Structure learning with large sparse undirected graphs and its applications

Fan Li

CMU-LTI-07-011

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

Yiming Yang, (Chair)
Jaime Carbonell
Roni Rosenfeld
Thomas Hofmann, Google Engineering Center

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

STRUCTURE LEARNING WITH LARGE SPARSE UNDIRECTED GRAPHS AND ITS APPLICATIONS

Fan Li
hustlf@cs.cmu.edu

Language Technologies Institute
School of Computer Science
Carnegie Mellon University

Abstract

Learning the structures of large undirected graphical models from data is an active research area and has many potential applications in various domains, including molecular biology, social science, marketing data analysis, among others. The estimated structures provide semantic clarity, the possibility of causal interpretation, and ease of integration with a variety of tools. For example, one very important direction in system biology is to discover gene regulatory networks from microarray data (together with other data sources) based on the observed mRNA levels of thousands of genes under various conditions. The basic assumption is that if two genes are co-regulated by the same proteins, then they tend to have similar patterns at the mRNA levels. Thus it is possible to learn a gene regulatory network from microarray data if we treat each gene as a node variable and each condition as a configuration instance.

Structure learning for undirected graphs is an open challenge in machine learning. Most probabilistic structure learning approaches enforce sparsity on the estimated structure by penalizing the number of edges in the graph, which leads to a non-convex optimization problem. Thus these approaches have to search for locally optimal solutions through the combinatorial space of structures, which makes them unscalable for large graphs. Furthermore, the local optimal solution they find could be far away from the global optimal solution, especially when the number of configuration instances is small compared with the number of nodes in the graph.

This thesis tries to address these issues by developing a novel structure learning approach that can learn large undirected graphs efficiently in a probabilistic framework. We use the Graphical Gaussian Model (GGM) as the underlying model and propose a novel ARD style Wishart prior for the precision matrix of the GGM, which encodes the graph structure we want to learn. With this prior, we can get the MAP estimation of the precision matrix by solving a modified version of Lasso regression and thus achieve a global optimal sparse solution. By proposing a generalized version of Lasso regression, which is called the Feature Vector Machine (FVM), our structure learning model is further extended so that it can capture non-linear dependencies between node variables. In particular, the optimization problem in our model remains convex even in non-linear cases, which makes our solution globally optimal. We have also developed a graph-based classification approach for predicting node labels given network structures, either observed or automatically induced. This approach is especially suitable when edges in the networks contain multiple input features.

The contributions of this thesis work can be seen from several aspects. First, it provides a probabilistic framework that allows us to learn global optimal undirected graph structures with a low polynomial (quadratic when the graph is sparse) computational cost . Second, the development of Feature Vector Machine theoretically enriches current approaches to feature selection and extends our structure learning model so that the non-linear dependencies among node variables can be captured. Third, a graph-based classification approach is developed for predicting node labels using the observed or learned network structures. Fourth, we provided empirical evidence for the proposed methods in gene regulatory network re-construction and gene function prediction, as well as multi-class text categorization tasks.

Acknowledgements

First, I would like to thank my advisor Yiming Yang, who brought me into the research field of information retrieval and machine learning. She not only gave me valuable advices in academics, but also helped my transition into a different culture. What I learned from her is far more than I expected and I am always impressed by her analytical skills and insightful ways of looking at problems.

I would also like to thank my other committee members, Jamie Carbonell, Roni Rosenfeld and Thomas Hofmann, for their invaluable assistance, feedback and patience at all stages of this thesis. Their criticisms, comments and advice were critical in making this thesis more accurate, more complete and clearer to read.

I have benefited a lot from the discussions with Eric Xing, who has provided me insightful suggestions and inspiration. Tong Zhang and Cliff Brunk also helped me a lot during my internship at Yahoo Applied Research and provided great ideas that are used in the thesis.

During my six years at Carnegie Mellon University, I received tremendous help from faculties, staffs, students and friends. They made my life enjoyable: Sachin Agarwal, Tom Ault, Jamie Callan, Yi Chang, Kevyn Collins-Thompson, Subramaniam Ganapathy, Benjamin Han, Abhay S. Harpale, Chun Jin, Rong Jin, Bryan Kisiel, Abhimanyu Lad, Ni Lao, Huaiwei Liao, Han Liu, Yan Liu, Jie Lu, Paul Oglivie, Jiazhi Ou, Yanjun Qi, Radhda Rao, Monica Rogati, Luo Si, Wen Wu, Rong Yan, Hui Yang, Jun Yang, Shinjae Yoo, Stacey Young, Jian Zhang, Yi Zhang, Ying Zhang, Jerry Zhu.

Finially, I would also like to thank my parents for their love and support, without which I would not have survived the Ph.D. process. Last but definitely not the least, I want to thank my wife, who brings to life so much love and happiness.

Contents

Chapter 1

Introduction

1.1 What is structure learning

A lot of real-world domains are richly structured, consisting of objects related to each other in complex ways. In many cases, the relationships among objects are unknown and can only be discovered from observed data. Structure learning algorithms can help us to determine these relationships and structures hidden in the data. The learned structures could be very useful because of their semantic clarity and understandability by humans, the possibility of causal interpretation, and ease of integration with a variety of decision support systems.

1.2 Where can structure learning be used

With the rapid emergence of high-throughput data, structure learning has been successfully applied to explore hidden patterns in many different fields. Some examples are given as following:

- **gene regulatory network analysis.** A *gene regulatory network* is a collection of DNA segments in a cell which interact with each other (indirectly through their RNA and protein expression products) and with other substances in the cell, thereby governing the rates at which genes in the network are transcribed into mRNA. Automated induction of large genetic regulatory networks has been an open challenge in computational biology. In recent years, advances in microarray technology have made possible the simultaneous monitoring of the mRNA levels of tens of thousands of genes from the entire genome under various conditions. If two genes are co-regulated by the same Transcriptional Factors (TFs), they will tend to have similar patterns of mRNA levels in different conditions. Thus it is possible to infer the structure of the

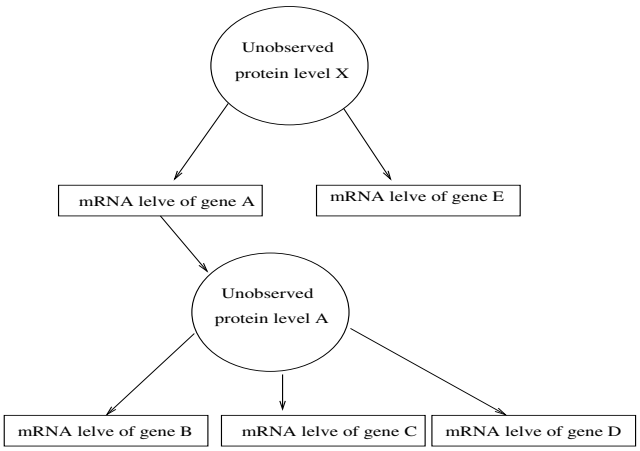

Figure 1.1: A typical sample of gene regulatory network

gene regulatory network from microarray data if we treat each gene as a node variable and treat the expression levels of all studied genes under a single condition as a (multivariate) sample. Figure 1.1 gives an examples of the gene regulated network.

- **Text mining and information retrieval.** In some textual corpus, documents have human-defined labels, which reflect the content of these articles. In other corpus, there are no human-defined labels at all. It is possible to learn category networks from observed data in both cases. When there are categorical labels defined by humans, we can treat each category as a node and learn a category network that reflects the relationship among these categories. When there are no human-defined labels, we can still assume that each document contains multiple hidden topics correlated to each other and induce a latent category network. The learned category networks would help us to predict document labels more accurately and guide users to search the whole collections in a much better way.

- **Social network analysis.** Social network analysis is the study of relationships among social entities; such as communications among members of a group, economic transactions between corporations, and treaties among nations. Structure learning algorithms have been widely used in this area to determin the relationship among users or groups from web-logs. Figure 1.2 gives an examples of the social networks among users.

