when $\gamma = 0$, it equals the unrestricted sample covariance matrix. For $\gamma > 0$, the resulting estimate is typically invertible, well-conditioned (the difference between the largest and smallest eigenvalue is small) and positive-definite (all eigenvalues are positive).

A big advantage of the shrinkage method is that it can be completely data-driven. Ledoit and Wolf [71] showed how to analytically obtain the optimal value of γ by minimizing a risk function giving squared estimation error. Schäffer and Strimmer [96] provide optimal values of γ for a set of commonly used targets. For the sample correlation matrix $W = BB^T$ and sample covariance matrix $S = \frac{BB^T}{(k-1)}$, and for the use of the identity matrix as a target, they give the optimal value of γ to be

$$\gamma = \frac{\sum_{i \neq j} z_{ij} + \sum_i z_{ii}}{\sum_{i \neq j} s_{ij}^2 + \sum_i (s_{ii} - 1)^2} \tag{2.1}$$

where

$$z_{ij} = \frac{k}{(k-1)^3} \sum_{l=1}^{k} ((b_{il} b_{jl})^2 + w_{ij}^2 - 2 b_{il} b_{jl} w_{ij}).$$

The last two terms in the sum give

$$\begin{aligned}\sum_{l=1}^{k}(w_{ij}^2 - 2b_{il}b_{jl}w_{ij}) &= kw_{ij}^2 - \sum_{l=1}^{k} 2b_{il}b_{jl}w_{ij} \\ &= kw_{ij}^2 - 2w_{ij}\sum_{l=1}^{k} b_{il}b_{jl} \\ &= kw_{ij}^2 - 2w_{ij}(b_i \cdot b_j) \\ &= kw_{ij}^2 - 2w_{ij}^2 \\ &= (k-2)w_{ij}^2.\end{aligned}$$

The first term can be expressed as another matrix $W' = B^{(2)}B^{(2)T}$, since

$$\begin{aligned}\sum_{l=1}^{k}(b_{il}b_{jl})^2 &= \sum_{l=1}^{k} b_{il}^2 b_{jl}^2 \\ &= B_i^{(2)} \cdot B_j^{(2)} \\ &= w'_{ij}.\end{aligned}$$

Hence we get

$$z_{ij} = \frac{k}{(k-1)^3}\left(w'_{ij} + (k-2)w_{ij}^2\right).$$

To compute the optimal γ, the numerator of Equation 2.1 can be simplified to

$$\begin{aligned}\sum_{i,j} z_{ij} &= \frac{k}{(k-1)^3}\sum_{i,j}\left(w'_{ij} + (k-2)w_{ij}^2\right) \\ &= \frac{k}{(k-1)^3}\left(\sum_{i,j} w'_{ij}\right) + \frac{k(k-2)}{(k-1)^3}\left(\sum_{i,j} w_{ij}^2\right).\end{aligned}$$

Let $c_1 = \dfrac{k}{(k-1)^3}$ and $c_2 = c_1(k-2)$, the above equation further simplifies to

$$\sum_{i,j} z_{ij} = c_1\sum_{i,j} w'_{ij} + c_2\sum_{i,j} w_{ij}^2.$$

The term $\sum_{i,j} w_{ij}^2$ can be computed from W^2, where $W_{ii}^2 = \sum_j w_{ij}^2$. The trace of W^2 gives the

required sum, and this can be easily computed implicitly by noting that the trace of a matrix is the sum of its eigenvalues, so $\sum_{i,j} w_{ij}^2 = \text{trace}(W^2) = \sum_i \lambda_i^2$, where λ is an eigenvalue of W. The term $\sum_{i,j} w'_{ij}$ can be computed from $\mathbf{1}^T B^{(2)} B^{(2)T} \mathbf{1}$ (written sum(W')) easily by parenthezation $\mathbf{1}^T (B^{(2)} (B^{(2)T} \mathbf{1}))$ in only $O(nk)$ time.

The denominator in the computation of the optimal shrinkage γ can also be expressed implicitly:

$$\begin{aligned}
\sum_{i \neq j} s_{ij}^2 + \sum_i (s_{ii} - 1)^2 &= \sum_{i \neq j} s_{ij}^2 + \sum_i s_{ii}^2 - 2 \sum_i s_{ii} + \sum_i 1 \\
&= \sum_{i,j} s_{ij}^2 - 2 \sum_i s_{ii} + n \\
&= \text{trace}(S^2) - 2\text{trace}(S) + n
\end{aligned}$$

Note that trace(S^2) and trace(S) can be easily computed from the eigenvalues of W, since $S = (k-1)W$. Therefore the optimal value of γ can be computed implicitly as:

$$\gamma = \frac{c_1 \text{sum}(W^{(2)}) + c_2 \text{trace}(W^2)}{\text{trace}(S^2) - 2\text{trace}(S) + n}$$

In Table 2.1 we have shown how to implicitly compute the value of γ for four of the most widely used targets as described in [96]. Target A is one of the most widely used targets, which is just the identity matrix. The spectrum of $W^* = \gamma I + (1-\gamma)W$ using Target A can be easily computed from the spectrum of W, which contains the same set of eigenvectors with eigenvalues appropriately shifted. For these reasons, we typically stick with Target A, since we can implicitly compute and store the spectrum of the resulting biased correlation estimate W^*. In most cases (especially with partial correlations), the spectrum is all we need to run the power method to compute the top few eigenvectors and eigenvalues. Other targets are not as suited to our implicit framework.

2.5.2 Partial Correlation

Sometimes it is necessary to measure the dependence of two random variables by removing the association of a set of controlling variables. Standard correlation does not capture this and gives high correlation between two random variables that are simultaneously influenced by a third variable. A well-known statistical quantity called the partial correlation measures correlations

Targets	Optimal γ	Implicit Equation for γ
Target A $T_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$	$\dfrac{\sum_{i \neq j} z_{ij} + \sum_{i=j} z_{ii}}{\sum_{i \neq j} s_{ij}^2 + \sum_{i=j} (s_{ii} - 1)^2}$	$\dfrac{c_1 \text{sum}(W') + c_2 \text{trace}(W^2)}{\text{trace}(S^2) - 2\text{trace}(S) + n}$
Target B $T_{ij} = \begin{cases} v = \text{avg}(s_{ii}) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$	$\dfrac{\sum_{i \neq j} z_{ij} + \sum_{i=j} z_{ii}}{\sum_{i \neq j} s_{ij}^2 + \sum_{i=j} (s_{ii} - v)^2}$	$\dfrac{c_1 \text{sum}(W') + c_2 \text{trace}(W^2)}{\text{trace}(S^2) - 2v\text{trace}(S) + nv^2}$
Target C $T_{ij} = \begin{cases} v = \text{avg}(s_{ii}) & \text{if } i = j \\ c = \text{avg}(s_{ij}) & \text{if } i \neq j \end{cases}$	$\dfrac{\sum_{i \neq j} z_{ij} + \sum_{i=j} z_{ii}}{\sum_{i \neq j} (s_{ij} - c)^2 + \sum_{i=j} (s_{ii} - v)^2}$	$\dfrac{c_1 \text{sum}(W') + c_2 \text{trace}(W^2)}{d_C}$
Target D $T_{ij} = \begin{cases} s_{ii} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$	$\dfrac{\sum_{i \neq j} z_{ij}}{\sum_{i \neq j} s_{ij}^2}$	$\dfrac{n_D}{\text{trace}(S^2) - \text{trace}(S)}$

Table 2.1: Table of implicit calculation of optimal value of the shrinkage parameter γ for four of the common targets provided in [96], where $d_C = \text{trace}(S^2) - 2c\text{sum}(S) + 2(c-v)\text{trace}(S) + \frac{n(n-1)c^2}{2} + nv^2$ and $n_D = c_1 \text{sum}(W') + c_2 \text{trace}(W^2) - c_1 \text{trace}(W_2) - c_2 \text{trace}(W')$.

between two random variables by controlling for a set of other random variables. The partial correlation $\text{par}(U, V|Z)$ between random variables U and V while controlling for a third variable Z is the correlation between the residuals R_U and R_V, where R_U is the linear regression of U with Z:

$$\text{par}(U, V|Z) = \frac{\text{corr}(U, V) - \text{corr}(U, Z)\text{corr}(V, Z)}{\sqrt{(1 - \text{corr}(U, Z)^2)((1 - \text{corr}(V, Z)^2))}}.$$

If vectors a_i, a_v and a_z contain samples of the probability distribution given by the random variables U, V and Z respectively, then the sample partial correlation estimate is given by

$$p_{uv} = \frac{w_{uv} - w_{uz} w_{vz}}{\sqrt{(1 - w_{uz}^2)(1 - w_{vz}^2)}}.$$

For a population of variables, it will also be interesting to look at partial correlations while controlling for all other variables. Interestingly, the inverse of the correlation matrix W^{-1} provides an easy way to calculate all the pairwise partial correlations while controlling for all other variables. The partial correlation estimate between random variables U and V is

$$p_{uv} = -\frac{\left(w^{-1}\right)_{uv}}{\sqrt{\left(w^{-1}\right)_{uu}\left(w^{-1}\right)_{vv}}}$$

As a matrix formula, the partial correlation matrix can be computed by

$$P = -DW^{-1}D \tag{2.2}$$

where D is a diagonal matrix with $d_{ii} = \frac{1}{\sqrt{(w^{-1})_{ii}}}$. If $n > k$, then the matrix W does not have full rank, and is not invertible. We typically use the shrinkage estimator from the previous section (with Target A) to compute a better estimate of correlations $W^* = \gamma I + (1-\gamma)W$ that is invertible. Using the eigendecomposition of $W = \sum_i \lambda_i u_i u_i^T$, we can determine the spectrum of $(W^*)^{-1}$:

$$\begin{aligned}
(W^*)^{-1} &= \sum_{i=1}^{k} \frac{1}{((1-\gamma)\lambda_i + \gamma)} u_i u_i^T + \sum_{i=k+1}^{n} \frac{1}{\gamma} u_i u_i^T \\
&= \sum_{i=1}^{k} \frac{1}{((1-\gamma)\lambda_i + \gamma)} u_i u_i^T + \sum_{i=k+1}^{n} \frac{1}{\gamma} u_i u_i^T + \sum_{i=1}^{k} \frac{1}{\gamma} u_i u_i^T - \sum_{i=1}^{k} \frac{1}{\gamma} u_i u_i^T \\
&= \sum_{i=1}^{k} \frac{1}{((1-\gamma)\lambda_i + \gamma)} u_i u_i^T + \frac{1}{\gamma} I - \sum_{i=1}^{k} \frac{1}{\gamma} u_i u_i^T \\
&= \frac{1}{\gamma} I + \sum_{i=1}^{k} \left(\frac{1}{((1-\gamma)\lambda_i + \gamma)} - \frac{1}{\gamma} \right) u_i u_i^T \\
&= \frac{1}{\gamma} I + Y
\end{aligned}$$

where Y is a matrix whose SVD can be expressed as $\sum_{i=1}^{k} \left(\frac{1}{((1-\gamma)\lambda_i + \gamma)} - \frac{1}{\gamma} \right) u_i u_i^T$. Y can be written as a dot product network for some data matrix E, that is, $Y = EE^T$. Note that Y shares the same set of eigenvectors as W with eigenvalues $\eta_i = \frac{1}{((1-\gamma)\lambda_i + \gamma)} - \frac{1}{\gamma}$. It can be easily shown that E shares the same singular vectors of B with singular values η_i, and can be explicitly computed: $E = \sum_i \eta_i u_i v_i^T$, where v_i's are the eigenvectors of $B^T B$. Rewriting Equation 2.2, we get the matrix

of partial correlations to be

$$\begin{aligned} P &= -D\left(\frac{1}{\gamma}I - Y\right)D \\ &= -\frac{1}{\gamma}D^2 - DYD \\ &= -\frac{1}{\gamma}D^2 - DEE^TD \\ &= -\frac{1}{\gamma}D^2 - DE(D^TE)^T \\ &= -\frac{1}{\gamma}D^2 - (DE)(DE)^T \\ &= -\frac{1}{\gamma}D^2 - FF^T \\ &= -\frac{1}{\gamma}D^2 - Z \end{aligned}$$

for some matrix $F = DE$ and $Z = FF^T$. Once E is computed using its SVD, F can also be computed and explicitly stored. This then allows to compute the spectrum of Z. Thus the partial correlation matrix can be expressed implicitly as a diagonal matrix whose entries can be determined in $O(nk)$ time, and a correlation matrix whose spectrum can be easily computed in $O(nk^{\omega-1})$ time, which is the same time required to compute the spectrum of the original correlation matrix BB^T.

It is also easy to multiply P with a vector x using the spectral decomposition of Z (see Section 2.3). Also note that we can always multiply a diagonal matrix[1] D with a vector x implicitly, since the i^{th} term is just $d_{ii}x(i)$. This then allows us to use the power method to find the top few eigenvectors and eigenvalues of P. It should be noted that we can also compute any individual entry in P in only $O(nk)$ time after computing the spectrum of Z (see Section 2.3). Also, we can compute the matrix product PX , where X is an $n \times k$ matrix in only $O(nk^{\omega-1})$ time.

In Chapter 5, we will discuss the coherence matrix, another useful metric to represent relationships between random variables. We also show how to express a coherence network as a dot product network.

[1]The diagonal matrix is stored only as a vector d, where $d(i) = d_{ii}$.

2.6 Dimensionality Reduction via Random Projection

In many situations we do not have feature vectors in the framework of "large n small k" data. Instead the number of features expressed by an individual node could be extremely large. For example, in text mining, there could be a feature for every phrase of three words (so feature vectors are very large but sparse). If the feature vectors are sparse, then matrix-vector multiplication can be performed extremely fast by just multiplying the corresponding non-zero entries. The power method can then be easily used to compute the top few eigenvectors which are solutions to various cut and centrality measurements. However, for measurements that require the entire spectrum, it may be required to reduce the set of features into a manageable set so that analysis of such networks can be performed efficiently. This is true especially in expressing edge weights as partial correlations.

One of the most common linear transformation techniques to reduce dimensions is to use principal component analysis (PCA), which seeks to preserve as much of the variance of the original data as possible. PCA accomplishes this by projecting each feature vector into a set of top eigenvectors of the covariance matrix. For an $n \times k$ data matrix B, we get the transformation

$$B' = EB$$

where the $d \times n$ matrix E contains the top d eigenvectors of the covariance matrix $S = \frac{1}{(k-1)} BB^T$ along its rows normalized to unit length. The above transformation requires computing the partial spectrum of a matrix, which could be computationally expensive especially if the number of original dimensions k is too high.

Another common method is based on random projections. A random projection is given by a function $f : \mathbb{R}^k \to \mathbb{R}^d$ that maps a length-k feature vector u into $f(u) = Gu$ with reduced dimension d. The $d \times k$ matrix G contains elements independently chosen from a Gaussian distribution with mean 0 and standard deviation $\frac{1}{\sqrt{d}}$; i.e., $g_{ij} \sim N(0, \frac{1}{\sqrt{d}})$. For any two feature vectors u and v, we have

$$\mathrm{E}\left[f(u) \cdot f(v)\right] = \mathrm{E}\left[Gu \cdot Gv\right] = \mathrm{E}\left[(Gu)^T Gv\right] = \mathrm{E}\left[u^T G^T Gv\right] = u^T \mathrm{E}\left[G^T G\right] v.$$

The expected value of a matrix is the expected value of its individual entries. Each entry in $G^T G$ is a dot product of two columns of G. Let G_i be the i^{th} column of G. Then the expected value

of the dot product between two different columns $G_i \cdot G_j$ is 0 since they are products and sums of independent Gaussian random variables each with mean 0, and $G_i \cdot G_i$ is chi-squared distributed with expected value[2] 1. We therefore have $\mathrm{E}\left[G^T G\right] = I$. Substituting in the above equation, we have

$$\mathrm{E}\left[f(u) \cdot f(v)\right] = u \cdot v.$$

The expected squared length of u after applying the mapping is $\mathrm{E}\left[||f(u)||^2\right] = \mathrm{E}\left[f(u) \cdot f(u)\right] = u \cdot u = ||u||^2$. Hence, f preserves both squared lengths and dot products in expectation.

A worst case bound on the distribution of lengths (or dot products) can be accomplished using the Johnson-Lindenstrauss lemma [60], which states that for a set of points $x_1 \ldots x_n$ in $\mathbb{R}^k$, there exists a function $f : \mathbb{R}^k \to \mathbb{R}^d$ such that

$$(1-\epsilon)||x_i - x_j|| < ||f(x_i) - f(x_j)|| < (1+\epsilon)||x_i - x_j||$$

for any error bound $\epsilon > 0$. As a corollary, due to the law of cosines, the dot products between all pair of points are also approximately preserved. The number of dimensions d we need to preserve all pairwise distances and dot products is at least $O(\epsilon^{-2} \log n)$. This is very surprising, since the number of dimensions is only dependent on the number of points n and not on either k or d. For $n \approx 100000$ points, the number of dimensions we need would be in the thousands in order to preserve all pairwise dot products within an error bound of 0.1. We are, however, content to have weaker guarantees that will in turn allow us to reduce to even fewer dimensions. Even though both lengths and dot products are preserved in expectation under f, there could be large variance if the number of dimensions d is small. Therefore to provide a threshold of the number of dimensions we need, it is required to compute the variance of the resulting dot product. Due to rotational symmetry of f, without loss of generality let $u = [1, 0, 0, \ldots, 0]$ and $v = [\cos\phi, \sin\phi, 0, 0, \ldots, 0]$ where ϕ is the angle between u and v. Let $X = f(u) \cdot f(v)$ be a random variable. Then the variance $\mathrm{var}[X]$ is the expected squared deviation from the mean:

$$\mathrm{var}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2 = \mathrm{E}[X^2] - \cos^2\phi. \tag{2.3}$$

[2]If Z is chi-squared distributed, being a sum of squares of d independent Gaussians with mean 0 and standard deviation σ, then Z has expected value $d\sigma^2$.