Other examples of applications include citation analysis, entity extraction, causal analysis,etc.

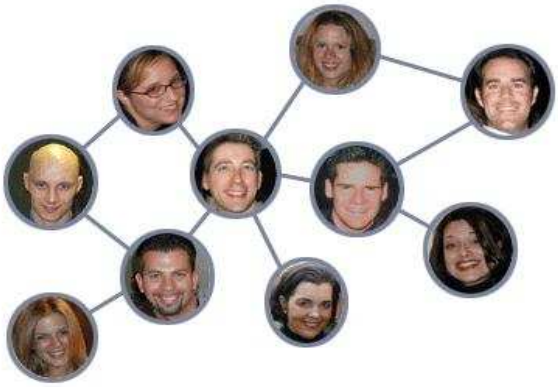

Figure 1.2: A typical sample of social network among users

1.3 Graphical models: a probabilistic framework for structure learning

Graphical models are a marriage between probability theory and graph theory. They provide a natural tool for us to model structure learning problem within a probabilistic framework. The random variables are represented as nodes in a graph and the conditional dependencies among random variables are represented as edges (either directed or not). In fact, graphical models not only give a natural representation of the relationships among multiple variables, but they also provide inference machinery for answering probabilistic distribution queries.

There are two kinds of graphical models in general: directed and undirected. The former restricts the structures to be directed acyclic graphs by definition while the latter can have arbitrary topologies.

- A directed graphical model is also called a Bayesian network. It is defined as a pair $< G, P >$: $G =< V, E >$ represents a directed acyclic graph (DAG), $V = \{V_i\}$ is a set of nodes, each of which corresponds to a random variable, $E = \{(V_i, V_j) : i \neq j\}$ is a set of edges with directions and P determines the conditional probabilistic distributions over the variables. If there is a directed edge in G from V_j to V_i, then V_j is called a parent of V_i and V_i is called a child of V_j. We use $parent(V_i)$ to represent the set of variables that are parents of V_i. The joint probability over all variables V can be calculated as the product of the conditional probability of each variable conditioned

on its parent, i.e.

$$P(V) = \prod_{V_i \in V} P(V_i | parent(V_i)) \tag{1.1}$$

- An undirected graphical model is also called a Markov Random Field. It is defined as a pair $< G, \Phi >$: $G =< V, E >$ represents an undirected graph, $V = \{V_i\}$ is a set of nodes, each of which corresponds to a random variable, $E = \{(V_i, V_j) : i \neq j\}$ represents a set of undirected edges and Φ is a set of potential functions defined over the maximal cliques * in graph G. We use ϕ_c to represent the potential function for maximal clique c. It can be any positive real-valued function. The joint probability of the variables V can be calculated as the normalized product of the potential functions over all the maximal cliques in G, i.e.

$$P(V) = \frac{\prod_{c \in C_G} \phi_c(V_c)}{\sum_{V'_c} \prod_{c \in C_G} \phi_c(V'_c)} \tag{1.2}$$

  where C_G represents the set of maximal cliques in graph G, V_c represents a specific configuration of variables in the maximal clique c and V'_c represents any possible configuration of variables in the maximal clique c.

Both bayesian networks and undirected graphical models are widely used in many real-world domains. Generally saying, if we do not care the direction information of the interactions among nodes in the network, then we would prefer to use undirected graphical models. If we want to model the directions of the interactions among nodes (for example,causal relationships), then bayesian networks are better choices.

1.4 Why Structure learning is difficult

Structure learning for undirected graphs is an open challenge in machine learning. The major difficulty of this problem comes from the fact that the number of possible structures is in exponential to the number of nodes in the graph, which makes it very hard to find the best structure in the searching process.

There are two major categories of existing structure learning approaches: constraint-based approaches and score-based approaches. Constraint-based approaches use conditional independence tests to tell the existence of edges between node variables. They lack an global objective function for structure finding and likelihood maximization. Score-based

*A maximal clique of a graph is a fully connected sub-graph that can not be further extended

approaches perform a search through the space of possible structures and attempt to identify the graph with the maximal likelihood. Most of them also enforce sparsity on the learned structure by penalizing the number of edges in the graph, which leads to a non-convex optimization problem. The local optimal solution they find could be far away from the global optimal solution, especially when the number of observed configuration instances is small compared with the number of nodes in the graph.

1.5 Our work and contributions

The contributions of this thesis work can be seen from several aspects.

- **Structure Learning.** The problem is to recover hidden structures from observed data. Existing methods are either unscalable for large networks or vulnerable to local optimal values due to their greedy search processes. In this thesis, a new probabilistic framework is developed for learning the structures of large undirected graphs. We use the Graphical Gaussian model (GGM) as the underlying model and propose a novel ARD style Wishart prior to enforce sparsity on the precision matrix of the GGM, which encodes the graph structure we want to learn. With this prior, we can get the MAP estimation of the precision matrix by solving a modified version of Lasso regression and thus achieve an optimal sparse solution with a low polynomial computational cost.

- **Feature Selection.** The problem is to discover a small subset of most predictive features in a high dimensional feature space, motivated for accurate modeling or human understandability. This task becomes more difficult when the dependency between response and predictor variables is non-linear and existing feature selection methods (like feature scaling kernel machine) often lead to highly non-convex optimizations, which are hard to solve. In this thesis, we proposed a generalized version of Lasso regression, called the Feature Vector Machine (FVM). It can capture non-linear dependencies between response and predictor variables and generates sparse solutions in the feature space, while still keeping the objective function convex and easy to solve. Integrated with FVM, our structure learning method can also be readily extended to capture non-linear dependencies between node variables in a graph.

- **Graph-based Classification.** The problem is to predict node labels given network structures, either observed or automatically induced. Graph-based semi-supervised learning approaches have been widely used for such tasks. However, one limitation of such approaches is that, the edge weights are treated as pre-determined constants, which can not be learned from the data. On the other hand, in many real world

domains, edge weights should be modeled as a function of multiple input variables. For example, in Internet, each edge represents a hyperlink and its weight is a function of the anchor text on that link. In gene regulatory networks, each edge represents a co-regulated gene pair and its weight is a function of the corresponding protein regulators. In citation networks, each edge represents an author pair and its weight is a function of the papers published by the two authors together.

In this thesis, we developed a graph-based classification approach, which can learn the edge weight functions automatically from the data and improve the finial classification performance.

1.6 Outline of the thesis

The rest of this thesis is organized as follows. Chapter 2 reviews the related work and also introduces some background knowledge. Chapter 3 presents our structure learning model for undirected graphs. Chapter 4 proposes a generalization of Lasso regression, named Feature vector machine, which enriches current approaches to feature selection and extends our structure learning model so that the non-linear dependencies among node variables can be captured. Chapter 5 presents our experimental results in the learning of gene regulatory networks. Chapter 6 gives the experimental results in text categorization using learned category networks. Chapter 7 proposed a graph-based classification approach that can learn edge weights automatically from the data, given network structures either learned or observed . Chapter 8 summarizes thesis claims.

Chapter 2

Literature Review and Background Knowledge

Structure learning is an active research area and has many potential applications in various domains, including causal discovery, molecular biology, information retrieval, and so on. [44], [33] and [22] developed several structure learning algorithms for Bayesian networks and used them to help causal discovery in decision support systems, covering a wide range of real life applications, such as medicine, agriculture, weather forecasting, financial modeling and animal breeding. [14], [18] and [4] used Bayesian networks to model the interactions between genes and tried to identify these interactions by structure learning algorithms. [50],[40],[17] and [9] did similar work but they used undirected graphical models instead of Bayesian networks as the underlying models. [14] tried to learn a network of words from a text corpus based on the co-occurrence of words in the same documents. [5] tried to extract topic networks from a text corpus and capture the correlations among topics using structure learning algorithms for undirected graphical models.

There are two major categories of structure learning approaches: score-based approaches and constraint-based approaches. Score-based approaches are mainly used to learn Bayesian networks (directed graphs), while constraint-based approaches have been used for both Bayesian networks and Markov Random Fields(undirected graphs). There are also some other approaches belonging to neither of these two categories. We will introduce them in detail for the three categories in the following sections.