Now

$$\begin{aligned}
\mathrm{E}[X^2] &= \mathrm{E}\left[(Gu \cdot Gv)^2\right] \\
&= \mathrm{E}\left[(G_1 \cdot (G_1 \cos\phi + G_2 \sin\phi))^2\right] \\
&= \mathrm{E}\left[((G_1 \cdot G_1)\cos\phi + (G_1 \cdot G_2)\sin\phi)^2\right] \\
&= \mathrm{E}\left[(G_1 \cdot G_1)^2 \cos^2\phi + (G_1 \cdot G_2)^2 \sin^2\phi + 2\cos\phi(G_1 \cdot G_1)(G_1 \cdot G_2)\sin\phi\right] \\
&= \cos^2\phi + \mathrm{E}\left[(G_1 \cdot G_2)^2\right]\sin^2\phi + \mathrm{E}\left[(G_1 \cdot G_1)(G_1 \cdot G_2)\right] 2\cos\phi\sin\phi \\
&= \mathrm{E}\left[(G_1 \cdot G_1)^2\right]\cos^2\phi + \mathrm{E}\left[(G_1 \cdot G_2)^2\right]\sin^2\phi + \mathrm{E}\left[(G_1 \cdot G_1)\right]\mathrm{E}\left[(G_1 \cdot G_2)\right] 2\cos\phi\sin\phi \\
&= \mathrm{E}\left[(G_1 \cdot G_1)^2\right]\cos^2\phi + \mathrm{E}\left[(G_1 \cdot G_2)^2\right](1 - \cos^2\phi) \qquad (2.4)
\end{aligned}$$

since $G_1 \cdot G_2$ is adding up products of independent Gaussians with mean 0, so $\mathrm{E}[(G_1 \cdot G_2)] = 0$. Expanding $\mathrm{E}[(G_1 \cdot G_1)^2]$, we get

$$\begin{aligned}
\mathrm{E}\left[(G_1 \cdot G_1)^2\right] &= \mathrm{E}\left[\left(\sum_{r=1}^{d} G_{r1}^2\right)^2\right] \\
&= \mathrm{E}\left[\sum_{r=1}^{d} G_{r1}^4\right] + \mathrm{E}\left[\sum_{r \neq r'} G_{r1}^2 G_{r'1}^2\right] \\
&= \sum_{r=1}^{d} \mathrm{E}\left[G_{r1}^4\right] + \sum_{r \neq r'} \mathrm{E}\left[G_{r1}^2 G_{r'1}^2\right].
\end{aligned}$$

Since G_{r1} and $G_{r'1}$ are independent, we have

$$\mathrm{E}\left[(G_1 \cdot G_1)^2\right] = \sum_{r=1}^{d} \mathrm{E}\left[G_{r1}^4\right] + \sum_{r \neq r'} \mathrm{E}\left[G_{r1}^2\right]\mathrm{E}\left[G_{r'1}^2\right].$$

The variables G_{r1}^2 and $G_{r'1}^2$ are chi-squared distributed with mean $\frac{1}{d}$. Also, G_{r1}^4 is the fourth moment of a Gaussian random variable, which has mean[3] $3\left(\frac{1}{\sqrt{d}}\right)^4 = \frac{3}{d^2}$. Putting all these together, we have

$$\mathrm{E}\left[(G_1 \cdot G_1)^2\right] = \sum_{r=1}^{d} \frac{3}{d^2} + \sum_{r \neq r'} \frac{1}{d^2} = d\frac{3}{d^2} + (d^2 - d)\frac{1}{d^2} = 1 + \frac{2}{d}. \qquad (2.5)$$

[3]If $Z \sim N(0, \sigma)$ is a Gaussian random variable, then $\mathrm{E}[Z^4] = 3\sigma^4$.

Similarly, expanding $\mathrm{E}\left[(G_1 \cdot G_2)^2\right]$ in (2.4), we get

$$\begin{aligned}
\mathrm{E}[(G_1 \cdot G_2)^2] &= \mathrm{E}\left[\left(\sum_{r=1}^{d} G_{r1}G_{r2}\right)^2\right] \\
&= \mathrm{E}\left[\sum_{r=1}^{d} G_{r1}^2 G_{r2}^2\right] + \mathrm{E}\left[\sum_{r\neq r'} G_{r1}^2 G_{r'2}^2\right] \\
&= \sum_{r=1}^{d} \mathrm{E}[G_{r1}^2 G_{r2}^2] + \sum_{r\neq r'} \mathrm{E}[G_{r1}^2 G_{r'2}^2].
\end{aligned}$$

G_{r1}^2 and $G_{r'2}^2$ are independent. Therefore $\mathrm{E}\left[G_{r1}^2 G_{r'2}^2\right] = 0$. Also, $\mathrm{E}\left[G_{r1}^2 G_{r2}^2\right] = \mathrm{E}\left[G_{r1}^2\right]\mathrm{E}\left[G_{r2}^2\right] = \frac{1}{d^2}$ since $\mathrm{E}\left[G_{r1}^2\right]$ is the expected value of the second moment of a Gaussian which is its variance $\frac{1}{d}$. Combining these in the equation above, we get

$$\mathrm{E}\left[(G_1 \cdot G_2)^2\right] = \sum_{r=1}^{d} \frac{1}{d^2} = \frac{1}{d}. \tag{2.6}$$

Substituting values obtained in (2.5) and (2.6) in (2.4), we get

$$\begin{aligned}
\mathrm{E}[X^2] &= (1 + \frac{2}{d})\cos^2\phi + \frac{1}{d}(1 - \cos^2\phi) \\
&= \frac{1}{d} + (1 + \frac{1}{d})\cos^2\phi
\end{aligned} \tag{2.7}$$

Substituting this in (2.3), we get

$$\begin{aligned}
\mathrm{var}[X] &= \frac{1}{d} + (1 + \frac{1}{d})\cos^2\phi - \cos^2\phi \\
&= \frac{1}{d} + \frac{1}{d}\cos^2\phi \\
&= \frac{1}{d} + \frac{1}{d}|u \cdot v|^2.
\end{aligned} \tag{2.8}$$

Since $|u \cdot v|^2$ takes values in $[0, 1]$, the variance lies in $\left[\frac{1}{d}, \frac{2}{d}\right]$ and the standard deviation in $\left[\sqrt{\frac{1}{d}}, \sqrt{\frac{2}{d}}\right]$. If we need the standard deviation in our estimate of dot products to be about 0.1, then we would therefore need to map to 200 dimensions. Note that this is independent of the number of initial dimensions k.

The random projection technique discussed above involves n matrix-vector multiplications,

each of which take $O(kd)$ time to compute in a straightforward fashion. Ailon et al. [1, 2] showed how to speed it up by modifying the distribution of G. Another approach is to use a distribution over sparse matrices [34, 62].

As a small caveat, the above projection does not preserve non-negativity, and is undesirable with dot product networks that are already non-negative because it can actually introduce negativity where there was none before. In Chapter 5, we introduce a different mapping that preserves non-negativity by approximating the absolute value or squares of the original correlations in expectation.

Chapter 3

Centrality

In this chapter, we study centrality measures in the context of implicit correlation networks. Centrality measures the influence or importance of individual nodes. There are many definitions of centrality in the literature. We look at three ways to define centrality – degree centrality, eigenvector centrality, and clustering coefficient – and show how we can compute these measures in our implicit framework.

It should be noted that most of the definitions in the literature are expressed in the context of non-negative networks. These definitions do not often work for correlation networks of the form $W = BB^T$. A common practice in correlation network analysis is to take the absolute values or squares of the correlations, since both highly positive and negative correlations must often be treated equally as they still provide linear relationships between the respective random variables. We attack this issue by converting the correlation network into another non-negative dot product network that approximately preserves the absolute values or squares of the original correlations. One of the major novel ideas of this thesis is in the usage of locality sensitive hash (LSH) functions in performing this conversion (discussed in Chapter 5). Using this transformation, definitions of centrality and cuts defined exclusively for non-negative networks can be used. We use this approach in this chapter for both eigenvector centrality and clustering coefficients, and in many clustering and cut algorithms discussed in Chapter 4.

We begin this chapter with a discussion of centrality in the context of non-negative networks. We provide definitions of degree centrality, eigenvector centrality and clustering coefficient found in the literature for non-negative networks and show how they can be computed implicitly for non-

negative dot product networks. Then, we discuss a generalization of degree centrality and clustering coefficient for correlation networks that contain negative edge weights.

3.1 Non-negative Dot Product Networks

In this section, we provide some common definitions of degree centrality, eigenvector centrality and clustering coefficient for non-negative networks. We show how they can be computed in our implicit dot product framework. To use these with negative correlation networks, one would need to first convert to another dot product network that is non-negative using an LSH transformation, a method that is discussed in Chapter 5.

3.1.1 Degree Centrality

As mentioned in Chapter 1, the node degree of a node i is the number of edge connections it has and does not provide useful information in the context of dot product networks, since $d(i) = n-1$ for all nodes i (one can include the self loop $w_{ii} = 1$, in which case $d(i) = n$). We therefore use node strength $s(i) = \sum_j w_{ij}$. The vector of node strengths can be computed implicitly by paranthezation:

$$\begin{aligned} c_{\text{strength}} &= W\mathbf{1} \\ &= B(B^T\mathbf{1}). \end{aligned}$$

For negative correlation networks, node strength cannot be used, since both highly positive and negative correlations provide linear relationships between the random variables. A node should have high centrality if it contains an equal distribution of highly positive and negative edge weights, but could end up with low node strength. In Section 3.2.1, we discuss several statistical properties of the weight distribution of a node among its neighbors, which provide an extension of the node strength to negative networks.

If edge weights are expressed as squared correlations, we can apply the LSH transformation discussed in Chapter 5. But a simpler approach can be used to compute the node strengths without

requiring to perform the LSH transformation. In this case the node strength of a node i is

$$s(i) = (W^2)_{ii}$$

since $(W^2)_{ii} = \sum_j w_{ij}^2$. The diagonal entries of W^2 can be computed in only $O(nk)$ time after computing the spectrum of W, which takes $O(nk^{\omega-1})$ time. An even simpler method is to multiply by parenthezation $(W^2)_{ii} = (B((B^TB)B^T))_{ii}$. The diagonal element $(W^2)_{ii}$ is the dot product between b_i and the i^{th} column of $(B^TB)B^T$. It takes only $O(nk)$ time to compute all the diagonal elements.

3.1.2 Eigenvector Centrality

As mentioned in Chapter 1, eigenvector centrality can be seen as an extension of node degree (for unweighted networks) and node strength (for weighted networks). Extending the definition of Bonacich's [**?**] formalization to non-negative dot product networks $W = BB^T$, we see that the centrality of a node i should satisfy

$$c_{\text{eig}}(i) = \alpha \sum_j w_{ij} c_{\text{eig}}(j)$$

where α is a constant of proportionality that will end up being related to the largest eigenvalue of W. In matrix form, we are looking for a vector c_{eig} that satisfies

$$c_{\text{eig}} = \alpha W c_{\text{eig}}.$$

So clearly c_{eig} must be an eigenvector of W, and moreover, we get maximum "reinforcement" by taking $\alpha = \frac{1}{\lambda_1}$, where λ_1 is the largest eigenvalue of W. That is, the vector of centralities c_{eig} satisfies $Wc_{\text{eig}} = \lambda_1 c_{\text{eig}}$, and therefore is the principal eigenvector of W. The entire spectrum of W can be computed in only $O(nk^{\omega-1})$ time, but it is typically much faster to use the power method (and for partial correlations, this is the only option!).

For negative correlation networks, the eigenvector centrality could contain negative elements and hence cannot easily be interpreted as giving centrality measurements any more. The interpretation of these elements can be understood from the context of clustering in principal component

analysis, which we discuss in Chapter 4.

Two other generalizations of the eigenvector centrality, known as Katz centrality [64] and PageRank centrality [21] require that a minimum amount of "free centrality" be added to every node in the network. Formalizing this notion, we have

$$c_{\text{Katz}}(i) = \alpha \sum_j w_{ij} c_{\text{Katz}}(j) + \beta \tag{3.1}$$

$$c_{\text{PageRank}}(i) = \alpha \sum_j w_{ij} \frac{c_{\text{Katz}}(j)}{s(j)} + \beta \tag{3.2}$$

where α is a positive constant that introduces a bias for the centrality term, and β is a positive constant that gives an amount of minimum centrality to all nodes. The above centralities can be expressed in matrix form as

$$\begin{aligned} c_{\text{Katz}} &= \alpha W c_{\text{Katz}} + \beta \mathbf{1} \\ c_{\text{PageRank}} &= \alpha W D^{-1} c_{\text{PageRank}} + \beta \mathbf{1} \end{aligned} \tag{3.3}$$

where D is a diagonal matrix with $d_{ii} = s(i)$. It can be observed by rearranging the terms and solving for respective centralities, that β only multiplies every centrality by a constant, and therefore we can set $\beta = 1$. Putting all these together, we get

$$\begin{aligned} c_{\text{Katz}} &= (I - \alpha W)^{-1} \mathbf{1} \\ c_{\text{PageRank}} &= D(D - \alpha W)^{-1} \mathbf{1}. \end{aligned}$$

To find Katz and PageRank centralities, it is required to set α. When α is set to 0, only the constant β term remains in Equation 3.3. When α is increased, the centralities also increase until they reach a point where they diverge. For Katz centrality, it can be seen that this happens at the point where $\det |A - \alpha^{-1} I|$ is 0. Thus, the value of α^{-1} is the largest root of the characteristic equation and is equal to the largest eigenvalue λ_1 of W. To give the maximum weight to the eigenvector centralities, α is chosen to be slightly less than $\frac{1}{\lambda_1}$. Using the same analogy for PageRank centrality, we see that α^{-1} should be set slightly less than the largest eigenvalue of WD^{-1}. The matrix WD^{-1} is also called a walk matrix, in which all the columns are normalized to add to 1. In such a matrix, the largest eigenvalue is 1.

Katz centrality can also be computed implicitly from the SVD of $W = \sum_i \lambda_i u_i u_i^T$, since the matrix $(I - \alpha W)^{-1}$ contains the same set of eigenvectors as W with eigenvalues $\frac{1}{(1-\alpha\lambda_i)}$. For PageRank centrality, one could use the power method in Equation 3.3 and compute c_{PageRank} implicitly. We could also expand equation 3.3 to bring it into a familiar form for the power method:

$$\begin{aligned} x &= \alpha W D^{-1} x + \beta \mathbf{1} \\ &= \alpha W D^{-1} x + \beta \mathbf{1}_{n\times n} x \\ &= \left(\alpha W D^{-1} + \beta \mathbf{1}_{n\times n}\right) x \end{aligned}$$

where $\mathbf{1}_{n\times n}$ is $n \times n$ matrix with all ones. It therefore follows that PageRank centrality is the principal eigenvector of $\left(\alpha W D^{-1} + \beta \mathbf{1}_{n\times n}\right)$.

3.1.3 Clustering Coefficient

As mentioned in Chapter 1, the local clustering coefficient of a node i in unweighted networks is the fraction of the number of triangles containing i to the total number of possible triangles that can be formed with it. This definition is due to Watts and Strogatz [112]:

$$c_{\text{Watts}}(i) = \frac{\sum_{j\neq q} a_{ij} a_{jq} a_{qi}}{d(i)(d(i)-1)}.$$

Among several generalizations [16, 76, 87, 95, 61, 77, 119] of the clustering coefficient to weighted networks, Zhang and Horvath [119] studied them in the context of gene co-expression networks which follow our model of large n small k dot product networks. The nodes in the network correspond to genes, and the edges correspond to the similarity (mostly Pearson's correlation) between the expressions of the corresponding genes. They define clustering coefficient as

$$c_{\text{Zhang}}(i) = \frac{\sum_{j,q} w_{ji} w_{iq} w_{jq}}{\left(\sum_q w_{iq}\right)^2 - \sum_q w_{iq}^2}.$$

It can be easily seen that the numerator is a generalization of c_{Watts}. The denominator, which

simplifies to $\sum_{j \neq q} w_{ji} w_{iq}$, is a normalizing term that keeps the value of $C_{\text{Zhang}}(i)$ in $[0, 1]$. The above definition can easily be computed implicitly

$$c_{\text{Zhang}}(i) = \frac{\left(w^3\right)_{ii}}{(W\mathbf{1})(i)^2 - \left(w^2\right)_{ii}}.$$

The term in the numerator denotes the sum of all cycles of length 3 that start and finish at node i. As seen earlier, any single entry of $\left(w^l\right)_{ii}$ can be computed in only $O(k)$ time from the eigendecomposition of W. In the denominator, $(W\mathbf{1})(i)$ denotes the node strength of i, and subtracting $\left(w^2\right)_{ii}$ from the square of node strength gives the required sum. This computation takes $O(nk)$ time in total for computing c_{Zhang} for all the nodes.

The above definition was used to analyze the correlation network of gene co-expressions, where Zhang and Horvath [119] preprocessed the network to make it non-negative by taking absolute values. Taking absolute values of individual entries in W is difficult in our implicit framework. We tackle this issue by approximating the absolute values by using a family of LSH functions, discussed in Chapter 5. In Section 3.2.2, we discuss generalization of clustering coefficient for the negative edge weight case by Constantini and Perugini [32].

3.2 Negative Dot Product Networks

For both eigenvector centrality and clustering coefficient, we use an LSH transformation to convert a correlation network into another non-negative dot product network that approximately preserves the absolute values or squares of the correlations in expectation (see Chapter 5). Here we extend the notion of the degree centrality to negative weighted networks.

3.2.1 Generalization of the Degree Centrality

Node strength $s(i)$ cannot effectively be applied to signed weighted networks, since a node can have zero strength if all its edge weights are evenly distributed between positive and negative. In correlation networks of the form $W = BB^T$, both highly positive and negative values indicate that there are linear relationships between the corresponding random variables, and hence both often need to be considered as important. Therefore, we may consider looking at second order (or higher) distributions of weights around a node.

The first moment or the mean of the weight distribution around a node i is the ratio of node strength to degree $c_{mean}(i) = \dfrac{s(i)}{(n-1)}$. The mean indicates how much the node strength is shared across individual links. The first moment is just normalized node strength and still does not eliminate the issue of a node having a large number of equally distributed positive and negative edges. The second moment, on the other hand, distinguishes these cases. The second central moment is defined as

$$c_{\text{second}}(i) = \frac{\sum_{j \neq i} w_{ij}^2}{(n-1)}.$$

Note that this measure is non negative, and deals with negative edges in a very natural fashion. An edge with weight -10 contributes more to the centrality than an edge that is -1, but just as much as an edge weight of $+10$. One can also consider the second moment about the mean, also called variance

$$c_{\text{variance}}(i) = \frac{\sum_{j \neq i} (w_{ij} - c_{\text{mean}}(i))^2}{(n-1)}.$$

The variance of a probability distribution measures the variability or the amount of deviation from the mean. If the variance is small, then all the edges are closely packed around the mean. If the variance is large, then the edges are widely spread out from the mean. Note that this is also a non-negative quantity, and very closely related to c_{second}. Expanding the above formula, we get:

$$\begin{aligned}
c_{\text{variance}}(i) &= \left(\frac{1}{n-1}\right)\left(\sum_{j \neq i} w_{ij}^2 - 2\sum_{j \neq i} w_{ij} c_{\text{mean}}(i) + (n-1)c_{\text{mean}}(i)^2\right) \\
&= c_{\text{second}}(i) + \left(\frac{1}{n-1}\right)\left(-2c_{\text{mean}}(i)\sum_{j \neq i} w_{ij} + (n-1)c_{\text{mean}}(i)^2\right) \\
&= c_{\text{second}}(i) + \left(\frac{1}{n-1}\right)\left(-2(n-1)c_{\text{mean}}(i)^2 + (n-1)c_{\text{mean}}(i)^2\right) \\
&= c_{\text{second}}(i) - c_{\text{mean}}(i)^2
\end{aligned}$$

The quantities $c_{\text{mean}}(i)$ and $c_{\text{second}}(i)$ (and thus $c_{\text{variance}}(i)$) can be easily computed implic-

itly using the following formulas:

$$c_{\text{mean}}(i) = \frac{W\mathbf{1}}{n}$$
$$c_{\text{second}}(i) = \frac{\left(w^2\right)_{ii}}{n-2}.$$

The term $\left(w^2\right)_{ii}$ can be computed implicitly from the singular value decomposition (SVD) of W, since the SVD of W^2 contains the same eigenvectors of W with corresponding eigenvalues squared. That is, if $\lambda_1, \ldots, \lambda_k$ are the eigenvalues[1] of W with eigenvectors $u_1, \ldots, u_k$, then

$$W^2 = \sum_{l}^{k} \lambda^2 u_i u_i^T.$$

Any individual entry of W^2 can be computed in $O(k)$ time. Therefore in only $O(nk)$ time, all the diagonal entries of W^2 can be computed.

Degree centrality can now be expressed as a triple $(c_{mean}(i), c_{second}(i), c_{variance}(i))$ to convey more information that captures well the high-level characteristics of the weight distribution. Higher order moments and other statistical properties can also be added into the triple. If only one value is required for the purposes of comparing the degree centralities of two nodes in the network, then we can have a convex combination of all the entries in the triple, determined completely by the user. For example, the user could be interested in the dispersion of the weight distribution of a node, and variance can be weighted more than the mean.

3.2.2 Clustering Coefficient For Negative Networks

In Section 3.1.3, we discussed clustering coefficient in the context of non-negative dot product networks. In this section, we discuss a generalization of clustering coefficient to negative correlation networks by Constantini and Perugini [32].

The intuition behind their generalization is as follows. Consider a node i connected to nodes j and q. If i has a positive (negative) relationship with each of j and q, then it is more likely that j and q also have a positive relationship with each other. On the other hand, if i has a mixed relationship between j and q (that is, one of the edges is positive and the other is negative), then

[1]Please note the slight notational difference of using λ instead of σ^2 for the eigenvalues of a matrix $W = BB^T$, where σ were the singular values of B.

the two nodes are more likely to also have a negative relationship with each other. In other words, if the sign of a triangle is the product of the signs of its edges, then a positive triangle around a node i contributes positively to the clustering coefficient index, and a negative triangle around i contributes negatively to the clustering coefficient index. Thus, the signed clustering coefficient of a node i is high if the number of negative edges in the triangle (i, j, q) is 2 or 0. It is low if the number of negative edges in the triangle is 1 or 3. So i has high local clustering coefficient, if it is connected to j and q with same (opposite) signs, and the direct edge (j, q) is more likely to be positive (negative).

It is to be noted that the signed clustering coefficient is more robust than its unsigned counterpart in the presence of noise. If the pairwise correlations are drawn from a small sample size, then they could be unreliable estimates [99] of the overall population. This could result in a large number of small triangles that are equally distributed between negative and positive cycles. All these triangles cancel out each other in the signed clustering coefficient index, whereas they contribute to the unsigned clustering coefficient index. This is also one of the reasons to use shrinkage estimators, to get more reliable estimates of the actual correlations between two stochastic processes.

Constantini and Perugini [32] provide a generalization of the unsigned clustering coefficient of Zhang and Hovrath [119]

$$c_{\mathrm{CP}}(i) = \frac{\sum_{j,q} w_{ij} w_{iq} w_{jq}}{\sum_{j \neq q} |w_{ij} w_{iq}|} \tag{3.4}$$

Note that signed weights are used in the numerator, but absolute values are used in the denominator. The denominator can be considered as a normalizing term, and use of negative quantities could affect the overall index. Also note that this definition is equivalent to c_{Zhang} if the network is non-negative.

The above definition of clustering coefficient was extensively tested by Constantino and Pergini [32] against simulated networks, and an actual personality psychology network. Though the numerator can be computed implicitly ($\left(w^3\right)_{ii}$), the absolute values in the denominator are problematic, since they requires modification of individual entries of W, which never exists in explicit form. We use LSH functions to transform the correlation network into a non-negative dot product network that approximately preserves the absolute values of the correlations. In the new dot product network, the unsigned clustering coefficient c_{Zhang} can be computed implicitly, and also the denominator in Equation 3.4.

Taking an aggregate of the local clustering coefficient of all the nodes, we get the *global clustering coefficient* of a network, which provides a measurement of how well all the nodes are connected in the network. In fact, any of the centrality measures can be aggregated to get a global network measurement. There are other definitions of global clustering coefficient which do not take a mean of the local clustering coefficients. One such definition for signed weighted networks given by Constantini and Perugini [32] takes the mediant of the local clustering coefficients

$$\text{gcc} = \frac{\sum_i \sum_{j,q} w_{ij} w_{jq} w_{qi}}{\sum_i \sum_{j \neq q} |w_{ij} w_{iq}| + \sum_i \sum_j w_{ij}^2}.$$

Though the numerator can be computed implicitly ($\text{trace}(W^3)$), the denominator again contains absolute values and is difficult to compute implicitly. In Chapter 5, we use locality sensitive hashing to transform a negative correlation network into a non-negative dot product network that approximately preserves the squares or absolute values of the correlations in expectation. This would allows us to implicitly compute both local and global clustering coefficients that are defined exclusively for non-negative networks.

Chapter 4

Clustering and Cuts

In this chapter, we study clustering and cut algorithms, which provide information of the global structure of the network. One obtain several such global properties by taking an aggregate of centrality measures discussed in the last chapter, just like for clustering coefficient (see Section 3.2.2). Both clustering and cut algorithms provide ways to decompose a graph into smaller communities (usually two, for cut algorithms). We start this chapter with a discussion of cut and clustering algorithms that are defined for non-negative networks. We can apply the locality sensitive hashing transformation discussed in Chapter 5 to convert any correlation network into another non-negative dot product network that approximately preserves the absolute values or squares of the correlations. We show how well-known spectral relaxations of the ratio and normalized cut objectives can be computed in our implicit framework. Next, we discuss modularity, which can be viewed from both cut and clustering perspectives. We also briefly discuss eigen gap which measure how well nodes in the network are connected. For correlation networks that contain negative edge weights, we draw insights from principal component analysis and correlation clustering, and show how they can be addressed in our implicit setting.

4.1 Non-Negative Dot Product Networks

For all algorithms in this section, we assume that the dot product network $W = BB^T$ is non-negative. A cut $(C, \overline{C})$ of a network is a partition of the node set V of the network into two disjoint pieces $C \subseteq V$ and $\overline{C} = V - C$. The value of the cut, written $\text{cut}(C, \overline{C})$ (sometimes just

cut(C)), is the sum of the weights of edges that cross the cut. That is,

$$\text{cut}(C) = \sum_{i \in C, j \in \overline{C}} w_{ij}.$$

The global minimum cut of a graph is the minimum value of a cut among all possible cuts in the graph. That is, the global minimum cut is the solution to

$$\min_C \{\text{cut}(C)\} = \min_C \left\{ \sum_{i \in C, j \in \overline{C}} w_{ij} \right\}.$$

Though the global minimum cut can be computed in polynomial time [81, 104], most networks yield a very trivial solution: a single node on one side of the cut and everything else on the other side. In the next two sections, we discuss *ratio* and *normalized* cuts that are designed to produce more balanced partitions, and show how they can be computed for non-negative dot product networks.

4.1.1 Ratio Cuts

The ratio cut objective was proposed by Wei and Cheng [113, 114], and independently by Leighton and Rao [73]. It has its applications in VLSI design [52, 114], especially in layout design, testing and hardware simulations, clustering in machine learning, and partitioning the background from an image in image segmentation [111]. A ratio cut is a cut that minimizes the value of the cut proportional to its size (the number of nodes). Formally, it is the solution to

$$\min_C \{\text{rcut}(C)\} = \min_C \left\{ \frac{\text{cut}(C, \overline{C})}{|C|} + \frac{\text{cut}(C, \overline{C})}{|\overline{C}|} \right\}$$

Unfortunately, finding a cut that minimizes the above objective is NP-Hard by a reduction from Bounded Min-Cut Graph Partition [45] (also see [109] for a discussion). Therefore, we discuss a well known spectral relaxation of the problem that was developed by Hagen and Kahng [52]. Before we discuss this, we briefly review the Laplacian matrix of a weighted graph.

4.1.1.1 Laplacian Matrix

The Laplacian matrix L for a symmetric non-negative weighted network $W = BB^T$ is defined as

$$L = D - W$$

where D is the diagonal matrix with $d_{ii} = \sum_j w_{ij}$. The Laplacian matrix satisfies the following properties.

1. For any vector $x \in \mathbb{R}^n$, $x^T L x = \frac{1}{2}\sum_{i,j} w_{ij}(x(i) - x(j))^2$. This can be seen from the following derivation:

$$\begin{aligned}
x^T L x = &\ x^T D x - x^T W x \\
= &\ \sum_i d_i x(i)^2 - \sum_{i,j} w_{ij} x(i) x(j) \\
= &\ \frac{1}{2}\left(\sum_i d_i x(i)^2 - 2\sum_{i,j} w_{ij} x(i) x(j) + \sum_j d_j x(j)^2\right) \\
= &\ \frac{1}{2}\left(\sum_{i,j} w_{ij} x(i)^2 - 2\sum_{i,j} w_{ij} x(i) x(j) + \sum_{i,j} w_{ij} x(j)^2\right) \\
= &\ \frac{1}{2}\left(\sum_{i,j} w_{ij}\left(x(i) - x(j)\right)^2\right).
\end{aligned}$$

2. L is symmetric, since both W and D are symmetric.

3. L is positive semi-definite, so all eigenvalues are non-negative. This can be seen from the equation above: $x^T L x = \frac{1}{2}\left(\sum_{i,j} w_{ij}\left(x(i) - x(j)\right)^2\right) \geq 0$.

4. L is singular. Since $v = \mathbf{1}$ is an eigenvector associated with the eigenvalue 0 and L is positive semi-definite. This can be seen from the fact that $v^T L v = \frac{1}{2}\left(\sum_{i,j} w_{ij}\left(v(i) - v(j)\right)^2\right) = 0$.

5. L has real eigenvalues, since it is symmetric.

4.1.1.2 Ratio Cut Clustering

We first present the well known linear algebraic formulation of the ratio cut objective. Next, we show how to relax the constraints so as to get an approximate solution to the ratio cut problem. We then show how this can be computed in our implicit framework.

Let $x = [x(1), \ldots, x(n)]$ be an indicator vector with

$$x(i) = \begin{cases} \sqrt{\frac{c}{nc'}} & \text{, if } i \in C \\ -\sqrt{\frac{c'}{nc}} & \text{, if } i \in \overline{C} \end{cases}$$

where $c = |C|$ and $c' = |\overline{C}| = n - c$. We have

$$\begin{aligned}
x^T L x = & \frac{1}{2}\left(\sum_{i,j} w_{ij}\left(x(i) - x(j)\right)^2\right) \\
= & \frac{1}{2}\left(\sum_{i,j} w_{ij}\left(x(i)^2 - 2x(i)x(j) + x(j)^2\right)\right) \\
= & \frac{1}{2}\sum_{i\in C, j\in\overline{C}} w_{ij}\left(\frac{c}{nc'} + \frac{2}{n} + \frac{c'}{nc}\right) + \frac{1}{2}\sum_{i\in\overline{C}, j\in C} w_{ij}\left(\frac{c'}{nc} + \frac{2}{n} + \frac{c}{nc'}\right) \\
= & \sum_{i\in C, j\in\overline{C}} w_{ij}\left(\frac{c'}{nc} + \frac{2}{n} + \frac{c}{nc'}\right) \\
= & \ \mathrm{cut}(C,\overline{C})\left(\frac{c'}{nc} + \frac{2}{n} + \frac{c}{nc'}\right) \\
= & \ \mathrm{cut}(C,\overline{C})\left(\frac{c'}{nc} + \frac{cc'}{ncc'} + \frac{cc'}{ncc'} + \frac{c}{n^2c'}\right) \\
= & \ \mathrm{cut}(C,\overline{C})\left(\frac{c'+c}{nc} + \frac{c'+c}{nc'}\right) \\
= & \ \mathrm{cut}(C,\overline{C})\left(\frac{n}{nc} + \frac{n}{nc'}\right) \\
= & \ \mathrm{cut}(C,\overline{C})\left(\frac{1}{c} + \frac{1}{c'}\right) \\
= & \ \frac{\mathrm{cut}(C,\overline{C})}{c} + \frac{\mathrm{cut}(C,\overline{C})}{c'} \\
= & \ \frac{\mathrm{cut}(C,\overline{C})}{|C|} + \frac{\mathrm{cut}(C,\overline{C})}{|\overline{C}|}.
\end{aligned}$$

This means that the value of $\min_x \left\{x^T L x\right\} = \min_C \left\{\frac{\mathrm{cut}(C,\overline{C})}{|C|} + \frac{\mathrm{cut}(C,\overline{C})}{|\overline{C}|}\right\}$ over all indicator vectors

x gives the optimal ratio cut. Also note that

$$\sum_i x(i) = \sum_{i \in C} \sqrt{\frac{c'}{nc}} + \sum_{i \in \overline{C}} -\sqrt{\frac{c}{nc'}} = c\sqrt{\frac{c'}{nc}} - c'\sqrt{\frac{c}{nc'}} = \sqrt{\frac{cc'}{n}} - \sqrt{\frac{cc'}{n}} = 0$$

and

$$||x||^2 = \sum_i x(i)^2 = \sum_{i \in C} \frac{c'}{nc} + \frac{c}{nc'} = \frac{cc'}{nc} + \frac{cc'}{nc'} = \frac{c'}{n} + \frac{c}{n} = \frac{c + c'}{n} = \frac{n}{n} = 1.$$

Thus the vector x has zero mean and unit variance. Hence, the ratio cut problem can be re-written as

$$\min_C \{\text{rcut}(C, \overline{C})\} = \min_{\substack{||x||^2=1 \\ x \cdot \mathbf{1}=0}} \left\{x^T L x\right\}$$

where x is also constrained to be an indicator vector as above. As mentioned earlier, finding the optimal value of the indicator vector x that solves the above optimization problem is NP-Hard. If we relax the constraint that x must be an indicator vector, and instead find a solution for any vector $x \in \mathbb{R}^n$, then the relaxed ratio cut objective becomes

$$\min_{\substack{||x||^2=1 \\ x \cdot \mathbf{1}=0 \\ x \in \mathbb{R}^n}} \left\{x^T L x\right\}.$$

The solution to the above relaxed optimization problem is the eigenvector associated with the second smallest eigenvalue of L. Note that the eigenvector $v = \mathbf{1}$ associated with the smallest eigenvalue 0 does not satisfy the constraint $v \cdot \mathbf{1} = 0$. The second smallest eigenvector being perpendicular to v satisfies all the constraints and minimizes the relaxed ratio cut objective.

Note that the Laplacian matrix L of a correlation network $W = BB^T$ has the same size as W, and therefore we do not have the luxury to store L explicitly. To find the eigenvector associated with the second smallest eigenvalue, we typically find the eigenvector associated with the second largest eigenvalue of $(L + \epsilon I)^{-1}$, where ϵ is a small value that makes $L + \epsilon I$ non-singular. Note that the matrices L and $(L + \epsilon I)^{-1}$ share the same set of eigenvectors and $\lambda' = \dfrac{1}{(\lambda + \epsilon)}$, where λ and λ' are eigenvalues of L and $(L + \epsilon I)^{-1}$ respectively.