2.1 Score-based approaches

Score based approaches ([20], [14],[18] and [4]) define a score that describes the fitness of

each possible structure to the observed data and then perform a search through the space of possible structures and attempt to identify the graph with the highest score. Obviously, the posterior probability of structure G given data D, represented as $P(G|D)$, can be used as the score. Thus, we have

$$\hat{G} = argmax_G logP(G|D) = argmax_G logP(D|G) + logP(G) \tag{2.1}$$

where $\hat{G}$ represents the estimated structure, and D represents the observed data. Here the prior $logP(G)$ can be thought as a regularizer that penalizes complex structures.

$$logP(D|G) = \int_\theta P(D|G,\theta)P(\theta|G)d\theta \tag{2.2}$$

is the marginal likelihood term where θ represents the parameters of the graphical models associated with G.

It is often intractable to calculate exact $logP(D|G)$ values due to the integration in formula 2.2. One popular solution is to use an asymptotic approximation called BIC (Bayesian Information Criterion) to replace this term as follows:

$$\log P(D|G) \approx \log P(D|G,\hat{\theta}_G) - \frac{\log N}{2}\#G \tag{2.3}$$

where $\hat{\theta}_G$ represents the maximum likelihood estimate of parameter θ, $\#G$ represents the number of parameters in graph G and N represents the number of samples in the observed data D. In this way, The integration 2.2 can be approximated by formula 2.3.

Once a score is defined, a searching strategy is needed to find the structure with the highest score. Generally, the number of undirected graph topologies with P node variables is $2^{\frac{P(P-1)}{2}}$, which is super-exponential in P. Thus when P is larger than 10, it is almost impossible to find the global optimal structure in such a huge searching space.

Hill-climbing search is a sub-optimal search method that is often used ([14]). It makes a series of edge changes (addition or deletion of one edge at a time). * For each edge change, we have a score for the graph before the change and a score for the graph after the change. Acceptance of the change depends on whether the latter score is higher than the previous one.

If a score for the whole graph can be decomposed as the sum of scores for individual nodes (and the associated edges) in the graph, then after each edge change, we only need to re-compute the scores for the nodes associated with the changed edge, but not the whole

*If we are learning a Bayesian network, we must guarantee that the resulting graph after each edge change is still a valid DAG.

graph. This can simplify the computation considerably.

The pseudocode of a hill-climbing search for Bayesian networks is listed as Algorithm 1.

Algorithm 1 Hill-climbing searching algorithm for structure learning

1. Set graph G' to be an initial graph structure, which could be random.
2. do
   (a) Set $G = G'$
   (b) Generate the acyclic graph set $Neighbor(G)$ by adding, removing or reversing an edge in graph G.
   (c) Set G' to be the graph in $Neighbor(G)$ with the highest score.
3. until the score of G' is not larger than the score of G
4. Return G

Recall that in a Bayesian network, the joint distribution of node variables $V = [V_1, V_2, ..., V_P]$ can be calculated as

$$logP(V) = \sum_{V_i \in V} logP(V_i|parent(V_i)) \tag{2.4}$$

This implies that the score of a structure in a Bayesian network can be decomposed as the sum of scores of nodes. When we add or delete an edge pointing to node V_j in a Bayesian network , we only need to re-compute the term $logP(V_j|parent(V_j))$ in the right-hand side of formula 2.4 and the scores associated with other nodes will not change. This property greatly reduces the computational cost of score-based structure learning approaches for Bayesian networks.

Note that the size of $Neighbor(G)$ is $O(P^2)$ where P is the total number of node variables. Thus step 2 needs to compute the scores of $O(P^2)$ structures, which makes the hill-climbing search unscalable when the graph is large (with several hundred nodes or larger). [14] proposed Sparse Candidate Hill Climbing (SCHC) to solve this problem. SCHC first estimates possible candidate parent set for each variable and then uses this set to constrain the hill-climbing search. The structure returned by the search can be in turn used to esti-

mate the possible candidate parent set for each variable in the next loop. In this way, the size of $Neighbor(G)$ is reduced to $O(P)$. In [14], SCHC has been used to learn Bayesian network structures on text data and genome microarray data with 500 to 800 variables. Besides Hill-climbing search, many other heuristic searching strategies have also been used to learn structures for Bayesian networks, including Best-first search([39]), genetic search ([25]), and greedy equivalence search([7]).

A limitation of score-based approaches is the local-optimal problem. The space of possible structures that needs to be explored will be extremely large. Thus the solution achieved by score-based approaches could be far away from the global optimal solution. This problem becomes more serious when the number of nodes P is large and the number of examples N is small.

While score-based approaches have been successfully used to learn structures for Bayesian networks, it is difficult to directly use them to learn the structures of Markov random fields. Undirected graphical models do not have the property shown in formula 2.4. Thus the score-based approaches for undirected graphical models are unscalable for large graphs. Instead, constraint-based approaches are common alternatives.

2.2 Constraint-based approaches

Constraint-based approaches ([44], [33] [22], [50] and [40]) use conditional independence tests to tell whether an edge between two node variables exists or not. The SGS algorithm (named after Spirtes, Glymour and Scheines in [44]) is the most straightforward constraint-based approach for Bayesian network structure learning. It determines the existence of an edge between every two node variables by conducting a number of independence tests between them conditioned on all the possible subsets of other node variables. The pseudocode of SGS is listed as Algorithm 2. †

The computational cost of SGS is exponential in the number of nodes in the graph because it has considered all the possible subsets of nodes that could be conditioned on. Thus the algorithm does not scale to large graphs.

Peter Spirtes and Clark Glymour (in [44]) developed a more efficient constraint-based algorithm, namely the PC algorithm. It conducts independence tests between all the variable pairs conditioned on the subsets of other node variables that are sorted by their sizes, from small to large. The subsets whose sizes are larger than a given threshold are not considered.

†Note that after slight modifications, SGS can be used to determine the existence of edges in Markov random fields. The details of step 3 are omitted here since this thesis only focuses on undirected graphs.

Algorithm 2 SGS for Bayesian network structure learning

1. Set graph G to be a completely connected graph. Set $\mathbf{U}$ to be the set of all node variables.
2. For each pair of node variables $\mathbf{x_1}$ and $\mathbf{x_2}$
   (a) For all possible subsets $\mathbf{S} \in \mathbf{U} - \mathbf{x_1} - \mathbf{x_2}$, conduct independence test between $\mathbf{x_1}$ and $\mathbf{x_2}$ conditioned on $\mathbf{S}$.
   (b) If for all possible subsets $\mathbf{S} \in \mathbf{U} - \mathbf{x_1} - \mathbf{x_2}$, the two variables are independent conditioned on $\mathbf{S}$, then delete the edge between $\mathbf{x_1}$ and $\mathbf{x_2}$ from graph G.
3. Determine the directions of the undirected edges found in the previous steps.

The PC algorithm can be applied on graphs with hundreds of node variables. However, when the number of nodes becomes even larger, the PC algorithm does not scale well either.

[34] has proposed the Grow-Shrinkage (GS) algorithm to learn Bayesian networks with thousands of node variables. In GS, two phases are used to identify the estimated Markov blanket $\hat{M}(V_j)$ for variable V_j. Initially each variable has an empty estimated Markov blanket set. In the "growing" phase, variables are added into $\hat{M}(V_j)$ sequentially using a forward feature selection procedure. The result is supposed to be a superset of the real Markov blanket. In the "shrinkage" phase, variables are deleted from $\hat{M}(V_j)$ if they are independent from the target variable conditioned on a subset of other variables in $\hat{M}(V_j)$. Then the algorithm tries to identify the parents and children for each variable from the estimated Markov blanket and tries to orient the edges. The time cost of GS is $O(P^2)$ where P represents the number of nodes. This algorithm can be used to learn structures of Markov random fields after some minor modifications.

When the sample size N is small compared to the number of nodes P (which is often the case in microarray data, for example), it is difficult to correctly test the partial correlation between two variables given all the other variables. Several possible solutions to circumvent this problem have been proposed. For example, [50] used Markov random fields to model gene networks and used order-limited significance tests to learn the graph structures (only first order partial correlations are considered). [40] did similar work but used multiple testing procedures to identify edges in Markov random fields.

A weakness of constraint-based approaches is the lack of an explicit global objective function directly engaged with likelihood maximization. Thus these approaches do not

support probabilistic optimization.

2.3 Other approaches

Other approaches for the learning of undirected graphical models include the follows, which are neither score-based approaches nor constraint-based approaches.

Learning undirected graphs via regression

The idea of regression-based approaches is to apply regression analysis to each variable as the response on the rest of the variables as the the input; the regression coefficients larger than a threshold are treated as links and those smaller than the threshold are ignored in the graph structure. Lasso regression ([47]) is often preferred in recent years because it tends to assign zero regression coefficients to most irrelevant or redundant variables (thus leading to a sparse graph structure). Furthermore, theoretical analysis in [36] and [35] shows that using L1 regularization, the sample complexity (i.e., the number of training examples required to learn "well") grows only logarithmically in the number of irrelevant features. This conclusion indicates that L1 regularized Lasso regression can be effective even if there are exponentially many irrelevant features as there are training examples (which is typical with microarray data). [35] used Lasso regression to estimate undirected graph structures and [17] used similar approaches to recover large scale transcriptional regulatory network from expression microarray data. However, the solutions of these approaches do not necessarily yield a maximum likelihood estimate of the undirected graphical model because the global objective functions in these models are not probabilistic.

Learning undirected graphs via corresponding directed acyclic graphs

[9] proposed an interesting approach that learns the structures of undirected graphical models by first learning their corresponding directed acyclic graphs (DAGs). In particular, they used the Graphical Gaussian model (GGM) [‡] as the underlying undirected graphical model. The main idea is that, for any undirected graph whose variables (nodes) follow a joint multivariate Gaussian distribution, there must be a DAG whose variables have exactly the same joint distribution (because a multivariate Gaussian distribution can be always decomposed into a set of one-dimensional Gaussian distributions using the chain rule). Thus, instead of learning the undirected graph directly, we can first learn a DAG with the largest posterior probability. Once this DAG is found, the joint multivariate Gaussian distribution of the variables in this DAG and its associated precision matrix (encoding an undirected graph

[‡]The details of GGM will be discussed in the next chapter.

structure) can be estimated. This precision matrix will be exactly the one in the GGM we are trying to learn. In this way, the problem of GGM structure learning has been transformed into the problem of DAG learning and any Bayesian network learning algorithms can be used to solve the problem.

One major problem of this approach is how to enforce sparsity on the estimated precision matrix, which encodes the connectivity in the undirected graph we are trying to learn. [9] achieved a sparse estimated precision matrix by penalizing the number of edges in the DAG that corresponds to the GGM. However, notice that there is not a simple mapping between the sparsity of an undirected graph and the sparsity of its corresponding DAGs. In general, an undirected graph can correspond to many DAGs with variables in different orders. The topologies of these DAGs could be very different although the variables in these DAGs share exactly the same joint multivariate Gaussian distribution. And the topology of the undirected graph could be very different from the topologies of all these DAGs. Thus, conceptually, it is not a good strategy to enforce sparsity on the corresponding DAGs in order to obtain the sparsity in the ultimate undirected graph. Also note that this approach needs to search DAGs with different variable orders since it penalizes the number of edges in a DAG, while the number of edges in a DAG depends on the variable order of this DAG for a given GGM. We will discuss this problem in detail in the next chapter.

Chapter 3

Learning Global Optimal Undirected Graphs with ARD Wishart Prior

In this chapter, we propose a new approach for learning the global optimal structure for undirected graphs. Our goal is to find an undirected graph structure that best explains the data, and is relatively sparse. We use Graphical Gaussian Models (GGM) ([26]) as the probabilistic framework, and we solve the problem by introducing a sparse prior for the precision matrix that encodes an undirected graph, and by estimating the Maximum a Posterior (MAP) for the precision matrix based on both the prior and the observed data. We also propose a modified version of Lasso regression that efficiently finds the DAG that corresponds to the most probable undirected graph, as an intermediate step toward the global optimal solution.

The rest of this chapter is organized as follows. Section 3.1 introduces Graphical Gaussian model (GGM) briefly. Section 3.2 introduces whishart prior for the precision matrix in GGM. Section 3.3 presents the relationship between the GGM and its corresponding DAGs and explains how to estimate the precision matrix in GGM by solving regressions. Section 3.4 presents approches to enforce sparsity on the estimated precision matrix. In this section, we proposes an ARD style Wishart prior for the GGM and shows its equivalence to the Laplace priors for the regression coefficients in the corresponding DAGs. Section 3.5 presents the parameter estimation procedure in our model. Section 3.6 gives a summarization. Section 3.7 gives of some proof that has been used in our derivations in this chapter.

3.1 Graphical Gaussian model

Graphical Gaussian model (GGM) ([26]) is one of the undirected graphical models that has been frequently used in recent years. In this chapter, we will use it as the underlying framework as our undirected graphic model. For simplicity, we assume all the variables have been preprocessed so that each of them follows a standard normal distribution. Let P be the number of nodes in the graph. We use $\mathbf{X} = [\mathbf{x_1}, ..., \mathbf{x_P}]'$to represent the $P \times 1$ column matrix. Then in GGM, the variables $\mathbf{X} = [\mathbf{x_1}, ..., \mathbf{x_P}]'$ is assumed to follow a multivariate normal distribution with covariance matrix Σ,i.e.

$$P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P}) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}\mathbf{X}'\Sigma^{-1}\mathbf{X}) \tag{3.1}$$

An edge between two nodes (variables) in the undirected graph encoded by the GGM implies that the two variables are NOT conditionally independent to each other given the rest variables (in other words, the conditional correlation between these two variables is not zero). Matrix $\Omega = \Sigma^{-1}$ is often called the precision matrix. The non-zero entries in matrix Ω are corresponding to the edges of the undirected graph. We attempt to learn the precision matrix of GGM with maximal posterior probability from the training data.

The most straight forward way of estimating Ω is to first calculate the estimated covariance matrix $\hat{\Sigma}$and then conduct the matrix inverse operation so that the estimated precision matrix is $\hat{\Omega} = \hat{\Sigma}^{-1}$. However, there are several problems if we estimate Ω in this way.

- First, when the number of observed instances is small compared to the number of node variables, the estimation may not be very reliable and we would want a prior on the precision matrix as a regularizer.
- Second, Ω is a $P \times P$ matrix and P may be a large number. Matrix inverse operation would take $O(P^3)$ computational cost, which could be too expensive.
- Third, we usually prefer to have a sparse undirected graph and $\hat{\Omega}$ estimated in this way is not sparse in general.

We will discuss how the previous related works address these concern, their limitations, and our strategy one by one in the following sections.

3.2 Wishart prior for the precision matrix in GGM

To address the first concern, one popular way is to introduce wishart prior as the regularizer when we estimate the precision matrix in GGM.

Wishart distribution, named in honor of John Wishart, is a family of probability distributions for nonnegative-definite matrices of random variables (”random matrices”). They are very important in multivariate statistics and they have been used as priors in the estimation of precision matrix in [37], [15], [9] and many other works. A detailed description of wishart distribution can be found in [37]. We only give a brief introduction in this section.