We now show how one can use the power method to find the largest few eigenvalues of

$(L+\epsilon I)^{-1}$. Note that the power method starts with an initial guess v_0, and in every iteration refines the guess by computing

$$v_{i+1} = \frac{(L+\epsilon I)^{-1} v_i}{|(L+\epsilon I)^{-1} v_i|}.$$

Since it may be difficult to solve a linear system $(L+\epsilon I)v_{i+1} = v_i$ implicitly, or to compute the inverse of the matrix $(L+\epsilon I)$ implicitly, we can compute the largest eigenvalue and its corresponding eigenvector of $\lambda_1 I - (L+\epsilon I)$, where λ_1 is the largest eigenvalue of L. Doing the above shift ensures that the smallest eigenvalue of L is the largest in magnitude and the power method finds this. Note that λ_1 can be computed implicitly using the power method, since it is easy to multiply L with a guess vector v;

$$Lv = (D-W)v_i = Dv - Wv.$$

As was shown in Chapter 2, one can easily compute the matrix vector product Wv implicitly, using parenthezation $Wv = B(B^T v)$ in only $O(nk)$ time. The diagonal elements of D can also be computed implicitly as $d_{ii} = (W\mathbf{1})(i)$.

After we obtain the solution x that solves the relaxed cut objective, we can partition the nodes in the network into two clusters – elements with $x(i) < d$ in one cluster and elements with $x(i) \geq d$ in another, for some constant d. A solution to the original objective can thus be obtained by using an indicator vector y with

$$y(i) = \begin{cases} \sqrt{\frac{c}{nc'}} & \text{if } x(i) \geq d \\ -\sqrt{\frac{c'}{nc}} & \text{if } x(i) < d. \end{cases}$$

Recall that $c = |C|$ and $c' = |\overline{C}|$ are the number of nodes on each side of the cut. We often get a reasonable partition if we choose $d = 0$. However, there are $n-1$ different cut-off points; each that puts the first i nodes into one cluster and the remaining $n-i$ into another. Recomputing the ratio cut objective each time takes $O(nk)$ time. But we can do better by calculating these objectives iteratively; we start from a cutting point with the smallest $x(i)$ and iteratively include the next smallest node into the cluster. For brevity, we only show how to recompute $y^T W y$. This can be easily extended for the Laplacian matrix (which includes D) and other matrices that are obtained

from the spectrum of W.

Suppose that we compute $y'^T W y'$ for some indicator vector y' based on some cut-off point described above. Let us move the cut-off point to include a node i into the set C and obtain another indicator vector y. We can easily compute the value $y^T W y$ quickly by subtracting out the contribution of the i^{th} node in $y'^T W y$ and adding it to $y^T W y$, and re-normalizing the terms. That is,

$$\begin{aligned} y^T W y = \ (y^T B)(B^T y) &= \left[\frac{\left(y'^T W y' - y'(i) b_i \cdot y'(i) b_i\right)}{\left(\frac{1}{|C'|} + \frac{1}{|\overline{C'}|}\right)} + y(i) b_i \cdot y(i) b_i \right] \left(\frac{1}{|C|} + \frac{1}{|\overline{C}|}\right) \\ &= \left(y'^T W y' - y'(i)^2 ||b_i||^2\right) \tau + y(i)^2 ||b_i||^2 \left(\frac{1}{|C|} + \frac{1}{|\overline{C}|}\right) \end{aligned}$$

where $\tau = \left(\dfrac{\dfrac{1}{|C|} + \dfrac{1}{|\overline{C}|}}{\dfrac{1}{|C'|} + \dfrac{1}{|\overline{C'}|}} \right)$ renormalizes the terms. Note that if $|C| = c$, then $|\overline{C}| = n - c$, $|C'| = c - 1$ and $|\overline{C'}| = n - c + 1$. The above method takes only $O(k)$ time per iteration, so $O(nk)$ time to try all cut-off points and choose the best one.

4.1.2 Normalized Cuts

Another balanced partition can be obtained by using normalized cuts, proposed by Shi and Malik [100]. These have widespread applications in image segmentation [100, 29, 40, 25, 26, 107], and also in graph clustering algorithms. Normalized cut partitions a graph into two disjoint sets much like ratio cut, only instead of normalizing by $|C|$, it uses the volume of the sets. The volume of a set C is defined as $\text{vol}(C) = \sum_{i \in C} s(i)$, where $s(i)$ is the strength of node i. This gives the total weight of edges associated with all the nodes in the set C, counting all edge weights within C twice and edge weights that form $\text{cut}(C, \overline{C})$ once. The normalized cut is the solution to

$$\min_C \left\{ \frac{\text{cut}(C, \overline{C})}{\text{vol}(C)} + \frac{\text{cut}(C, \overline{C})}{\text{vol}(\overline{C})} \right\}.$$

Shi and Malik [100] showed that this is equivalent to

$$\min_{C} \left\{ \frac{\text{cut}(C, \overline{C})}{\text{vol}(C)} + \frac{\text{cut}(C, \overline{C})}{\text{vol}(\overline{C})} \right\} = \min_{\substack{y \in \{1, -b\}^n \\ y^T D\mathbf{1}=0}} \left\{ \frac{y^T(D-W)y}{y^T D y} \right\} \tag{4.1}$$

where y is an indicator vector

$$y(i) = \begin{cases} 1 & \text{, if } i \in C \\ -\dfrac{\sum_{i \in C} s(i)}{\sum_i s(i)} & \text{, if } i \in \overline{C} \end{cases}$$

and $b = \dfrac{\dfrac{\sum_{i \in C} s(i)}{\sum_i s(i)}}{1 - \dfrac{\sum_{i \in C} s(i)}{\sum_i s(i)}}$. Finding an indicator vector y that solves the above minimization problem is NP-Hard. As with ratio cuts, we can relax the above problem to find a vector $y \in \mathbb{R}^n$

$$\min_{\substack{y \in \mathbb{R}^n \\ y^T D\mathbf{1}=0}} \left\{ \frac{y^T(D-W)y}{y^T D y} \right\}.$$

To understand the above objective, we need to understand the generalized eigenvalue problem. The generalized eigenvalue problem finds eigenvectors x and corresponding eigenvalues λ that satisfy the following equation for matrices A and B

$$Ax = \lambda Bx.$$

If B is non-singular, then the generalized eigenvalue problem is the same as a standard eigenvalue problem

$$B^{-1}Ax = \lambda x.$$

Note that if B is the identity matrix, then the above equation solves an ordinary eigenvalue problem $Ax = \lambda x$. The Rayleigh quotient of a vector is very closely related with the generalized eigenvalue problem. For any vector x, and symmetric real matrices A and B, the Rayleigh quotient $r(x)$ is

defined as

$$r(x) = \frac{x^T Ax}{x^T Bx}.$$

Determining the extremum points, we solve $\bigtriangledown r(x) = 0$:

$$\bigtriangledown r(x) = \frac{2Ax(x^T Bx) - 2(x^T Ax)Bx}{(x^T Bx)^2} = 0.$$

Solving the above equation, we get the generalized eigenvalue problem $Ax = r(x)Bx$. That is, the extremum points of the Rayleigh quotient are the eigenvalues of the generalized eigenvalue problem.

From (4.1), it can be seen that the normalized cut objective can be solved by a generalized eigenvalue problem

$$(D - W)y = \lambda Dy. \tag{4.2}$$

For a vector $z = D^{\frac{1}{2}}y$, the above equation can be reduced to the standard eigenvalue problem

$$D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}z = \lambda z.$$

Note that the matrix $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ is positive semi-definite, since the Laplacian matrix $L = D - W$ is positive semi-definite. Once the eigenvector-eigenvalue pair (z, λ) of the above matrix is found, one can find the corresponding vector $y = D^{-\frac{1}{2}}z$. Suppose that the vectors $y_1 \ldots y_n$ are associated with eigenvectors $z_1 \ldots z_n$ corresponding to eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Note that $y_1 = \mathbf{1}$ is associated with $\lambda_1 = 0$, since we have $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}z_1 = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}D^{\frac{1}{2}}\mathbf{1} = D^{-\frac{1}{2}}(D - W)\mathbf{1}$, and $(D - W)\mathbf{1}$ is the sum of the value of the Laplacian matrix which is 0. Also, all the eigenvectors z are orthonormal. The second eigenvector z_2 satisfies the equation $z_2^T z_1 = 0$ and therefore satisfies the constraint $y_2^T D\mathbf{1} = 0$. Thus z_2 is the solution to the relaxed normalized cut minimization problem.

The second smallest eigenvector z_2 of $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ can be computed implicitly using the power method, since multiplying this matrix with another guess vector x can be performed implicitly by parenthezation $D^{-\frac{1}{2}}((D - W)(D^{-\frac{1}{2}}x))$. As discussed in the previous section, we can multiply the Laplacian matrix $D - W$ with another vector implicitly.

As with ratio cuts, we can partition the network into two sets based on a cut-off point d, and we can minimize the normalized cut objective over all the $n-1$ cut-off points, which can all be computed in only $O(nk)$ time (see Section 4.1.1.2).

4.1.3 Modularity

In this section, we discuss modularity, a measure for detecting and characterizing community structures in networks. A community structure is a division of the network into sets of nodes or clusters such that the connection within each cluster is dense and the connections between clusters is sparse. This is slightly different than a cut. While cut objectives minimize the edge weights crossing the cut, community structures are based on characterizing denseness of connections within clusters and sparseness of connections between clusters. One application is to reveal topically related documents among thousands of documents in the world-wide web [43]. Other applications are in sociological and biological networks [47, 116, 56, 91, 18].

Modularity was first proposed by Newman and Girvan [85], based on the notion that a community structure or a cluster is one that contains more connections within the cluster than expected in a corresponding null model of the network. A null model is another network that preserves the weight distribution of the edges around nodes, and one in which edges are placed at random. We produce a common derivation of modularity for weighted networks.

Let C_i be a cluster to which node i belongs to. Let

$$[C_i = C_j] = \begin{cases} 1 & \text{if } C_i = C_j \\ 0 & \text{otherwise.} \end{cases}$$

Using the above predicate, we can express the total fraction of weights within a cluster as

$$\frac{1}{2}\sum_{i,j} w_{ij}[C_i = C_j]. \tag{4.3}$$

We divide the sum by 2 to remove double counting of edges. For unweighted networks, a null model is formed by repeatedly adding an edge between two randomly chosen nodes with probability proportional to their degree. Extending this to weighted networks, we add a small weight ϵ between two randomly chosen nodes i and j proportional to their strengths. Suppose we do this process T

times. Let R_t be a random variable that describes the total weight of edges between nodes i and j.

$$\mathrm{E}[X_{ij}] = \sum_{t}^{T} \mathrm{E}[R_t]$$

where

$$R_t = \begin{cases} \epsilon & \text{with probability } \dfrac{s(i)s(j)}{4\mathrm{sum}(W)^2} \\ 0 & \text{otherwise.} \end{cases}$$

One typically stops the above random process when the total weight of all edges equals $\mathrm{sum}(W)$. So

$$\begin{aligned} \mathrm{E}[X_{ij}] &= \mathrm{sum}(W)\frac{s(i)s(j)}{4\mathrm{sum}(W)^2} \\ &= \frac{s(i)s(j)}{4\mathrm{sum}(W)}. \end{aligned}$$

If s is a vector of node strengths, then the right hand side of the above expression can be written in matrix form as $\dfrac{1}{4\mathrm{sum}(W)} ss^T$. We can now formulate the expected number of edges within a cluster as

$$\frac{1}{2}\sum_{i,j} \frac{s(i)s(j)}{2\mathrm{sum}(W)}[C_i = C_j].$$

Subtracting this from (4.3), we get the formula for modularity as

$$\begin{aligned} \text{modularity} &= \frac{1}{2}\sum_{i,j} w_{ij}[C_i = C_j] - \frac{1}{2}\sum_{i,j} \frac{s(i)s(j)}{2\mathrm{sum}(W)}[C_i = C_j] \\ &= \frac{1}{2}\sum_{i,j} \left(w_{ij} - \frac{s(i)s(j)}{2\mathrm{sum}(W)} \right) [C_i = C_j]. \end{aligned}$$

We can normalize the modularity by the total weight of all the edges

$$\text{modularity} = \frac{1}{2\mathrm{sum}(W)}\sum_{i,j} \left(w_{ij} - \frac{s(i)s(j)}{2\mathrm{sum}(W)} \right) [C_i = C_j]. \tag{4.4}$$

A partition of the network that has positive modularity is a prospective choice for a good pairwise clustering, since the number of edges within clusters exceeds the expected number of edges just on

the basis of random chance. Subsequently, negative values of modularity for every partition indicates the inability of the network to break down into any number of communities. Finding an optimal partition that maximizes modularity is NP-Hard [20]. Some of the combinatorial methods in the literature look for partitions with positive modularity [50, 33]. Others have used different heuristic approaches [82, 38, 3, 31, 28, 110]. Newman [83] gave a fast heuristic based on spectral methods by defining a modularity matrix M, where

$$m_{ij} = \ w_{ij} - \frac{s(i)s(j)}{2\text{sum}(W)}.$$

In matrix form, $M = W - \frac{1}{2\text{sum}(W)} ss^T$. Let $x \in \{-1, +1\}^n$ be an indicator vector where $x(i) = 1$ indicates that the i^{th} node belongs to cluster C and -1 otherwise. Observe that $[C_i = C_j] = \frac{1}{2}(x_i x_j + 1)$. The formula for modularity can be rewritten as

$$\text{modularity} = \ \frac{1}{4\text{sum}(W)} \sum_{i,j} m_{ij}(x_i x_j + 1).$$

One can easily show that $\sum_{i,j} m_{ij} = 0$. Therefore, modularity can be expressed in matrix form as

$$\text{modularity} = \ \frac{1}{4\text{sum}(W)} x^T M x.$$

The denominator $4\text{sum}(W)$ is a normalizing term. Therefore, finding the partition that maximizes modularity is the solution to

$$\max_{x \in \{-1,+1\}^n} \left\{ x^T M x \right\}.$$

Recall that the above optimization problem is NP-Hard. However, we get a reasonable heuristic if we relax the constraint that $x \in \mathbb{R}^n$ and $||x|| = 1$. We then get the relaxed problem

$$\max_{\substack{x \in \mathbb{R}^n \\ ||x||=1}} \left\{ x^T M x \right\}.$$

The solution is to find the eigenvector corresponding to the largest eigenvalue of the modularity matrix. This can be computed implicitly using the power method, since we can easily multiply the

modularity matrix by any vector x, by multiplying each of the components Wx and $\frac{s(i)s(j)}{2\text{sum}(W)}x = \frac{1}{2\text{sum}(W)}ss^Tx$ separately. After the solution x is obtained, it is required to interpret the vector as an approximate solution to the original modularity problem, which also provides a means to partition the nodes into two clusters. Nodes that have positive values in x can be put into one cluster, and negative values in another. One can also use different cut-off points as discussed in Section 4.1.1.2 and take the partition for which modularity is the largest.

Newman [83] also proposed a hierarchical clustering algorithm based on repeatedly partitioning a subgraph G of a network into two pieces based on the modularity of the subgraph. It is not enough to simply remove the edges that cross the partition and treat either subgraph as independent from each other since this changes the node strengths of each node as given in (4.4). Let $B^{[G]}$ be the data matrix with feature vectors of nodes only in $n^{[G]} \times n^{[G]}$ subgraph $W^{[G]} = B^{[G]}B^{[G]T}$. The modularity of the subgraph is

$$M_{ij}^{[G]} = W_{ij}^{[G]} - \frac{s(i)s(j)}{2\text{sum}(W)} - [i=j]\left(s^{[G]}(i) - s(i)\frac{\text{sum}(W^{[G]})}{2\text{sum}(W)}\right)$$

where $s^{[G]}(i)$ is the node strength of i in $W^{[G]}$. Note that $\text{sum}(W^{[G]}) = \sum_i s^{[G]}(i)$. The subgraph $W^{[G]}$ can be stored implicitly the same way as W, and all other quantities in the above equation can be computed implicitly. The modularity of the subgraph is then $x^TM^{[G]}x$. As long as the modularity of the subgraph is greater than 0, then we have a subgraph that contains more than expected edges in a null model, and can be further subdivided.

Gomez et al. [49] generalized modularity for negative correlation networks by looking at positive and negative edges separately. This, however, is not possible in our implicit setting because it is difficult to find the strengths of positive or negative edges for each node without storing the entire weight matrix explicitly.

4.1.4 Eigen Gap

This is very closely related to ratio and normalized cut objectives, and to a well-studied problem called conductance. Conductance was first introduced by Jerrum and Sinclair [58] and is used to bound the convergence rate of a Markov chain [58, 79, 68]. The conductance of a cut gives the probability of a random walk to transition from the small side of the cut to the other side, and

is defined as

$$\phi(C, \overline{C}) = \frac{\text{cut}(C, \overline{C})}{\min\{\text{vol}(C), \text{vol}(\overline{C})\}}$$

If the conductance of a cut is small, then it provides a possible partition of the network into two clusters, and if it is large, then the graph is well connected between the two sides of the cut. The conductance of the entire graph is the minimum conductance over all possible cuts

$$\phi_G = \min_C \{\text{cut}(C, \overline{C})\}.$$

A graph has high conductance if a random walk tends to jump very often between clusters, and small conductance if it is more likely to stay trapped within some cluster. Computing the optimal value of conductance of a graph is NP-Hard [45] and several approximation algorithms exist [8, 9, 88, 10, 67]. Using Cheeger's inequality, we can get a bound on ϕ_G (see [5, 4, 79, 108]);

$$2\phi_G \geq 1 - \lambda_2 \geq 1 - \sqrt{1 - \phi_G^2} \geq \frac{\phi_G^2}{2}.$$

where λ_2 is the second largest eigenvalue of the walk matrix WD^{-1}, which is a matrix that describes the transition probabilities of a single step in a random walk. Note that the largest eigenvalue of the walk matrix is 1, and the quantity $1 - \lambda_2$ is called the eigengap. Therefore, the eigengap can be used as an approximate solution to the conductance of a graph. Using the power method, λ_2 can be computed implicitly.