The Wishart distribution can be characterized by its probability density function, as follows. Let S be a $p \times p$ symmetric matrix of random variables that is positive definite and let V be a (fixed) $p \times p$ positive definite matrix. Then, S has a Wishart distribution $\mathbf{S} \sim W_p(n, V)$ if it has a probability density function $f_{\mathbf{S}}$ given by

$$f(\mathbf{S}) = \frac{|\mathbf{S}|^{(n-p-1)/2} \exp\left[-\text{trace}(V^{-1}\mathbf{S}/2)\right]}{2^{np/2}\,|V|^{n/2}\,\Gamma_p(n/2)}$$

where n is called degrees of freedom and V is called the scaling matrix. They are two major parameters that characterize the distribution. $\Gamma_p(\cdot)$ is the multivariate gamma function defined as

$$\Gamma_p(n/2) = \pi^{p(p-1)/4} \Pi_{j=1}^{p} \Gamma\left[(n+1-j)/2\right].$$

This distribution looks complex but is simply a multivariate generalization of the Gamma distribution. The following well-known property provides a intuitive way to understand it. Suppose $\mathbf{X}$ is a $P \times 1$ column-vector random variable that follows a multivariate gaussian distribution $\mathbf{X} \sim N_p(0, \Sigma)$. We randomly take n samples from $\mathbf{X}$ and get $\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_n}$. The the $P \times P$ matrix $\mathbf{S} = \sum_{k=1}^{n} \mathbf{X_i}\mathbf{X_i'}$ follows a wishart distribution $S \sim W_p(n, \Sigma)$ where parameter n is the degree of freedom and parameter Σ is the scaling matrix.

One important property of whishart distribution is that, it is the conjungate prior of the precision matrix in multivariate gaussian distribution. In other words, if we assume a wishart prior for the precision matrix and the observed data are generated from a multivariate gaussian distribution with zero means, then the posterior distribution of the precision matrix also follows a wishart distribution. This is the main reason that whishart prior has been used by many previous researchers.

Let's suppose for precision matrix Ω, we define a Wishart prior $\Omega \sim W_p(\delta, T)$ with δ degrees of freedom and diagonal scaling matrix T. If there is no specific prior knowledge for the precision matrix, the scaling matrix T in the wishart prior is often set to be a diagonal matrix (in which the off-diagonal elements are set to be zero and the diagonal elements are set to be a constant). That is, we let $T = diag(t, ..., t)$ where t is a constant. It is easy to see that such a prior would encourage the off-diagonal elements in the estimated precision matrix to shrink to zero.

However, the introduction of whishart prior can not solve our second and third con-

cerns in the previous section. It can not help us to avoid the matrix inverse operation and it does not lead to a sparse estimated precision matrix in general (just like in ridge regression, the norm-2 regularizer would not result in sparse regression coefficients, although the regression coefficients are encouraged to shrink to zero.

3.3 Estimating the precision matrix in GGM by solving regressions in a DAG

In this section, we will discuss how to address the second concern. That is, how to estimate the precision matrix without matrix inverse operation.

One possible strategy is to treat the multivariate gaussian distribution as the product of multiple conditioned univariate gaussian distribution, and then solve a set of regressions. This method has been used by [9] and the key idea is described as follows.

3.3.1 Mapping between the precision matrix in GGM and regression coefficients in a DAG

We can always represent the joint probability $P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P})$ as the product of multiple conditional probabilities. That is:

$$P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P}) = \prod_{i=1}^{P} P(\mathbf{x_i}|\mathbf{x_{i+1}}, ..., \mathbf{x_P}) \tag{3.2}$$

In GGM, $P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P})$ is assumed to follow a multivariate Gaussian distribution. From the properties of Gaussian distributions, we know that each conditional probability $P(\mathbf{x_i}|\mathbf{x_{i+1}}, ..., \mathbf{x_P})$ also follows a Gaussian distribution. This implies that, for any GGM, there must be at least one DAG with exactly the same joint distribution as the GGM [*]. Once we estimated all the parameters in the corresponding DAG, we can recover the precision matrix in the original GGM. Thus our task of structure learning in GGM is equivalent to learning the DAG with maximal posterior probability.

For any DAG, there is a specific ordering of variables, which we take here for discussion as simply $1, 2, ..., P$ without loss of generality. For any variable $\mathbf{x_i}$, only variables whose indexes are larger than i could be considered as $\mathbf{x_i}$'s parents. We use $\mathbf{x_{(i+1):p}}$ to represent these variables.

[*] In general, for any given GGM, there are more than one DAGs with the same joint probability distribution since the variables can be ordered in multiple ways. For example, $P(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}) = P(\mathbf{x_1}|\mathbf{x_2}, \mathbf{x_3})P(\mathbf{x_2}|\mathbf{x_3})P(\mathbf{x_3}) = P(\mathbf{x_1}|\mathbf{x_2}, \mathbf{x_3})P(\mathbf{x_3}|\mathbf{x_2})P(\mathbf{x_2})$.

Let's use β to represent the regression coefficients of the DAG. Notice once β is known, the DAG is determined.

For each $\mathbf{x_i}$, we can have

$$\mathbf{x_i} = \sum_{j=i+1}^{P} \beta_{ji}\mathbf{x_j} + \epsilon_i \tag{3.3}$$

where $\epsilon_i \sim N(0, \psi_i)$.

This can be further written in a matrix form

$$\mathbf{X} = \Gamma\mathbf{X} + \epsilon \quad and \quad \epsilon \sim N_P(0, \psi)$$

where $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P}]'$ represents the $P\times1$ column matrix, $\epsilon = (\epsilon_1, ..., \epsilon_P)'$, $\psi = diag(\psi_1, ..., \psi_P)$, and Γ is the $P \times P$ upper triangular matrix with zero diagonal elements and upper triangular, non-diagonal entries $\Gamma_{ij} = \beta_{ji}, (j > i)$.

Form the above formula, we can get

$$\mathbf{X} = (I - \Gamma)^{-1}\epsilon$$

Since ϵ follows a multivariate Gaussian distribution, from the properties of Gaussian distribution, we know $\mathbf{X}$ also follows a joint multivariate Gaussian distribution with covariance matrix $\Sigma = (I - \Gamma)^{-1}\psi((I - \Gamma)^{-1})'$. Thus the precision matrix of this distribution is

$$\Omega = \Sigma^{-1} = (I - \Gamma)'\psi^{-1}(I - \Gamma) \tag{3.4}$$

Notice this precision Ω is just the precision matrix of the undirected graph we are trying to learn and it encodes the structure of the undirected graph.

It's easy to see that the off-diagonal element Ω_{ji} (let's suppose $j > i$) has the value

$$\Omega_{ji} = \sum_{q<j,i} \beta_{iq}\beta_{jq}\psi_q^{-1} - \beta_{ji}\psi_i^{-1}$$

And the diagonal element Ω_{ii} has the value

$$\Omega_{ii} = \sum_{q<j,i} \beta_{iq}^2\psi_q^{-1} + \psi_i^{-1}$$

Thus, once we can solve the regressions in formula 3.3 and get estimations of β_{ji} and ψ_i for any index i, j, then we can get the estimation of precision Ω using formula 3.4.

The time complexity of standard linear regression is quadratic with the number of predictor variables. However, if we apply some sparse regression techniques and constrain the number of non-zero weighted predictor variables to be small, the computational cost of each regression could be reduced from quadratic to linear. Since we need to solve P regressions in the structure learning process, the total computational cost would be $O(P^2)$, smaller than the $O(P^3)$ cost for matrix inverse operations. We will have more discussion on this issue in the next section.

3.3.2 Translate the priors of the precision matrix Ω into the priors of regression coefficients β and noise variance ψ

As described in the previous subsection, the second issue can be addressed by avoiding the matrix inverse operation. And at the same time, we also want to address the first issue by defining a wishart prior for the precision matrix. This requires us to translate the wishart prior for the precision matrix into the prior for the regression coefficients β and noise variance ψ (in formula 3.3).

Let's suppose we have defined a wishart prior $\Omega \sim W_p(\delta, T)$ for the precision matrix where T is a diagonal matrix with fixed diagonal elements $T = diag(t, t, .., t)$. Let's use β_i to represent the $(p-i) \times 1$ matrix $\beta_i = (\beta_{(i+1)i}, ...\beta_{pi})'$. let's use T_i to represent the sub-matrix of T corresponding to variables $X_{(i+1):p}$.