The eigengap is also very closely related to the mixing time of a Markov chain [101], which is the time it takes for a Markov chain to converge to its steady state distribution. If the eigengap is small, then a random walk is more likely to be trapped within a small subset of nodes and therefore has slower mixing time. On the other hand, if eigengap is large, then a random walk moves freely between many nodes in the graph which results in faster mixing time. In fact, a Markov Chain has rapid mixing time if and only if $\phi_G \geq \frac{1}{\text{poly}(n)}$ (see [101]).

4.2 Negative Dot Product Networks

In this section, we discuss two clustering algorithms that are essentially equivalent. We start with a discussion on correlation clustering which tries to find clusters in correlated data and then show how it is mathematically equivalent to principal component analysis (PCA). PCA is used to reduce the dimensionality of a data set by projecting into a set of orthonormal principal components that maximizes the variance of the original data.

4.2.1 Correlation Clustering and Principal Component Analysis

Introduced by Bansal et al. [14], *correlation clustering* tries to cluster nodes based on a similarity function f where nodes in clusters are highly similar with each other. This can be viewed as either a maximization problem: maximize the total weight of positive edges within clusters and the total weight of absolute values of negative edges between clusters, or as a minimization problem: minimize the total weight of positive edges between clusters and the absolute value of the negative edge within clusters. Though some of the well-known clustering algorithms require the number of clusters as input, correlation clustering conveniently does not [14]. This problem was shown to be NP-Hard [14] and several approximation algorithms exist [14, 48, 36, 39, 106, 35].

Formalizing the above maximization objective, we would like to find a partition of the graph into two clusters so as to solve

$$\max_{C} \left\{ \sum_{i,j} w_{ij}[C_i = C_j] - \sum_{i,j} |w_{ij}|[C_i \neq C_j] \right\} = \max_{x \in \{-1,+1\}^n} \left\{ \sum_{i,j} w_{ij} x_i x_j \right\}$$

where x is an indicator vector where $x(i) = +1$ if node $i \in C$, -1 otherwise. The above optimization problem can be expressed in matrix quadratic form

$$\max_{x \in \{-1,+1\}^n} \left\{ x^T W x \right\}.$$

This contributes $+w_{ij}$ if both i and j are in the same cluster, and $-w_{i,j}$ if they are in different clusters. The above objective gives credit for high positive edges within clusters and high negative edges between clusters. Since solving for the optimal vector x is NP-Hard, we can relax the problem

to

$$\max_{\substack{x \in \mathbb{R}^n \\ ||x||=1}} \left\{ x^T W x \right\}.$$

As seen before, the solution to the relaxed objective is the largest eigenvector of W, which can be easily computed implicitly. To form the actual clusters, we could use a cut-off point of 0 and put elements less than zero in one cluster and elements bigger in another cluster, or use the other $n-1$ cut-off points discussed in Section 4.1.1.2.

The largest eigenvector of W has a nice interpretation from a one-dimensional embedding of a network using principal component analysis (PCA). As briefly discussed in Chapter 2, the data matrix B can be projected onto a set of top eigenvectors of the covariance matrix $S = \frac{BB^T}{(n-1)}$ using the transformation

$$B' = EB$$

where the $d \times k$ matrix E contains the top d eigenvectors of S along its rows. Implicitly, we can obtain E after computing the spectrum of $W = BB^T$ in $O(nk^{\omega-1})$ time. If we we only project to the largest eigenvector of W, which can be computed using the power method, we get a one-dimensional

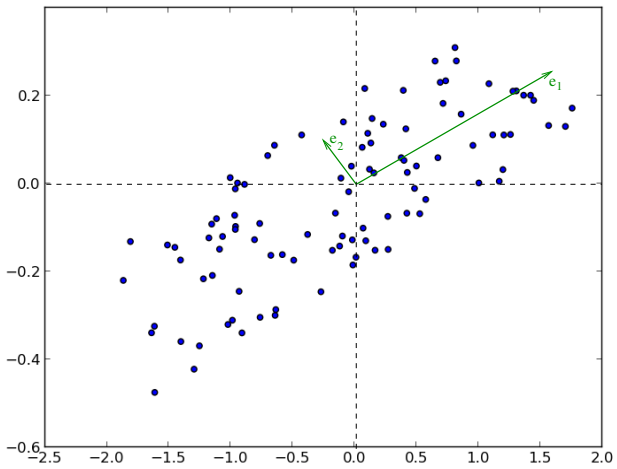

Figure 4.1: Principal component interpretation of the largest eigenvector of a negative correlation network.

embedding of the data set. In Chapter 3, we saw how the largest eigenvector gives us a measure of centrality for non-negative dot product networks. We now give an interpretation of the largest eigenvector of in terms of clustering for negative correlation networks.

Figure 4.1 gives a set of points in 2-dimensional space. The principal components e_1 and e_2 are shown. The largest eigenvector e_1 can be considered as the axis along which the points "stretch" the maximum. Note that the projection of points onto e_1 divides the point set into two clusters – nodes with positive projection in one set, and nodes with negative projection in another set. In Euclidean space, if we consider points that are very close to each other "similar" in some way, then each cluster contains points that are similar or well correlated with each other, while points in different clusters are dissimilar or anti-correlated.

Chapter 5

Locality Sensitive Hashing

In the preceding chapters, we saw how various centrality and cut algorithms can be computed implicitly in a dot product network. One of the main challenges in applying these techniques is that most centrality and cut metrics are defined only for non-negative networks. Correlation networks may contain negative edges, and there are many ways to handle them. Negative edges may be eliminated by zeroing them out completely, or zeroing edges that are less than a particular threshold. This is difficult to do in our implicit setting, as it changes the spectrum of the weight matrix unpredictably besides requiring to have an explicit representation to perform the thresholding. If negative edges are not interesting, we can translate the range of correlations from $[-1, +1]$ to the range $[0, 2]$ by the function $1 + \text{corr}(.)$. This is very trivially accomplished implicitly by adding an extra column to the data matrix B with all ones. The new dot product $b'_i \cdot b'_j$ then becomes $\sum_{l=1}^{k+1} b'_i(l)b'_j(l) = \sum_{l=1}^{k} b'_i(l)b'_j(l) + 1 = \text{corr}(b_i, b_j) + 1$. However, the fact that zero becomes one distorts many common metrics, and this approach is not widely used.

In most correlation analysis, it is preferred to treat both highly correlated and highly anti-correlated edges equally, since this still provides a measure of linear dependence between the random variables. In such situations, analysis is performed with either absolute values or squares of the correlations. In this chapter, we first show how a non-negative dot product network can be obtained with squares of correlations using analysis mathematically similar to a well-known "kernel trick". Though this approach preserves squares of original correlations exactly, it maps all feature vectors to $O(k^2)$ dimensions. We then show how a family of locality sensitive hash (LSH) functions can be used

to acquire a mapping that approximately preserves the squares of the correlations in expectation, which can potentially allow us to reduce the number of dimensions significantly, albeit with a slight increase in computational time. The LSH transformation can be similarly applied to approximately preserve in expectation the absolute values of correlations or actual correlations for non-negative networks. Recall that we can use the dimensionality reduction using random projections discussed in Chapter 2 if we need a dot product network that preserves correlations in expectation. However, this might yield negative numbers, even if the original network was non-negative, while the output of our LSH approach is non-negative.

Finally, we look at how to express edge weights as coherence. The coherence between two random variables may be thought of as correlations in the frequency domain and gives the amount of phase consistency that exists between the corresponding time signals. We show how to express a coherence network as a dot product network.

5.1 Exact Mapping For Squared Correlations

A technique found in machine learning and other related areas, known as the "kernel trick", is used to implicitly operate on dot products of feature vectors after mapping the vectors to higher dimensions. This is usually done to transform a possibly non-linear relationship in data into a linear relationship in higher dimensions. For example, Figure 5.1(a) contains points in two dimensions that are linearly inseparable. A machine learning classifier like the support vector machine may not be able to classify the points accurately, since there is no linear boundary that separates the dataset. However, if the data points are projected to three dimensions along a paraboloid (Figure 5.1(b)), we find that there is enough gap between the two datasets to perform binary classification. Kernels are functions that compute dot products *implicitly* in higher dimensions without having to explicitly perform the higher dimensional transformation. Formally, for feature vectors u and v and function $f : \mathbb{R}^k \to \mathbb{R}^d$, a kernel is a function $j : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ that implicitly computes the dot products between two feature vectors in higher dimensions; that is, $j(u, v) = f(u) \cdot f(v)$. Among many kernel functions, the polynomial kernel is widely used, and is defined as $j(u, v) = (\alpha(u \cdot v) + \beta)^p$ where α and β are constant parameters of the function and p is the degree of the polynomial. Note that polynomial kernels operate on feature vectors in $d = O(k^p)$ dimensions, and thus are particularly useful in situations when such an explicit transformation is intractable. A small caveat is that

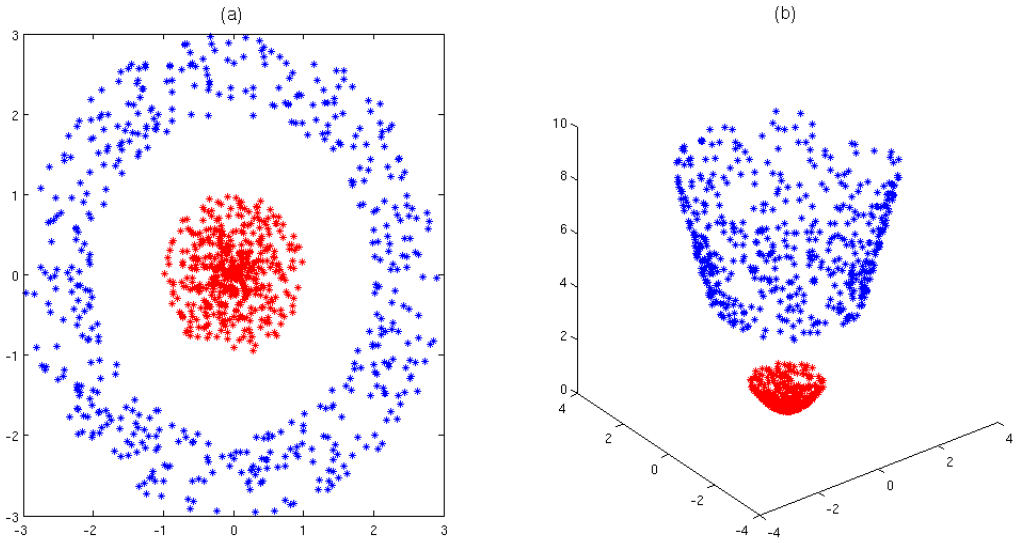

Figure 5.1: (a) A two dimensional linearly inseparable dataset. (b) The dataset projected into three dimensions along a paraboloid $z = x^2 + y^2$ becomes linearly separable.

it may not be easy to fine-tune the parameters α and β for a specific function f (see [98]). We show a mapping f that transforms feature vectors in k dimensions to $O(k^2)$ dimensions that exactly preserves the squares of the correlations. Normally, this mapping would be done implicitly; however, we need the feature vectors expressed explicitly to compute various analytics.

For any length-k feature vector u, the mapping $f : \mathbb{R}^k \to \mathbb{R}^{O(k^2)}$ is defined as

$$f(u) = [u(1)^2, u(2)^2, \ldots, u(k)^2, \sqrt{2}u(1)u(2), \sqrt{2}u(1)u(3), \ldots].$$

The mapping $f(u)$ is a length $O(k^2)$ feature vector with the first k terms $u(i)^2$ and the next $\binom{k}{2}$ terms the pairwise products $\sqrt{2}u(i)u(j)$, for $i \neq j$ and $1 \leq i, j \leq k$. For any two feature vectors u and v and a polynomial kernel j, we have the squares of the correlation to be

$$j(u, v) = (u \cdot v)^2 = \left(\sum_i u(i)v(i) \right)^2 = \sum_i u(i)^2 v(i)^2 + 2 \sum_{i,j} u(i)^2 v(j)^2 = f(u) \cdot f(v).$$

Thus, the mapping f preserves the squares of the correlations exactly. If D is an $n \times O(k^2)$ matrix with feature vectors of the mapping $f(u)$ along its rows, then the dot product network $V = DD^T$ is non-negative, and represents the squares of the correlations $W = BB^T$. Equivalently $V = W^{(2)}$,

but we cannot store this explicitly. Storing D allows us to perform the power method implicitly in $O(nk^2)$ time per iteration. It takes $O(nk^{2(\omega-1)}+k^6)$ time to compute the entire spectrum of V, since it takes $O(nk^{2(\omega-1)})$ time to compute D^TD and then $O(k^6)$ time to compute its entire spectrum (see Section 2.3). If k gets too large, the $O(k^6)$ part becomes an issue, and therefore the above approach can only be efficiently used for about $k \leq 30$ in practice. We could use random projections discussed in Section 2.6 to reduce the number of dimensions to a more manageable level. This however, does not preserve non-negativity. It is possible to reduce the number of dimensions to d by applying an LSH transformation very similar to the one described in the next section, which approximately preserves the dot products in expectation and which preserves non-negativity. The computing time to make the transformation is $O(nk^{\omega-1}d)$ time using block multiplication discussed in Section 2.2, which can be performed even without having to store D explicitly, since the dot product between a length-k^2 random vector r with $r(i) \sim N(0,1)$ and $f(u)$ can be performed without generating $f(u)$ – by computing u^TRu where R is a $k \times k$ matrix that contains all the elements of r.

5.2 Approximate Mapping Using LSH

In the last section, we saw how to preserve squares of correlations exactly. A only drawback is that we are mapping the feature vectors to $O(k^2)$ dimensions. In this section, we use another approach using a family of LSH functions to reduce the feature vectors to d dimensions that allows us to create a non-negative dot product network that approximately preserves the squares of the dot products in expectation. The absolute values of dot products (actual dot products in case of non-negative dot product networks) can also be approximately preserved using a similar approach.

Let f be a single variable non-negative-valued function. A family of locality sensitive hash functions H can be defined by

$$h(u) = \frac{f\left(\frac{u \cdot r}{||u||}\right)}{\sqrt{\mathrm{E}\left[f(x)^2\right]}} \tag{5.1}$$

where $h \in H, h : \mathbb{R}^k \to \mathbb{R}$, u is a length-k feature vector, $x \sim N(0,1)$ is normally distributed and r is a length-k random vector with $r(i) \sim N(0,1)$. The denominator is used for normalization, while the numerator is a random projection of u onto a random vector[1] r which is then fed into

[1]We get similar results when we normalize r and use $f(\sqrt{k}\frac{u \cdot r}{||u||||r||})$, where in the limit as $k \to \infty$, the quantity $\frac{\sqrt{k}}{||r||}$

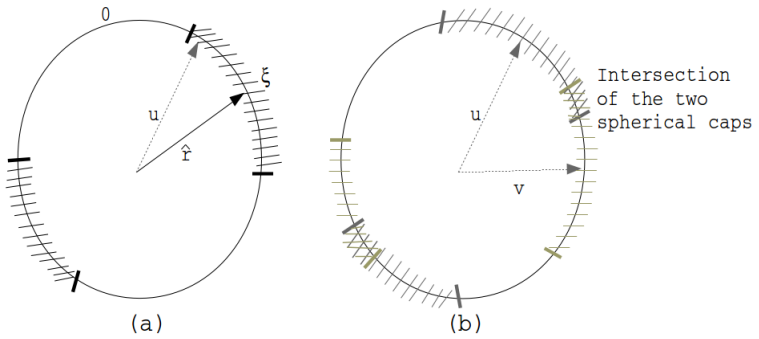

Figure 5.2: (a) The regions for which the hash function $h(u) = \xi$ for a specific random vector $\widehat{r}$. (b) The area of the two spherical caps gives the probability that the hashes of both vectors u and v are ξ.

f. To get good approximations of $(u \cdot v)^2$, we require that f is symmetric about $f(x) = 0$. For example, if f is the step function, which takes a value ξ if the absolute value of x is greater than a specified threshold, the output of $h(u)$ is ξ if the angle between u and r lies in the shaded region of Figure 5.2(a). Two feature vectors u and v collide if r is chosen to be in the intersection of the two arcs shown in Figure 5.2 (b). Using d of these hash functions, we can obtain a feature vector $u' = [h_1(u), h_2(u), \ldots, h_d(u)]$. We would like the dot product $\frac{1}{d}u' \cdot v'$ to have expected value approximately $(u \cdot v)^2$. Now, $\frac{1}{d}u' \cdot v' = \dfrac{1}{d}\displaystyle\sum_{i=1}^{d} h_i(u)h_i(v)$. Therefore, it is enough if we show each of the hashes h_i preserves the squares of the dot products in expectation[2], that is, $\mathrm{E}\,[h(u) \cdot h(v)] \approx (u \cdot v)^2$. In Figure 5.4, we have plotted several examples of $\mathrm{E}\,[h(u) \cdot h(v)]$ verses $u \cdot v$. This "LSH curve" should ideally be as close as possible to the plot of $(u \cdot v)^2$.

First, we show that the expected value at the top of the curve is 1, that is, if $(u \cdot v) = 1$, then $\mathrm{E}\,[h(u) \cdot h(v)] = 1$. Since h is rotationally symmetric, without loss of generality we can assume $u = [1, 0, \ldots, 0]$ and $v = [1, 0, \ldots, 0]$. We have

$$\mathrm{E}\,[h(u) \cdot h(v)] = \frac{\mathrm{E}\left[f(r(1))^2\right]}{\mathrm{E}\left[f(x)^2\right]} = 1$$

approaches one. This can be seen from $\frac{||r||^2}{k} = \frac{1}{k}\sum_i r(i)^2$ which is an average of identically distributed (chi-squared in this case) random variables. According to the central limit theorem, this average approaches a normal distribution with variance 0, as $k \to \infty$. Though this approach matches in the limit as $k \to \infty$, for small k it can give a good approximation but needs to be corrected by a multiplicative factor related to the marginal distribution of a spherical distribution, which can be cumbersome to estimate. So this is a plausible approach, but takes more care to ensure that it works properly.

[2]For the sake of presentation, we use the notation h instead of h_i.

since both $r(1)$ and x are normally distributed with variance 1. At the bottom of the curve ($u \cdot v = 0$), we get a slight baseline shift δ, that is, if u and v are orthogonal with each other, we get a slight shift of δ in the expected value. We can easily obtain a formula for the baseline shift. By assuming (again, without loss of generality, due to rotational symmetry) $u = [1, 0, 0, \ldots, 0]$ and $v = [0, 1, 0, 0, \ldots, 0]$, we get

$$\begin{aligned} \mathrm{E}\left[h(u) \cdot h(v)\right] = & \frac{1}{\mathrm{E}\left[f(x)^2\right]} \mathrm{E}\left[f(r(1) f(r(2)))\right] \\ = & \frac{\mathrm{E}\left[f\left(r(1)\right)\right] \mathrm{E}\left[f\left(r(2)\right)\right]}{\mathrm{E}\left[f(x)^2\right]} \\ = & \frac{\mathrm{E}\left[f\left(x\right)\right]^2}{\mathrm{E}\left[f(x)^2\right]} \end{aligned} \tag{5.2}$$

since all the variables $r(1)$, $r(2)$ and x are normally distributed with mean 0 and variance 1, we have $\mathrm{E}[f(r(1)] = \mathrm{E}[f(x)]$.

We now compute the variance at the top of the LSH curve. Recall that $u = v = [1, 0, 0, \ldots, 0]$. We have

$$\begin{aligned} \mathrm{var}(h(u) \cdot h(v)) &= \mathrm{E}\left[(h(u) \cdot h(v))^2\right] - \mathrm{E}\left[h(u) \cdot h(v)\right]^2 \\ &= \frac{\mathrm{E}\left[\left(f(r(1)) f(r(1))\right)^2\right]}{\mathrm{E}\left[f(x)^2\right]^2} - 1 \\ &= \frac{\mathrm{E}\left[f(x)^4\right]}{\mathrm{E}\left[f(x)^2\right]^2} - 1. \end{aligned} \tag{5.3}$$

At the bottom of the LSH curve, recall $u = [1, 0, 0, \ldots, 0]$ and $v = [0, 1, 0, 0, \ldots, 0]$. We have

$$\begin{aligned} \mathrm{var}(h(u) \cdot h(v)) &= \frac{\mathrm{E}\left[\left(f(r(1)) f(r(2))\right)^2\right]}{\mathrm{E}\left[f(x)^2\right]^2} - \left(\frac{\mathrm{E}\left[f\left(x\right)\right]^2}{\mathrm{E}\left[f(x)^2\right]}\right)^2 \\ &= \frac{\mathrm{E}\left[f(r(1))^2\right] \mathrm{E}\left[f(r(2))^2\right]}{\mathrm{E}\left[f(x)^2\right]^2} - \left(\frac{\mathrm{E}\left[f\left(x\right)\right]^2}{\mathrm{E}\left[f(x)^2\right]}\right)^2 \\ &= \frac{\mathrm{E}\left[f(x)^2\right]^2}{\mathrm{E}\left[f(x)^2\right]^2} - \left(\frac{\mathrm{E}\left[f\left(x\right)\right]^2}{\mathrm{E}\left[f(x)^2\right]}\right)^2 \\ &= 1 - \left(\frac{\mathrm{E}\left[f\left(x\right)\right]^2}{\mathrm{E}\left[f(x)^2\right]}\right)^2. \end{aligned}$$

Thus we have generic formulas for the baseline shift, and the mean and variance at both the top

and bottom of the LSH curve. To get a generic formula for the equation of the curve, we set $u = [1, 0, \ldots, 0]$ and $v = [\cos\phi, \sin\phi, 0, \ldots, 0]$ (again, due to rotational symmetry of h). We get

$$\mathrm{E}\left[h(u) \cdot h(v)\right] = \frac{\mathrm{E}\left[f(r(1))f(r(1)\cos\phi + r(2)\sin\phi)\right]}{\mathrm{E}\left[f(x)^2\right]}. \tag{5.4}$$

To get good approximations of $(u \cdot v)^2$, we can make use of a number of non-negative functions f. We discuss three of these functions that give good approximations.

1. $f(x)$ is a step function that we define by

$$x = \begin{cases} \xi & \text{if } |x| \geq \rho \\ 0 & \text{otherwise} \end{cases}$$

where ρ is a constant that can be used to parameterize the function, and ξ is a constant that depends on ρ which we discuss below. Note that, while computing the hash function in Equation (5.1), we will evaluate x at the point $\frac{u \cdot r}{||u||}$. This can be interpreted for two dimensions from Figure 5.2(a), where for a specific random unit vector[3] $\widehat{r}$, the function f takes the value ξ if the angle between u and r is within the shaded region. For two vectors u and v, the expected dot product of the hashes is

$$\mathrm{E}\left[h(u)h(v)\right] = \xi^2 \Pr\left[h(u) = \xi \;\&\; h(v) = \xi\right].$$

The above probability is the length of the intersection of the two arcs shown in Figure 5.2(b). In higher dimensions, these probabilities can be obtained using complicated formulas for computing surface areas of two spherical caps as given in [72, 75], or through simulation. Using a similar proof to the one shown above, we can also obtain closed form formulas for the baseline shift δ, which is $\Pr\left[r(1) \geq \xi\right] = 1 - \mathrm{cdf}(\xi)$, where cdf is the Gaussian cumulative distribution function which can be expressed in terms of the Gaussian error function $\mathrm{cdf}(x) = \frac{1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$. Note that the error function is defined as $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2} \mathrm{dt}$. Using this, we can obtain an expression for ξ in terms of erf, $\xi = \frac{1}{\sqrt{\Pr\left[r(1) \geq \rho\right]}} = \frac{1}{\sqrt{1 - \mathrm{cdf}(\rho)}} = \sqrt{\frac{2}{1 - \mathrm{erf}(\rho)}}$. The LSH curve (expected value of $h(u) \cdot h(v)$) at any point x can also be obtained, but this involves

[3]The above interpretation can be extended for any random vector r.

integrating Gaussians in polar coordinates,

$$\begin{aligned}
\mathrm{E}\left[h(u)\cdot h(v)\right] = &\ \Pr\left[(r_1, r_2) \in \{\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4\}\right] \\
= &\ \frac{A^2}{2\pi}\left[\int_{\theta=\alpha}^{\frac{\pi}{2}} -\mathrm{e}^{-\frac{B^2}{2\cos^2\theta}}\ \mathrm{d}\theta\ +\ \int_{\theta=\phi-\frac{\pi}{2}}^{\alpha} -\mathrm{e}^{-\frac{B^2}{2\cos^2(\theta-\phi)}}\ \mathrm{d}\theta\right. \\
&\left. +\ \int_{\theta=\frac{\pi}{2}}^{\beta} -\mathrm{e}^{-\frac{B^2}{2\cos^2\theta}}\ \mathrm{d}\theta\ +\ \int_{\theta=\beta}^{\frac{\pi}{2}+\phi} -\mathrm{e}^{-\frac{B^2}{2\cos^2(\theta-\phi)}}\,\mathrm{d}\theta\right]
\end{aligned}$$

which is the integral of a $2D$ Gaussian over the area of the shaded region $\{\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4\}$ given in Figure 5.3, and where $\alpha = \tan^{-1}\left(\frac{1-\cos\phi}{\sin\phi}\right)$, $\beta = \tan^{-1}\left(\frac{1+\cos\theta}{\sin\theta}\right)$ and ϕ is the angle whose cosine is $u \cdot v$.

Figure 5.4(a) shows the actual squared correlations curve in red and the LSH approximation curve in blue. In Figure 5.4(d), for a specific hash vector $r = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$, the white region is the

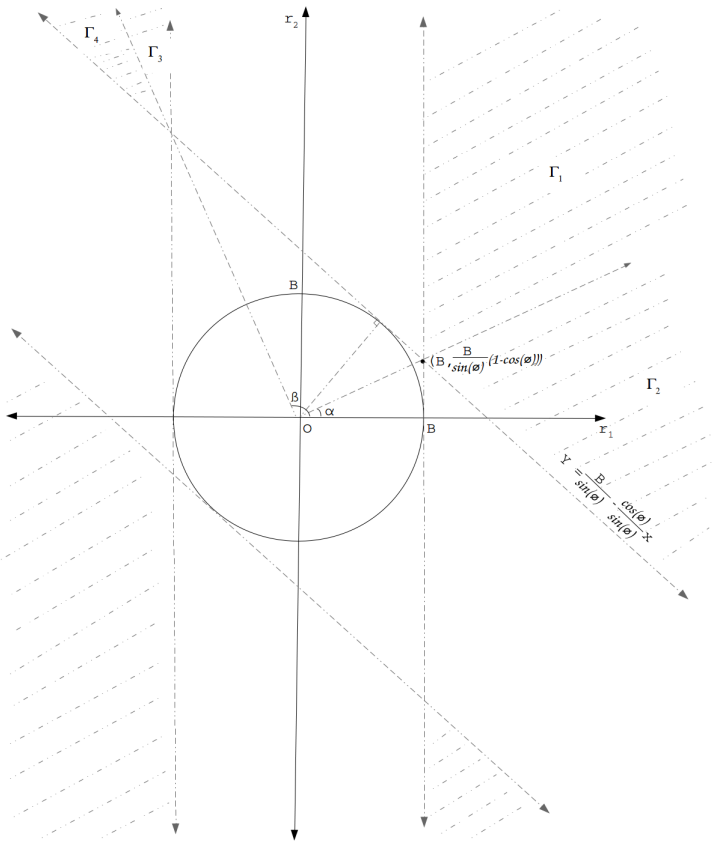

Figure 5.3: The probability determined by the area of the shaded region gives the equation of the LSH curve.

space in which any vector u in the two dimensional plane has a non-zero value for $h(u)$. We can see that two vectors u and v will have the same hash output with r only if they both lie in the white region. Using many of these hash functions, vectors that are highly correlated or anti-correlated with each other are more likely to have a large dot product $h(u) \cdot h(v)$ in expectation, while two vectors that are uncorrelated are more likely to have a small dot product in expectation. The best baseline shift we get is for $\delta = 0.147$, since lowering δ produces a shape that doesn't exactly match the actual curve and gives us greater error, where error is defined as the maximum of the absolute difference of $(u \cdot v)^2$ and $\mathrm{E}\left[h(u) \cdot h(v)\right]$ over all values of $(u \cdot v)$. Performing a numerical analysis, we find that the standard deviation at the top of the curve is about 1.85. To bring this down to 0.1, which is a reduction of 18.4%, we would require $18.4^2 = 339$ dimensions.

2. $f(x) = x^p$ for p even. We can obtain all the required formulas – standard deviation at the top, the baseline shift, and the equation of the curve, since we are only taking higher order moments of a Gaussian. For example, if $p = 4$, from (5.2), we get the baseline shift to be

$$\delta = \frac{\mathrm{E}\left[x^4\right]^2}{\mathrm{E}\left[x^8\right]} = \frac{9}{105} = 0.086$$

and from (5.3), we get the standard deviation at the top of the curve to be

$$\sqrt{\frac{\mathrm{E}\left[x^{16}\right]}{\mathrm{E}\left[x^8\right]^2} - 1} = \sqrt{\frac{2027025}{105^2} - 1} = 13.5.$$

We can derive the equation of the LSH curve from (5.4);

$$\begin{aligned}
\mathrm{E}\left[h(u) \cdot h(v)\right] = & \frac{1}{\mathrm{E}\left[f(x)^2\right]} \mathrm{E}\left[f(r_1) f(r_1 \cos\phi + r_2 \sin\phi)\right] \\
= & \frac{1}{\mathrm{E}\left[x^8\right]} \mathrm{E}\left[r_1^4 \left(r_1 \cos\phi + r_2 \sin\phi\right)^4\right] \\
= & \frac{1}{105} \mathrm{E}\left[r_1^4 \left(r_1^2 \cos^2\phi + r_2^2 \sin^2\phi + 2 r_1 r_2 \cos\phi \sin\phi\right)^2\right] \\
= & \frac{1}{105} \mathrm{E}\Bigg[r_1^4 \Big(r_1^4 \cos^4\phi + r_2^4 \sin^4\phi + 6 r_1 r_2 \cos^2\phi \sin^2\phi \\
+ & \qquad 4 r_1^3 r_2 \cos^3\phi \sin\phi + 4 r_1 r_2^3 \cos\phi \sin^3\phi\Big)\Bigg].
\end{aligned}$$

The last two terms contain products of independent odd moments of Gaussian random vari-

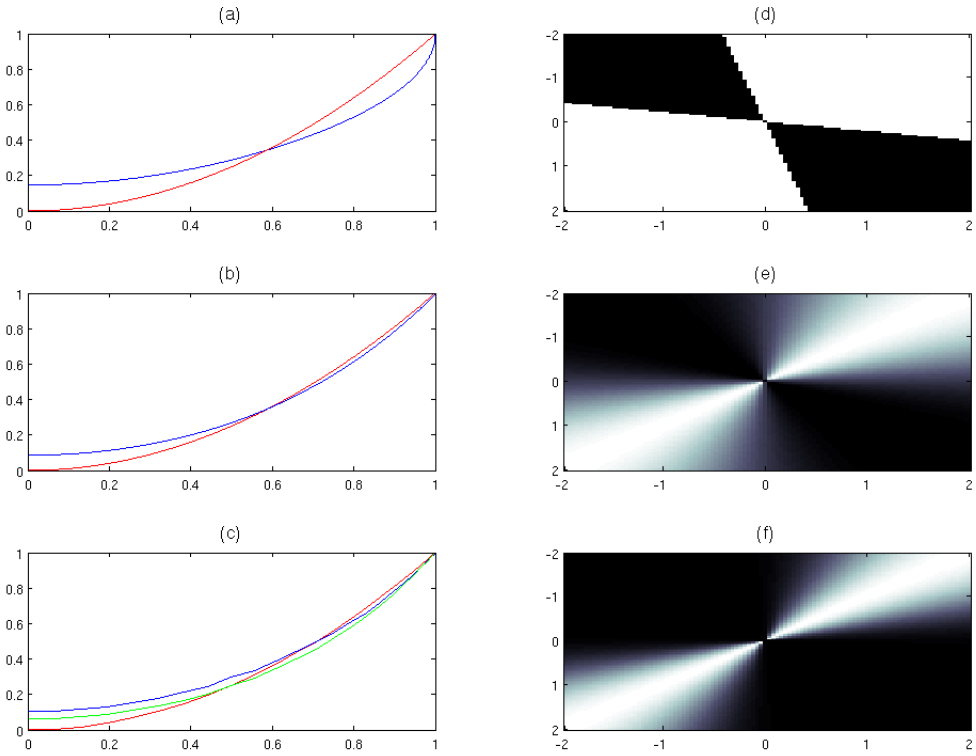

Figure 5.4: Squares of correlations in the range $[0, 1]$ described by the curve in red, and the LSH curve defined by $\mathrm{E}\left[h(u) \cdot h(v)\right]$ in blue using functions (a) $f(x)$, a step function; (b) $f(x) = x^4$; and (c) $f(x) = \mathrm{erf}(\frac{x}{\sqrt{2}})^{18}$. We also show $f(x) = \tanh(\frac{x}{\sqrt{2}})^{18}$ in green that approximates erf. For a hash vector $r = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$, the regions in white show positive admissions with the hash function using (d) $f(x)$, a step function; (e) $f(x) = x^4$; and (f) $f(x) = \mathrm{erf}(\frac{x}{\sqrt{2}})^{18}$.

ables, which is 0 in expectation. The above equation can be simplified as

$$\begin{aligned}
\mathrm{E}\left[h(u) \cdot h(v)\right] &= \frac{1}{105}\mathrm{E}\left[r_1^4\left(r_1^4\cos^4\phi + r_2^4(1-\cos^2\phi)^2 + 6r_1r_2\cos^2\phi(1-\cos^2\phi)\right)\right] \\
&= \frac{1}{105}\mathrm{E}\left[r_1^4\left((r_1^4 + r_2^4 - 6r_1^2r_2^2)\cos^4\phi + (6r_1^2r_2^2 - 2r_2^4)\cos^2\phi + r_2^4\right)\right] \\
&= \frac{1}{105}\left(\mathrm{E}\left[r_1^8 + r_1^4r_2^4 - 6r_1^6r_2^2\right]\cos^4\phi + \mathrm{E}\left[6r_1^6r_2^2 - 2r_2^4\right]\cos^2\phi + \mathrm{E}\left[r_2^4\right]\right).
\end{aligned}$$

Using linearity of expectations and the fact that the expectation of product of independent

random variables is the product of their expectations, we get

$$\begin{aligned}\mathrm{E}\left[h(u)\cdot h(v)\right] &= \frac{1}{105}\left((105+9-6(15)(1))\cos^4\phi+(6(15)(1)-2(9))\cos^2\phi+9\right)\\ &= \frac{1}{105}\left(24(u\cdot v)^4+72(u\cdot v)^2+9\right)\end{aligned}$$

where $\cos\phi=(u\cdot v)$. Though the baseline shift is small, the standard deviation at the top of the curve is huge. It requires $135^2=18825$ dimensions to reduce the standard deviation to 0.1. Using $p=2$ gives a worse baseline shift $\delta=\dfrac{\mathrm{E}\left[x^2\right]^2}{\mathrm{E}\left[x^4\right]}=0.3333$. The standard deviation at the top of the curve is $\sqrt{\dfrac{\mathrm{E}\left[x^8\right]}{\mathrm{E}\left[x^4\right]^2}-1}=0.54$. We note that using $p=6$ doesn't seem to approximate the curve well. Figure 5.4(b) shows the LSH approximation curve (blue) along with the curve that represents actual squared correlations (red). In Figure 5.4(e), we can see that vectors that have high projection along $r=[\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{2}]$ are further magnified by the polynomial function f. Therefore, two vectors that are highly correlated or anti-correlated have high dot products in expectation.