From the derivations in [15], [†] we know the associated prior for ψ_i is [‡]

$$P(\psi_i^{-1}) = Gamma(\frac{\delta - i + 1}{2}, \frac{t^{-1}}{2}) \tag{3.5}$$

And the associated prior for β_i is $P(\beta_i|\psi_i, \theta_{(i+1):p}) = N_{p-i}(0, T_i\psi_i)$. Thus

$$P(\beta_{ji}|\psi_i) = N(0, t\psi_i) \tag{3.6}$$

With these priors of β and ψ, we can calculate the MAP estimation of β and ψ. and then recover Ω.

[†]The Gamma distribution here is defined as $Gamma(x; \alpha, \beta) = x^{\alpha-1}\frac{\beta^\alpha e^{-\beta x}}{\Gamma(\alpha)}$, which is slightly different from some other literatures. Also note the difference between the degree of freedom δ defined in this thesis and the degree of freedom δ' defined in [9]. In fact, they have a relationship $\delta = \delta' + P - 1$. These small differences in definition lead to slight difference between the formulas derived here and derived in [9].

[‡]To make the thesis self-contained, we also provided a brief proof in the third appendix of this chapter

3.3.3 A brief summary

As discussed in the previous sections, both the first and the second concern can be addressed by defining a wishart prior on the precision matrix and then transfer the problem of estimating precision matrix into the problem of solving a set of regressions. Also notice that, until now, the MAP estimation of the precision matrix will not be influenced by the variable order of the DAG in subsection 2.1. This is because the only prior defined is the wishart prior for the precision matrix and this prior does not depend on the variable order. The priors of β and ψ in formula 3.3 are derived from the wishart prior instead of being defined directly.

We will discuss how to enforce sparsity on the estimated precision matrix, which is the third concern, in the next section.

3.4 Enforcing sparsity on the estimated precision matrix

Based on our discussions in the previous two sections, both the first and the second concerns can be addressed in the GGM framework by introducing wishart priors and solving regressions in the DAG. The only left concern is how to make the estimation of the precision matrix Ω sparse.

One straight-forward approach is to define some additional penalizer on the regression coefficients β_{ji} in formula 3.3 when we solve regressions in the DAG. Note that β_{ji}s are the elements of matrix Γ. Thus if the estimated β_{ji}s are sparse, then the estimated $\Omega = (I - \Gamma)'\psi^{-1}(I-\Gamma)$ will be inheritably sparse. [9] has adopted this idea. In additional to defining a whishart prior of the precision matrix in the GGM, they introduced a zero-norm regularizer for β when solving regressions in the DAG. However, this strategy (named "strategy 1") could result in several important disadvantages, which will be discussed in detail in the following sub-section.

In this chapter, we proposed a new method to achieve sparseness in the estimated precision matrix, which we called "strategy 2". Our method differs fundamentally from [9] and has overcome many of its problems. It will be described in detail in later sub-sections.

3.4.1 Strategy 1: defining sparse priors on the regression coefficients

Dobra ([9]) introduced a wishart prior on the precision matrix, and then estimated the precision matrix by solving regressions in the DAG. They further defined additional zero-norm priors on the regression coefficients β in order to enforce sparsity. Such zero-norm priors on regression coefficients penalize the number of non-zero β_{ji} elements (which is also the number of edges in the DAG). At the same time, it also raises some new problems.

- First, the zero-norm penalizer on the regression coefficients β leads to a highly non-convex optimization problem, which is hard to solve. We can only get local optimal solutions.

- Second, a zero-norm penalizer on the regression coefficients penalizes the number of edges in the DAG. However, what we want to get is a sparse undirected graph. Note that there is not an explicit mapping between the number of edges in an undirected graph and the number of edges in its corresponding DAG, although the undirected graph and the DAG share the same joint multivariate distribution. Thus, conceptually saying, since our finial purpose is to learn a sparse undirected graph, we would prefer a sparse prior defined on the precision matrix, instead of a prior defined on the number of edges in the DAG.

  An intuitive example is given in figure 3.1 and 3.2. Figure 3.1 shows an undirected graph that we want to learn, with 8 node variables following a joint multivariate Gaussian distribution. If we represent the joint probability $P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_8})$ as the product of multiple conditional probabilities $P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_8}) = \prod_{i=1}^{8} P(\mathbf{x_i}|\mathbf{x_{i+1}}, ..., \mathbf{x_8})$ as formula 3.2, we can get the corresponding DAG shown in figure 3.2. In other words, in order to learn the undirected graph structure in figure 3.1 correctly, we need to learn the DAG structure in 3.2, which is not sparse. If we penalize the number of edges in this DAG and force it to be sparse, then we can not learn the undirected graph in figure 3.1 correctly.

- Third, a zero-norm prior directly defined on the regression coefficients depends on the variable order of the DAG in general [§] Thus if a zero-norm prior defined on the regression coefficients is introduced, the finial estimation of the precision matrix would depend on the variable order of the DAG. Consequently, a search over all the possible variable orders is needed, which is computational expensive.

A more natural and consistent approach (and our approach) is to directly define a sparse prior for the precision matrix that encodes the undirected graph, and then derive the DAG priors correspondingly, instead of starting for the DAG priors. This is the essential and unique part of our new approach.

3.4.2 Strategy 2: defining an ARD wishart prior for the precision matrix

In order to enforce sparsity on the estimated precision matrix, we do not use the standard wishart prior $\Omega \sim W_p(\delta, diag(t, ..., t))$. Instead, we use a modified version of wishart prior,

[§]Notice the fact that for a given undirected graph, its corresponding DAGs with different variable orders have different topologies.

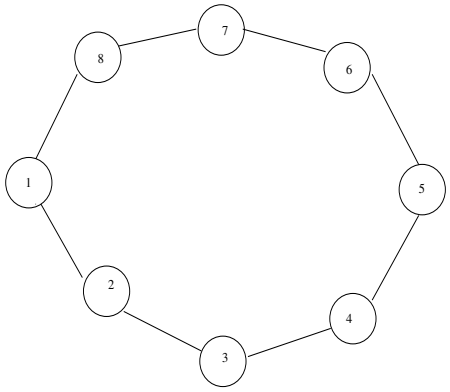

Figure 3.1: An undirected graph with 8 node variables, which are assumed to follow a joint multivariate Gaussian distribution.

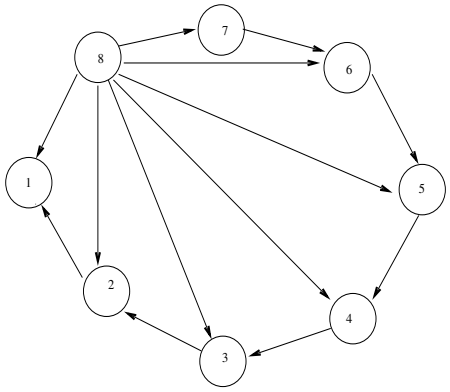

Figure 3.2: One of the corresponding DAGs of the undirected graph in figure 3.1.

which has the form $\Omega \sim W_p(\delta, T)$ and $T = diag(\theta_1, ..., \theta_P)$. Here θ_i are positive parameters and follow the hyper prior distribution:

$$P(\theta_i) = \frac{\gamma}{2} exp(\frac{-\gamma\theta_i}{2}) \tag{3.7}$$

This kind of hyper prior distributions have been suggested by [12] in a single regression model.

Recall the regression model defined on DAGs in formula 3.3,

$$\mathbf{x_i} = \sum_{j=i+1}^{P} \beta_{ji}\mathbf{x_j} + \epsilon_i$$

where β_{ji} represent the regression coefficients and $\epsilon_i \sim N(0, \psi_i)$ represents the Gaussian noise. Let's use β_i to represent the $(p-i)\times 1$ matrix $\beta_i = (\beta_{(i+1)i}, ...\beta_{pi})'$. let's use T_i to represent the sub-matrix of T corresponding to variables $X_{(i+1):p}$.