The case $f(x)=x^2$ leads to some interesting analytical results aside from the drawback of having a huge baseline shift. Suppose that the vector u is normalized to unit length, we get the hash function $h(u)=(r\cdot u)^2$, where r is a random vector with $r_i\sim N(0,1)$. Note that $\mathrm{E}\left[r_i\right]=0$ and $\mathrm{E}\left[r_i^2\right]=1$. We have

$$\begin{aligned}\mathrm{E}\left[h(u)\cdot h(v)\right] &= \mathrm{E}\left[\left(\sum_i u_i r_i\right)^2\left(\sum_i v_i r_i\right)^2\right]\\ &= \mathrm{E}\left[\left(\sum_i u_i^2 r_i^2+\sum_{i\neq j}u_i r_i u_j r_j\right)\left(\sum_p v_p^2 r_p^2+\sum_{p\neq q}v_p r_p v_q r_q\right)\right]\\ &= \mathrm{E}\Bigg[\sum_{i,p}u_i^2 v_p^2 r_i^2 r_p^2+\sum_{i,p\neq q}u_i^2 v_p v_q r_i^2 r_p r_q+\sum_{i\neq j,p}u_i u_j v_p^2 r_i r_j r_p^2\\ &\qquad\qquad +\sum_{i\neq j,p\neq q}u_i u_j v_p v_q r_i r_j r_p r_q\Bigg]\\ &= \mathrm{E}\left[\sum_{i,p}u_i^2 v_p^2 r_i^2 r_p^2\right]+\sum_{i,p\neq q}u_i^2 v_p v_q \mathrm{E}\left[r_i^2 r_p r_q\right]+\sum_{i\neq j,p}u_i u_j v_p^2\mathrm{E}\left[r_i r_j r_p^2\right]\\ &\qquad\qquad +\sum_{i\neq j,p\neq q}u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right].\end{aligned}$$

The second and the third terms are 0 since we have products of two independent random variables, each with expectation 0. Expanding the first term, we get

$$\begin{aligned}
\mathrm{E}\left[h(u)\cdot h(v)\right] &= \mathrm{E}\left[\sum_i u_i^2 v_i^2 r_i^4 + \sum_{i\neq p} u_i^2 v_p^2 r_i^2 r_p^2\right] + \sum_{i\neq j, p\neq q} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + \sum_{i\neq p} u_i^2 v_p^2 \mathrm{E}\left[r_i^2\right]\mathrm{E}\left[r_p^2\right] + \sum_{i\neq j, p\neq q} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + \sum_{i\neq p} u_i^2 v_p^2 + \sum_{i\neq j, p\neq q} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + \sum_{i,p} u_i^2 v_p^2 - \sum_i u_i^2 v_i^2 + \sum_{i\neq j, p\neq q} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + ||u||^2||v||^2 - \sum_i u_i^2 v_i^2 + \sum_{i\neq j, p\neq q} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + 1 - \sum_i u_i^2 v_i^2 + \sum_{i\neq j, p\neq q} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + 1 - \sum_i u_i^2 v_i^2 + 2\sum_{(i=p)\neq(j=q)} u_i u_j v_p v_q \mathrm{E}\left[r_i r_j r_p r_q\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + 1 - \sum_i u_i^2 v_i^2 + 2\sum_{i\neq j} u_i u_j v_i v_j \mathrm{E}\left[r_i^2\right]\mathrm{E}\left[r_j^2\right] \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + 1 - \sum_i u_i^2 v_i^2 + 2\sum_{i\neq j} u_i u_j v_i v_j \\
&= \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] + 1 - \sum_i u_i^2 v_i^2 + 2\left((u\cdot v)^2 - \sum_i u_i^2 v_i^2\right) \\
&= 2(u\cdot v)^2 + 1 + \sum_i u_i^2 v_i^2 \mathrm{E}\left[r_i^4\right] - 3\sum_i u_i^2 v_i^2.
\end{aligned}$$

Note that since r's components are Gaussian distributed, $\mathrm{E}\left[r_i^4\right] = 3$ and the last two terms cancel each other, giving $\mathrm{E}\left[\frac{h(u)\cdot h(v)}{2}\right] = (u\cdot v)^2 + \frac{1}{2}$. This is interesting because the expected dot products of the hashes $h(u)$ and $h(v)$ is exactly the squares of the corresponding correlations $(u\cdot v)^2$ with a slight baseline shift. The above derivation is interesting for another fact – r's components may also be chosen randomly from $\{-1, +1\}$. In this case, we have $\mathrm{E}\left[r_i^4\right] = 1$, giving $\mathrm{E}\left[\frac{h(u)\cdot h(v)}{2}\right] = (u\cdot v)^2 + \frac{1}{2} - \sum_i u_i^2 v_i^2$. The last term can be removed by using d extra dimensions for each feature vector u with components $u_i^2, \ldots, u_d^2$. In both cases, the baseline shift is $\frac{1}{2}$ and the value of $\mathrm{E}\left[h(u)\cdot h(v)\right]$ is in the range $[\frac{1}{2}, \frac{3}{2}]$. We can multiply the hash output by $\frac{2}{3}$ to get values in the range $[\frac{1}{3}, 1]$, which gives a baseline shift of

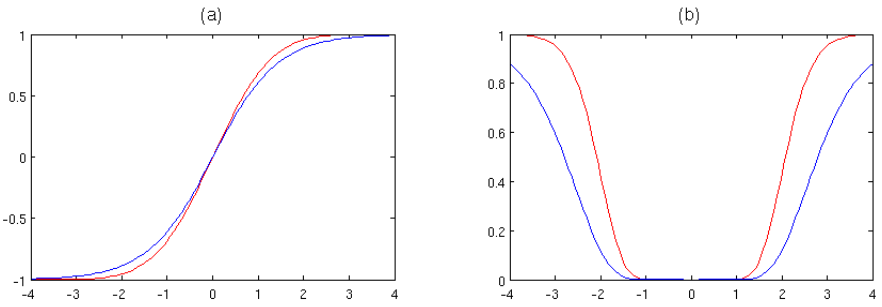

Figure 5.5: (a) Shapes of the Gaussian error function $\text{erf}(\frac{x}{\sqrt{2}})$ in red and the hyperbolic tangent $\tanh(\frac{x}{\sqrt{2}})$ in blue. (b) Shapes of functions $\text{erf}(\frac{x}{\sqrt{2}})^{18}$ in blue and $\tanh(\frac{x}{\sqrt{2}})^{18}$ in red. The hyperbolic tangent approximates the Gaussian error function.

$\frac{1}{3}$. We could also subtract the hash output by $\frac{1}{2}$, which matches with the $(u \cdot v)^2$ in expectation, although the randomness could introduce some negative values, for example, if the hash output lies in the range $[0, \frac{1}{2})$. As a follow-up work, it will be interesting to see how these negative edges impact the various cut and centrality measurements. As another future work, it will be interesting to see what the baseline shifts do to the various analytics computations. Centrality measures like eigenvector centrality emphasizes each edge more due to the baseline shift. But cut metrics like ratio and normalized cuts and modularity, the effect of baseline shift needs to be quantified since it could affect the optimal partition of the cuts.

3. $f(x) = \text{erf}(\frac{x}{\sqrt{2}})^p$ for p even. Recall that the Gaussian error function erf is defined as $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \text{dt}$, which is a sigmoid shaped function that generalizes the step function (see Figure 5.5(a)). When we raise this function to an even power, say $p = 18$, we get a function that is symmetric about $f(x) = 0$ (see Figure 5.5(b)). The expectation for higher moments of f can be computed using the formula $\text{E}[f(x)^p] = \int_{-\infty}^{\infty} x^p f(x)\ \text{dx}$. The cumulative distribution function of a Gaussian can be expressed as an error function, $\text{cdf}(x) = \frac{1}{2}\left(1 + \text{erf}(\frac{x}{\sqrt{2}})\right)$. Taking the derivative on both sides, we observe that $g(x) = \frac{1}{2}\text{erf}'(\frac{x}{\sqrt{2}})$, where g is a Gaussian. Using the identity that $\int g'(x)(g(x))^p\ \text{dx} = \frac{g(x)^{p+1}}{p+1}$, we can calculate the expectation of any even moment of erf,

$$\text{E}\left[\text{erf}\left(\frac{x}{\sqrt{2}}\right)^p\right] = \frac{1}{2}\int_{-\infty}^{\infty} \text{erf}\left(\frac{x}{\sqrt{2}}\right)^p g(x)\ \text{dx} = \frac{1}{2}\left[\frac{\text{erf}(\infty)^{p+1}}{p+1} - \frac{\text{erf}(-\infty)^{p+1}}{p+1}\right] = \frac{1}{p+1}.$$

Using (5.2) and (5.3), we get the baseline shift to be $\frac{2p+1}{(p+1)^2}$ and the standard deviation at the top of the curve to be $\frac{2p}{\sqrt{4p+1}}$. However, getting the entire analytic description for the LSH curve may be more involved. Even numerically, it may be computationally expensive to evaluate Gaussian integrals to compute this LSH function. We can therefore use hyperbolic tangents $\tanh(x) = \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}}$, which are sigmoid-shaped functions that are easier to compute that approximate $\mathrm{erf}(x)$ (see Figure 5.5). In Figure 5.4(c), we show an approximation of squared dot products (red) with both $\mathrm{erf}(\frac{x}{\sqrt{2}})^{18}$ (blue) and $\tanh(\frac{x}{\sqrt{2}})^{18}$ (green). Figure 5.4(f) shows the effect of using f and a vector $r = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$. We can see that vectors that have high projection along r are more pronounced, which more likely gives high dot products between two vectors that are highly correlated or anti-correlated. Using hyperbolic tangents for $p = 18$, we get a baseline shift of 0.086 and a standard deviation at the top of the curve to be 5.3. The number of hash functions we will need is in the early thousands, which is slightly more reasonable than using $f(x) = x^4$.

5.3 Estimation of Coherence

Coherence provides linear relationships between two random variables in the frequency domain, and can be interpreted as the amount of phase consistency between the corresponding time signals. A signal in time domain can be converted to frequency domain by taking its Fourier transform, which re-expresses the signal as a linear combination of cosine waves where each frequency is represented as a complex number that determines its amplitude and phase. For a specific frequency ω, the coherence between feature vectors b_i and b_j is defined as

$$\mathrm{coh}_{ij}(\omega) = \frac{[P_{ij}(\omega)]^2}{P_i(\omega)P_j(\omega)}$$

where the *cross-power spectrum* P_{ij} is the Fourier transform of the cross-correlation function R_{ij} and the *power spectrum* P_i is the Fourier transform of the auto-correlation function R_i. The *auto-correlation* function R_i at a specific time point is defined as $R_i(t) = \mathrm{E}[b_i[t] \cdot b_i]$ where $b_i[t]$ is the feature vector b_i shifted by t time points. If b_i and b_j are normalized to mean zero and unit variance, then $R_i(t) = \mathrm{corr}(b_i[t], b_i)$. The *cross-correlation* function can be similarly defined as $R_{ij}(t) = \mathrm{corr}(b_i[t], b_j)$. Note that since the power spectrum is based on the Fourier transform of the

cross-correlation function, which is equivalent to taking the convolution of the corresponding time signals, gives the correlations between the two signals over all possible time shifts.

Among different ways to compute magnitude squared coherence [27, 17, 94], Welch's method [27, 115] of averaging periodograms is very popular. Specifically, we take the Fourier transform of the original time-series signal modulated by a window function at d different equally-spaced window positions. Window functions, like the Hanning window function, are primarily used to make the frequency components in the middle of the window more pronounced and to reduce noise. For a survey of different window functions, see [53]. The frequency component (amplitude and phase) of a specific frequency ω is extracted out from each of the d windows and put into a length-d feature vector c_i. Sample coherence between nodes i and j can now be obtained by taking $\text{corr}(c_i, c_j)$. If the $n \times d$ complex matrix C contains these features along its rows normalized to mean zero and unit variance, then we get the coherence network $Y = CC^H$, where C^H is a matrix that contains the element-wise complex conjugates of C^T. The matrix Y is a Hermitian matrix where $y_{ij} = y_{ji}^*$, and can be expressed using its singular value decomposition

$$Y = \sum_{i}^{k} \lambda_i u_i u_i^{*T}$$

where λ_is are the singular values of Y and u_is are its singular vectors. Hermitian matrices are a generalization of real symmetric matrices, and their spectra can be computed in $O(nk^{\omega-1})$ time using the algorithm described in Section 2.3.

For a specific frequency ω, the magnitude squared coherence network Z between all pairs of nodes is defined as the element-wise squares of Y, that is, $z_{ij} = |y_{ij}|^2 = (c_i \cdot c_j^*)$. Using the exact mapping described in Section 5.1, we can express Z as a dot product network. Though we can use the power method to compute analytics implicitly in only $O(nk^{2(\omega-1)})$ time, we need $O(k^6)$ time if we need the entire spectrum, in which case we can use the above mapping only for small values of k. The next natural question is to apply the technique described in the previous section, and design functions f that operate on complex vectors to produce non-negative dot product networks with reduced dimensionality that approximately preserves the squares of the dot products in expectation. These functions could also generalize the non-negative mapping described for negative correlation networks. We have reasons to believe these approaches do not generalize well, but leave this problem open.

In many situations, it is desirable to analyze coherence network across a range of frequencies. A common approach is to average the coherence network across the desired frequencies. For instance, if Z_1 and Z_2 are magnitude squared coherence networks at two different frequencies, then we can form one coherence network that is representative of both frequencies by taking the average $\overline{Z} = \frac{1}{2}(Z_1 + Z_2)$. Since we can express both Z_1 and Z_2 implicitly, we can run the power method effectively. However, it is difficult to compute the entire spectrum of the resulting $\overline{Z}$.

We can also represent edge weights in partial coherence, which is an expression of the partial correlation between two time signals in the frequency domain. Using the definitions of partial correlation given in Section 2.5.2, we can obtain an expression which estimates the partial coherence matrix for a given frequency ω (see [78] for a discussion),

$$P = -DZ^{-1}D$$

where D is a diagonal matrix with $d_{ii} = \dfrac{1}{\sqrt{z_{ii}^{-1}}}$. As mentioned in Section 2.5.2, we can express P as an addition of a diagonal matrix and a matrix for which the spectrum is known. Thus, partial coherence can also be used as edge weights to compute analytics implicitly. However, if we need to obtain a partial coherence network that is the average of many frequencies, we can only use the power method to compute analytics that are dependent on the top few eigenvectors. Computing the entire spectrum in this case is difficult.

Chapter 6

Conclusions

In this dissertation, we have shown how to express a correlation network implicitly as a dot product network. One can use the power method to compute the top few eigenvalues and eigenvectors, and also compute the entire spectrum of a correlation network using its singular value decomposition. We showed how to compute a shrinkage parameter implicitly, and use this parameter to compute the spectrum of a matrix of biased correlation estimates, which can then be used to compute various analytics implicitly. We also showed how to express edge weights as partial correlations, coherence and partial coherence. Partial correlations and partial coherence can be expressed as a sum of a diagonal matrix and a matrix for which the spectrum can be computed. Correlation type networks are represented as $W = BB^T$, where data is represented in only one matrix B. It will be interesting to see other similarity functions that can be expressed as dot product of two different matrices $W = BC$ or dot products of more than two matrices $W = BCD \ldots$. A matrix of magnitude squared coherence can also be expressed as a dot product network using a mapping that preserves squares of correlations exactly. This however, requires $O(k^2)$ dimensions. An approximate mapping which requires relatively less dimensions can be obtained using locality sensitive hashing (LSH) that preserves the squares of correlations in expectation. We saw how different functions affect this approximate mapping. To obtain magnitude squared coherence, we however need a mapping that works on complex vectors. One future work of this dissertation is to design functions that can produce non-negative dot product networks that work on complex feature vectors. This can be seen as a generalization of the methods discussed in this thesis that eliminate negativity.

We showed how several analytics can be computed on dot product networks. Specifically,

node strength can be computed using matrix vector multiplication, and the eigenvector centrality and eigengap can be computed using the power method. The Katz, PageRank centralities and clustering coefficient can be computed implicitly after computing the spectrum of the dot product network. Other metrics that can be computed after computing the spectrum of the correlation network include ratio cuts, normalized cuts and modularity. One can also perform principal component analysis and correlation clustering implicitly. It will be interesting to see other metrics that can be computed in our dot product implicit model.

The approximate mapping discussed in this dissertation produces a non-negative dot-product network. As future work, it would be useful to know how this approximate mapping affect the various analytics mentioned. It is the hope that using a good choice of function f in our LSH functions would not affect the analytics too much. Another experimental analysis is to compare the efficiencies of our algorithms with current state-of-the-art explicit algorithms for computing various analytics. Our algorithms are also easily parallelize-able, and a parallel implementation and performance analysis can also be performed.

For big data that comes from biological applications, especially from fMRI, it will also be interesting to see how analytics effect with different resolutions. Such analysis can now be performed efficiently using our implicit algorithms.