From the derivations in [9] and [15], we know that if the prior for precision matrix Ω is defined as $\Omega \sim W_p(\delta, T)$, then:

- The associated prior for ψ_i is

$$P(\psi_i^{-1}|\theta_i) = Gamma(\frac{\delta - i + 1}{2}, \frac{\theta_i^{-1}}{2}) \tag{3.8}$$

- The associated prior for β_i is $P(\beta_i|\psi_i, \theta_{(i+1):p}) = N_{p-i}(0, T_i\psi_i)$. Thus

$$P(\beta_{ji}|\psi_i, \theta_j) = N(0, \theta_j\psi_i) \tag{3.9}$$

Following the idea in [12], we can integrate out the hyper parameter θ from prior distribution of β_{ji} and get

$$\begin{aligned} P(\beta_{ji}|\psi_i) &= \int_0^\infty P(\beta_{ji}|\psi_i, \theta_j)P(\theta_j)d\theta_j \\ &= \frac{1}{2}\sqrt{\frac{\gamma}{\psi_i}}exp(-\sqrt{\frac{\gamma}{\psi_i}}|\beta_{ji}|) \end{aligned} \tag{3.10}$$

The detailed proof of this integration can be found in the appendix 3.7 at the end of this section.

Formula 3.8 and 3.10 give us the priors of β and ψ. Now it's clear that the prior of β is a kind of norm-1 prior and would lead to a sparse estimation of regression coefficients. We can calculate the MAP estimation β and ψ in the way the thesis proposal suggested. The finial estimated precision matrix can be then achieved using formula 3.4.

Let's suppose there are K samples and we use k to index them. x_{ki} represents the value of the ith variable in the kth sample and $x_{k(i+1):kp}$ represents the values of the $i+1$th, $i+2$th,..,pth

variables in the kth sample. Thus

$$\begin{aligned} P(\beta_i|\psi_i, Data) &\propto \prod_k P(x_{ki}|x_{k(i+1):kp}, \beta_i, \psi_i)P(\beta_i|\psi_i) \\ &\propto exp(\frac{\sum_k (x_{ki} - \sum_{j=i+1}^{p} \beta_{ji} x_{kj})^2 + \sqrt{\gamma\psi_i} \sum_{j=i+1}^{p} |\beta_{ji}|}{-\psi_i}) \end{aligned} \tag{3.11}$$

and

$$\begin{aligned} &P(\psi_i^{-1}|\theta_i, \beta_i, Data) \\ &\propto P(Data|\psi_i^{-1}, \beta_i, \theta_i)P(\beta_i|\psi_i^{-1}, \theta_i)P(\psi_i^{-1}|\theta_i) \\ &\sim Gamma(\frac{\delta + 1 + p - 2i + K}{2}, \frac{\sum_{j=i+1}^{p} \beta_{ji}^2 \theta_i^{-1} + \theta_i^{-1} + \sum_k (x_{ki} - \sum_{j=i+1}^{p} \beta_{ji} x_{kj})^2}{2}) \end{aligned} \tag{3.12}$$

Our ARD wishart prior is defined on the precision matrix Ω and does not depend on the variable order. Thus we do not need to search the order of variables in the DAG. In our model, the finial MAP estimation of the precision matrix will not be influenced by the variable order. Note that a multivariate Gaussian distribution can be re-written as the product of a set of one-dimensional Gaussian distributions using chain rule in an arbitrary order. This implies that, for DAGs with different variable orders, as far as the DAG parameters are all learned using MAP estimations and the priors of the DAG parameters are all derived from a unified global prior (not depending on the variable order) for the precision matrix, the finial recovered joint distributions of the variables in these DAGs should all be the same. In other words, these DAGs with different variable orders should all correspond to the same undirected graph, even if the structures of these DAGs are very different (e.g. some are sparse while some are dense). Thus, in principle, we can use an arbitrary variable order in our model.

3.5 Parameter estimation of the modified Lasso regression

From formula 3.11, we see the MAP estimation of β_i is

$$\hat{\beta}_i = argmin \sum_k (x_{ki} - \sum_{j=i+1}^{p} \beta_{ji} x_{kj})^2 + \sqrt{\gamma\psi_i} \sum_{j=i+1}^{p} |\beta_{ji}| \tag{3.13}$$

Thus $\hat{\beta}_i$ is the solution of a Lasso regression.

However, we cannot solve formula 3.13 as a standard Lasso regression. The reason is that ψ_i with different i values are generally not the same. Thus the regularization parameter $\sqrt{\gamma\psi_i}$ can not be treated as a single parameter. In order to estimate β_i using Lasso regression in formula 3.13, we need to first estimate the regularization parameter $\sqrt{\gamma\psi_i}$.

Formula 3.11 and 3.13 provide the posterior distribution of β_i conditioned on ψ_i while formula 3.12 provides us the posterior distribution of ψ_i conditioned on β_i (the effect of θ_i in formula 3.12 can be numerically integrated out by sampling method [¶]). Thus we can use an iterative procedure to estimate $\hat{\beta}_i$ and $\hat{\psi}_i$. We start from an initial guess $\hat{\psi}_i = 0$. Using this $\hat{\psi}_i$ value, we can estimate $\hat{\beta}_i$. Then using the estimated $\hat{\beta}_i$, we estimate $\hat{\psi}_i$ again. This procedure is iterated until the estimation $\hat{\psi}_i$ and $\hat{\beta}_i$ do not change.

It is obvious that this iterative process will converge because the joint probability increases in each iteration. Also note that the second order partial derivative of the log-likelihood for non-zero weighted β is $\frac{2\sum_k x_{kj}^2}{-\psi}$, which is always negative (according to formula 3.11). From the properties of convex function, we can also show that our solution is global optimal. A detailed proof is provided in the second appendix of this chapter.

Once $\hat{\psi}_i$ and $\hat{\beta}_i$ are known, we can estimate Ω using formula 3.4 easily. Thus the graph structure is learned. The overall procedure of of our structure learning approach is summarized as follwoing.

In the real implementation, we can incorporate the iterative procedure into the fast grafting algorithm that is used to solve Lasso regression [‖]. The fast grafting algorithm uses the sparse property of Lasso regression to accelerate the optimization. It scales linearly in the total number of features when Lasso regression has a sparse solution. Suppose we have P nodes in the graph. Note that we need to conduct P Lasso regressions. Thus the total computational cost of our structure learning algorithm is $O(P^2)$.

[¶]Ideally, we'd like to multiply formula 3.7 and formula 3.12 and analytically integrate out θ_i. However, it's unclear to us whether θ_i can be integrated out in a close form. Thus we just sample θ_i values under the distribution in formula 3.7 and use these values to simulate the integration operation. Since θ_i (with different i values) all follow the same distribution in formula 3.7, the sampled θ_i values can be used repeatedly with different i values.

[‖]Fast grafting is a kind of stage-wise gradient descent algorithm. It uses the sparseness property of Lasso regression to accelerate the optimization. The algorithm starts with two feature sets: feature set F and feature set Z. Then it gradually moves features from Z to F while training a predictor model only using features in F. The details of this algorithm are referred to [42]. In order to incorporate the iterative procedure adjusting the regularization parameter into the fast grafting algorithm, our implementation is a little different from [42]. We initialize ψ_i with a zero value. In the end of each iteration in the fast grafting algorithm, ψ_i and the regularization parameter λ are re-estimated based on the current regression coefficients β_i.

Algorithm 3 Pseudo code of our structure learning approach

1. Define an ARD-style wishart prior for the precision matrix in the GGM model.
2. Transform the parameters from the precision matrix to regression coefficients $\vec{\beta}$ and variance ψ, by decomposing the joint multivariate Gaussian distribution using chain rules.
3. Set initial values for the paramters ψ.
4. Loop until convergence:
   (a) Given ψ, estimate the regression coefficients $\vec{\beta}$ using formula 3.13.
   (b) Given $\vec{\beta}$, estimate the regression variance ψ using formula 3.12.
5. Recover the estimated precision matrix by the learned $\vec{\beta}$ and ψ using formula 3.4.