Bibliography

[1] Nir Ailon and Bernard Chazelle. Approximate Nearest Neighbors and the Fast Johnson-Lindenstrauss Transform. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), Seattle, WA, USA, May 21-23, 2006*, pages 557–563. ACM, 2006.

[2] Nir Ailon and Edo Liberty. Fast Dimension Reduction Using Rademacher Series on Dual BCH Codes. *Discrete & Computational Geometry*, 42(4):615–630, 2009.

[3] Daniel Aloise, Gilles Caporossi, Pierre Hansen, Leo Liberti, Sylvain Perron, and Manuel Ruiz. Modularity Maximization in Networks by Variable Neighborhood Search. In David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors, *Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings*, volume 588 of *Contemporary Mathematics*, pages 113–128. American Mathematical Society, 2012.

[4] N. Alon and V. D. Milman. lambda$_1$, Isoperimetric Inequalities for Graphs, and Superconcentrators. *J. Comb. Theory, Ser. B*, 38(1):73–88, 1985.

[5] Noga Alon. Eigenvalues and Expanders. *Combinatorica*, 6(2):83–96, 1986.

[6] Alexandr Andoni and Piotr Indyk. Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*, 51(1):117–122, January 2008.

[7] Walter Edwin Arnoldi. The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem. *Quarterly of Applied Mathematics*, 9(17):17–29, 1951.

[8] Sanjeev Arora, Elad Hazan, and Satyen Kale. $O(\mathrm{sqrt}\ (\log n))$ Approximation to SPARSEST CUT in $\tilde{O}(n^2)$ Time. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 238–247. IEEE Computer Society, 2004.

[9] Sanjeev Arora and Satyen Kale. A Combinatorial, Primal-Dual Approach to Semidefinite Programs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 227–236. ACM, 2007.

[10] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander Flows, Geometric Embeddings and Graph Partitioning. *J. ACM*, 56(2), 2009.

[11] Sanjeev Arora, David Steurer, and Avi Wigderson. Towards a Study of Low-Complexity Graphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 2009.

[12] F. Gregory Ashby. *Statistical Analysis of fMRI Data*. MIT Press, 2011.

[13] Sean Bailey. Dot Product Graphs and their Applications to Ecology. *All Graduate Theses and Dissertations*, 2013.

[14] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation Clustering. *MACHLEARN: Machine Learning*, 56(1-3):89–113, 2004.

[15] Albert László Barabási. *Network science*. Cambridge University Press, August 2016.

[16] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The Architecture of Complex Weighted Networks. volume 101, pages 3747–3752, 2004.

[17] Jacob Benesty, Jingdong Chen, and Yiteng Huang. Estimation of the Coherence Function with the MVDR Approach. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP 2006, Toulouse, France, May 14-19, 2006*, pages 500–503. IEEE, 2006.

[18] Marian Boguna, Romualdo Pastor-Satorras, Albert Diaz-Guilera, and Alex Arenas. Emergence of Clustering, Correlations, and Communities in a Social Network Model. *Arxiv preprint cond-mat/0309263*, 2003.

[19] Phillip Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):1170–1182, March 1987.

[20] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On Modularity Clustering. *IEEE Trans. Knowl. Data Eng*, 20(2):172–188, 2008.

[21] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[22] Andrei Z. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–. IEEE Computer Society, 1997.

[23] Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[24] Jeremy Buhler and Martin Tompa. Finding Motifs Using Random Projections. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, RECOMB '01, pages 69–76. ACM, 2001.

[25] Wenchao Cai, Jue Wu, and Albert C. S. Chung. Shape-Based Image Segmentation Using Normalized Cuts. In *Proceedings of the International Conference on Image Processing, ICIP 2006, October 8-11, Atlanta, Georgia, USA*, pages 1101–1104. IEEE, 2006.

[26] Julio Carballido-Gamio, Serge J. Belongie, and Sharmila Majumdar. Normalized Cuts in 3-D for Spinal MRI Segmentation. *IEEE Trans. Medical Imaging*, 23(1):36–44, January 2004.

[27] Gary Carter, Charles Knapp, and Albert Nuttall. Estimation of the Magnitude-Squared Coherence Function via Overlapped Fast Fourier Transform Processing. *IEEE Transactions on Audio and Electroacoustics*, 21(4):337–344, 1973.

[28] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary Clustering. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 554–560. ACM, 2006.

[29] Selene E. Chew and Nathan D. Cahill. Semi-Supervised Normalized Cuts for Image Segmentation. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1716–1723. IEEE Computer Society, 2015.

[30] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs. *CoRR*, abs/1010.2921, 2010.

[31] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding Community Structure in Very Large Networks. *Phys. Rev. E*, 70:066111, December 2004.

[32] Giulio Costantini and Marco Perugini. Generalization of Clustering Coefficients to Signed Correlation Networks. *PLoS ONE*, 9(2):1–10, February 2014.

[33] Leon Danon, Jordi Duch, Albert Diaz-Guilera, and Alex Arenas. Comparing Community Structure Identification. *Journal of Statistical Mechanics: Theory and Experiment*, (09):P09008, 2005.

[34] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A Sparse Johnson–Lindenstrauss Transform. *CoRR*, abs/1004.4240, 2010.

[35] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *TCS: Theoretical Computer Science*, 361(2-3):172–187, 2006.

[36] Erik D. Demaine and Nicole Immorlica. Correlation Clustering with Partial Information. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003, Proceedings*, Lecture Notes in Computer Science, pages 1–13. Springer, 2003.

[37] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[38] Jordi Duch and Alex Arenas. Community Detection in Complex Networks using Extremal Optimization. *Phys. Rev. E*, 72:027104, August 2005.

[39] Dotan Emanuel and Amos Fiat. Correlation Clustering – Minimizing Disagreements on Arbitrary Weighted Graphs. In *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, Lecture Notes in Computer Science, pages 208–220. Springer, 2003.

[40] Andres P. Eriksson, Carl Olsson, and Fredrik Kahl. Normalized Cuts Revisited: A Reformulation for Segmentation with Linear Grouping Constraints. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8. IEEE Computer Society, 2007.

[41] Ernesto Estrada and Philip A. Knight. *A First Course in Network Theory*. Oxford University Press, May 2015.

[42] Charles M. Fiduccia, Edward R. Scheinerman, Ann N. Trenk, and Jennifer S. Zito. Dot Product Representations of Graphs. *Discrete Mathematics*, 181(1-3):113–138, 1998.

[43] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-Organization of the Web and Identification of Communities. *Computer*, 35(3):66–70, 2002.

[44] François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, Kobe, Japan, July 23-25, 2014*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.

[45] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[46] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of the 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, VLDB '99, pages 518–529. Morgan Kaufmann Publishers Inc., 1999.

[47] Michelle Girvan and M. E. J. Newman. Community Structure in Social and Biological Networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):8271–8276, 2002.

[48] Sally Goldman, Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with Qualitative Information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

[49] Sergio Gómez, Pablo Jensen, and Alex Arenas. Analysis of Community Structure in Networks of Correlated Data. *Phys. Rev. E*, 80:016114, July 2009.

[50] Roger Guimera and Luis A. Nunes Amaral. Functional Cartography of Complex Metabolic Networks. *Nature*, 433:895–900, February 2005.

[51] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. WTF: The Who to Follow Service at Twitter. In *Proceedings of the 22nd International Conference on World Wide Web, Rio de Janeiro, Brazil, May 13-17, 2013*, WWW '13, pages 505–514. International World Wide Web Conferences Steering Committee, 2013.

[52] Lars W. Hagen and Andrew B. Kahng. New Spectral Methods for Ratio Cut Partitioning and Clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[53] Frederic J. Harris. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. *IEEE Proceedings*, 66(1):51–83, January 1978.

[54] Nicholas J. A. Harvey. Algebraic Structures and Algorithms for Matching and Matroid Problems. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 531–542. IEEE Computer Society, October 2006.

[55] Taher Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable Techniques for Clustering the Web. In *Third International Workshop on the Web and Databases WebDB*, pages 129–134, 2000.

[56] Petter Holme, Mikael Huss, and Hawoong Jeong. Subnetwork Hierarchies of Biochemical Pathways. *Bioinformatics*, 19(4):532–538, 2003.

[57] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, STOC '98, pages 604–613. ACM, 1998.

[58] Mark Jerrum and Alistair Sinclair. Conductance And The Rapid Mixing Property For Markov Chains: The Approximation Of The Permanent Resolved. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, 20th Annual ACM STOC, pages 235–243. ACM, 1988.

[59] Matthew Johnson, Daniël Paulusma, and Erik Jan van Leeuwen. Algorithms to measure diversity and clustering in social networks through dot product graphs. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 2013.

[60] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[61] Gabriela Kalna and Desmond J. Higham. A Clustering Coefficient for Weighted Networks, with Application to Gene Expression Data. *AI Commun.*, 20(4):263–271, December 2007.

[62] Daniel M. Kane and Jelani Nelson. A Derandomized Sparse Johnson-Lindenstrauss Transform. *CoRR*, abs/1006.3585, 2010.

[63] Ross J. Kang, László Lovász, Tobias Müller, and Edward R. Scheinerman. Dot Product Representations of Planar Graphs. *Electr. J. Comb*, 18(1), 2011.

[64] Leo Katz. A New Status Index Derived from Sociometric Analysis. *Psychometrika*, 18(1):39–43, March 1953.

[65] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. *CoRR*, abs/1304.2338, 2013.

[66] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time. *CoRR*, bs/1301.6628:911–920, 2013.

[67] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4), 2009.

[68] Jamie King. Conductance and rapidly mixing markov chains. *Self Notes*, 2003.

[69] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. *CoRR*, abs/1003.2958:235–244, 2010.

[70] Ioannis Koutis, Gary L. Miller, and Richard Peng. Solving SDD linear systems in time $\tilde{O}(m \log n \log(1/\epsilon))$. *CoRR*, abs/1102.4842, 2011.

[71] Olivier Ledoit and Michael Wolf. Improved Estimation of the Covariance Matrix of Stock Returns with an Application to Portfolio Selection. *Journal of Empirical Finance*, 10(5):603–621, December 2003.

[72] Yongjae Lee and Woo Chang Kim. Concise formulas for the surface area of the intersection of two hemispherical caps. Technical report, KAIST Technical Report, 2014.

[73] Tom Leighton and Satish Rao. An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms. In *In Proceedings IEEE Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[74] Bo-Jr Li and Gerard Jennhwa Chang. Dot Product Dimensions of Graphs. *Discrete Applied Mathematics*, 166:159–163, 2014.

[75] Shengqiao Li. Concise Formulas for the Area and Volume of a Hyperspherical Cap. *Asian Journal of Mathematics and Statistics*, 4(1):66–70, 2011.

[76] Yonghui Li, Yong Liu, Jun Li, Wen Qin, Kuncheng Li, Chunshui Yu, and Tianzi Jiang. Brain Anatomical Network and Intelligence. *PLoS Computational Biology*, 5(5), 2009.

[77] Michael P. McAssey and Fetsje Bijma. A Clustering Coefficient for Complete Weighted Networks. *Network Science*, 3:183–195, June 2015.

[78] Tarek Medkour, Andrew T. Walden, and Adrian Burgess. Graphical Modelling for Brain Connectivity via Partial Coherence. *Journal of Neuroscience Methods*, 180(2):374–383, 2009.

[79] Milena Mihail. Conductance and Convergence of Markov Chains – A Combinatorial Treatment of Expanders. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.

[80] R. V. Mises and H. Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung . *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 9(2):152–164, 1929.

[81] Hiroshi Nagamochi and Toshihide Ibaraki. Computing Edge-Connectivity in Multigraphs and Capacitated Graphs. *SIJDM: SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[82] Mark E. J. Newman. Fast Algorithm for Detecting Community Structure in Networks. *Phys. Rev. E*, 69:066133, 2003.

[83] Mark E. J. Newman. Modularity and Community Structure in Networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[84] Mark E. J. Newman. *Networks: an Introduction*. Oxford University Press, Oxford; New York, 2010.

[85] Mark E. J. Newman and Michelle Girvan. Finding and Evaluating Community Structure in Networks. *Phys. Rev. E*, 69:026113, 2003.

[86] Christine Leigh Myers Nickel. Random dot product graphs: A model for social networks. *PhD dissertation, Johns Hopkins University*, February 2008.

[87] Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and Coherence of Motifs in Weighted Complex Networks. *Phys. Rev. E*, 71:065103, June 2005.

[88] Lorenzo Orecchia, Leonard J. Schulman, Umesh V. Vazirani, and Nisheeth K. Vishnoi. On partitioning graphs via single commodity flows. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 461–470. ACM, 2008.

[89] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[90] E. Pohlhausen. Berechnung der eigenschwingungen statisch-bestimmter fachwerke. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 1(1):28–42, 1921.

[91] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and Identifying Communities in Networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.

[92] Jan Reiterman, Vojtech Rödl, and Edita Sinajová. Geometrical Embeddings of Graphs. *DMATH: Discrete Mathematics*, 74(3):291–319, 1989.

[93] Jan Reiterman, Vojtech Rödl, and Edita Sinajová. On Embedding of Graphs into Euclidean Spaces of Small Dimension. *JCTB: Journal of Combinatorial Theory, Series B*, 56(1):1–8, 1992.

[94] Ignacio Santamaría and Javier Vía. Estimation of the Magnitude Squared Coherence Spectrum Based on Reduced-Rank Canonical Coordinates. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2007, Honolulu, Hawaii, USA, April 15-20, 2007*, pages 985–988. IEEE, 2007.

[95] Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertész. Generalizations of the Clustering Coefficient to Weighted Complex Networks. *Phys. Rev. E*, 75:027105, February 2007.

[96] Juliane Schäfer and Korbinian Strimmer. A Shrinkage Approach to Large-Scale Covariance Matrix Estimation and Implications for Functional Genomics. *Statistical Applications in Genetics and Molecular Biology, The Berkeley Electronic Press*, 4(1), 2005.

[97] Edward R. Scheinerman and Kimberly Tucker. Modeling Graphs Using Dot Product Representations. *Computational Statistics*, 25(1):1–16, 2010.

[98] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[99] Felix D. Schönbrodt and Marco Perugini. At What Sample Size do Correlations Stabilize? *Journal of Research in Personality*, 47(5):609–612, 2013.

[100] Jianbo Shi and Jitendra Malik. Normalized cuts and Image Segmentation. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), June 17-19, 1997, San Juan, Puerto Rico*, pages 731–737. IEEE Computer Society, 1997.

[101] Alistair Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Progress in theoretical computer science. Birkhäuser, 1993.

[102] Daniel A. Spielman and Shang-Hua Teng. Nearly-Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems. *CoRR*, abs/cs/0607105, 2006.

[103] Charles Stein. Inadmissibility of the Usual Estimator for the Mean of a Multivariate Normal Distribution. In Jerzy Neyman, editor, *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 197–206. University of California Press, 1956.

[104] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. In Jan van Leeuwen, editor, *Algorithms - ESA '94, Second Annual European Symposium, Utrecht, The Netherlands, September 26-28, 1994, Proceedings*, volume 855 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 1994.

[105] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, fifth edition, 2016.

[106] Chaitanya Swamy. Correlation Clustering: Maximizing Agreements via Semidefinite Programming. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 526–527. SIAM, 2004.

[107] Martijn van den Heuvel, Rene Mandl, and Hilleke Hulshoff Pol. Normalized cut group clustering of resting-state fmri data. *PLoS ONE*, 3(4):1–11, April 2008.

[108] Nisheeth K. Vishnoi. Lx = b Laplacian Solvers and their Algorithmic Applications. *Foundations and Trends® in Theoretical Computer Science*, 8(1-2):1–141, 2013.

[109] Dorothea Wagner and Frank Wagner. Between Min Cut and Graph Bisection. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings*, volume 711 of *Lecture Notes in Computer Science*, pages 744–750. Springer, 1993.

[110] Ludo Waltman and Nees Jan van Eck. A Smart Local Moving Algorithm for Large-Scale Modularity-Based Community Detection. *CoRR*, abs/1308.6604, 2013.

[111] Song Wang and Jeffrey Mark Siskind. Image Segmentation with Ratio Cut. *IEEE Trans. Pattern Anal. Mach. Intell*, 25(6):675–690, 2003.

[112] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[113] Yen-Chuen A. Wei and Chung-Kuan Cheng. Towards Eficient Hierarchical Designs by Ratio Cut Partitioning. In *1989 IEEE International Conference on Computer-Aided Design, ICCAD 1989, Santa Clara, CA, USA, November 5-9, 1989. Digest of Technical Papers*, pages 298–301. IEEE, 1989.

[114] Yen-Chuen A. Wei and Chung-Kuan Cheng. Ratio Cut Partitioning for Hierarchical Designs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 10(7):911–921, 1991.

[115] Peter D. Welch. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms. *IEEE Transactions on Audio and Electroacoustics.*, 15:70–73, 1967.

[116] Dennis Wilkinson and Bernardo A. Huberman. A Method for Finding Communities of Related Genes. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 101, page 5241–5248, April 2003.

[117] Stephen J. Young and Edward R. Scheinerman. Random Dot Product Graph Models for Social Networks. In Anthony Bonato and Fan R. K. Chung, editors, *Algorithms and Models for the Web-Graph, 5th International Workshop, WAW 2007, San Diego, CA, USA, December 11-12, 2007, Proceedings*, volume 4863 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2007.

[118] Stephen J. Young and Edward R. Scheinerman. Directed Random Dot Product Graphs. *Internet Mathematics*, 5(1):91–111, 2008.

[119] Bin Zhang and Steve Horvath. A General Framework for Weighted Gene Coexpression Network Analysis. In *Statistical Applications in Genetics and Molecular Biology*, volume 4, 2005.