3.6 Summary

In this chapter, we develop a new approach for learning large undirected graphs for GGM in a probabilistic framework.

To be precise, we propose an Automatic Relevance Determination (ARD) style Wishart prior for the precision matrix [31]. The ARD prior is defined in a way that 1) the probability of each undirected graph can be estimated using a corresponding DAG; and 2) the MAP estimation of the precision matrix does not depend on the variable order in the DAG. Notice that the second property is only true for our ARD style Wishart prior defined for the precision matrix, and not true for the priors penalizing the number of edges in the DAG previously reported by [9] in their approach. As a result, the previous approach requires examining all possible orders of variables in order to find the best DAG, while our approach does not need to do so. This is a key property that enables us to solve the optimization problem far more efficiently, as described below.

We have shown that our ARD style Wishart prior for the precision matrix is equivalent to the Laplace priors for the regression coefficients in the corresponding DAGs. This means that we can find the MAP estimation of the precision matrix efficiently using a modified version of Lasso regression and a fast grafting algorithm (proposed in [42]). In this way, we have an efficient solution for finding the global optimal undirected graph structure in a probabilistic framework, instead of the exhaustive search of all DAGs which has an exponential complexity, or some greedy search which may be more scalable but cannot guarantee the global optimal solution. An important advantage of Lasso regression is its 1-norm regu-

larization, making the solution space convex and hence the efficiency in finding the global optimal point. Previous approaches ([9]) used a zero-norm regularizer in the search for the optimal DAG, where the solution space can be very bumpy.

Our structure learning algorithm scales quadratically in the total number of nodes and can be used to learn very large graphs. The L-1 norm regularizer and the convexity of the optimization problem in our model provide special advantages when the number of training examples is small compared to the number of nodes.

3.7 Appendix 1

We want to prove: Given $P(\theta_j) = \frac{\gamma}{2} exp(\frac{-\gamma\theta_j}{2})$ and $P(\beta_{ji}|\psi_i, \theta_j) = N(0, \theta_j\psi_i)$, then $P(\beta_{ji}|\psi_i) = \frac{1}{2}\sqrt{\frac{\gamma}{\psi_i}} exp(-\sqrt{\frac{\gamma}{\psi_i}}|\beta_{ji}|)$

Since

$$P(\beta_{ji}|\psi_i) = \int_0^\infty P(\beta_{ji}|\psi_i, \theta_j)P(\theta_j|\psi_i)d\theta_j = \int_0^\infty P(\beta_{ji}|\psi_i, \theta_j)P(\theta_j)d\theta_j$$

So, we need to prove:

$$\int_0^\infty \frac{\gamma}{2} exp(\frac{-\gamma\theta_j}{2}) \frac{1}{\sqrt{2\pi\theta_j\psi_i}} exp(\frac{-\beta_{ji}^2}{2\theta_j\psi_i}) d\theta_j = \frac{1}{2}\sqrt{\frac{\gamma}{\psi_i}} exp(-\sqrt{\frac{\gamma}{\psi_i}}|\beta_{ji}|) \tag{3.14}$$

We use LHS (left hand side) and RHS (right hand side) to represent two sides of the above formula.

The LHS can be re-written as

$$LHS = \int_0^\infty \frac{\gamma}{2} exp(\frac{-\gamma\theta_j}{2}) \frac{1}{\sqrt{2\pi\theta_j\psi_i}} exp(\frac{-\beta_{ji}^2}{2\theta_j\psi_i}) d\theta_j$$

$$= \frac{\gamma}{2\sqrt{2\pi}} \int_0^\infty \frac{1}{\sqrt{\theta_j\psi_i}} exp(\frac{\gamma\theta_j + \frac{\beta_{ji}^2}{\theta_j\psi_i}}{-2}) d\theta_j$$

Let $\theta = \theta_j\psi_i$. Then $\theta_j = \frac{\theta}{\psi_i}$.Use θ to replace θ_j in the above equation , then LHS can be

re-written as

$$LHS = \frac{\gamma}{2\psi_i\sqrt{2\pi}}\int_0^\infty \frac{1}{\sqrt{\theta}} exp(\frac{\gamma\frac{\theta}{\psi_i} + \frac{\beta_{ji}^2}{\theta}}{-2})d\theta$$

Let $\frac{\gamma}{\psi_i} = \gamma'$, then LHS can be re-written as

$$\begin{aligned}
LHS &= \frac{\gamma'}{2\sqrt{2\pi}}\int_0^\infty \frac{1}{\sqrt{\theta}} exp(\frac{\gamma'\theta + \frac{\beta_{ji}^2}{\theta}}{-2})d\theta \\
&= \frac{\gamma'}{2\sqrt{2\pi}}\int_0^\infty \frac{1}{\sqrt{\theta}} exp(\frac{(\sqrt{\gamma'\theta} - \frac{|\beta_{ji}|}{\sqrt{\theta}})^2 + 2\sqrt{\gamma'}|\beta_{ji}|}{-2})d\theta \\
&= \frac{\gamma'}{2\sqrt{2\pi}}\int_0^\infty \frac{1}{\sqrt{\theta}} exp(-\sqrt{\gamma'}|\beta_{ji}|) exp(\frac{(\sqrt{\gamma'\theta} - \frac{|\beta_{ji}|}{\sqrt{\theta}})^2}{-2})d\theta \\
&= \frac{\sqrt{\gamma'}}{2} exp(-\sqrt{\gamma'}|\beta_{ji}|)\frac{\sqrt{\gamma'}}{\sqrt{2\pi}}\int_0^\infty \frac{1}{\sqrt{\theta}} exp(\frac{(\sqrt{\gamma'\theta} - \frac{|\beta_{ji}|}{\sqrt{\theta}})^2}{-2})d\theta
\end{aligned} \tag{3.15}$$

Let

$$f(\beta_{ji}) = \frac{\sqrt{\gamma'}}{\sqrt{2\pi}}\int_0^\infty \frac{1}{\sqrt{\theta}} exp(\frac{(\sqrt{\gamma'\theta} - \frac{|\beta_{ji}|}{\sqrt{\theta}})^2}{-2})d\theta \tag{3.16}$$

Then, equation 3.15 becomes

$$LHS = \frac{\sqrt{\gamma'}}{2} exp(-\sqrt{\gamma'}|\beta_{ji}|) f(\beta_{ji}) \tag{3.17}$$

Comparing equation 3.14 and 3.17, we know now we only need to prove $f(\beta_{ji}) = 1$. Let $\sqrt{\theta} = t$, then $\theta = t^2$. Replace θ by t in equation 3.16, we have

$$\begin{aligned}
f(\beta_{ji}) &= \frac{\sqrt{\gamma'}}{\sqrt{2\pi}}\int_0^\infty \frac{1}{t} exp(\frac{(\sqrt{\gamma'}t - \frac{|\beta_{ji}|}{t})^2}{-2})dt^2 \\
&= \frac{\sqrt{2\gamma'}}{\sqrt{\pi}}\int_0^\infty exp(\frac{(\sqrt{\gamma'}t - \frac{|\beta_{ji}|}{t})^2}{-2})dt
\end{aligned} \tag{3.18}$$

Let

$$g(t) = exp(\frac{(\sqrt{\gamma'}t - \frac{|\beta_{ji}|}{t})^2}{-2}) \tag{3.19}$$

Then equation 3.18 can be written as

$$f(\beta_{ji}) = \frac{\sqrt{2\gamma'}}{\sqrt{\pi}} \int_0^\infty g(t)dt \tag{3.20}$$

Now we define the function $K(t)$ as:

$$K(t) = \frac{\sqrt{2\pi}}{4\sqrt{\gamma'}}[exp(2|\beta_{ji}|\sqrt{\gamma'})\text{erf}(\frac{t\sqrt{2\gamma'}}{2} + \frac{|\beta_{ji}|\sqrt{2}}{2t}) + \text{erf}(\frac{t\sqrt{2\gamma'}}{2} - \frac{|\beta_{ji}|\sqrt{2}}{2t})] \tag{3.21}$$

Here the erf() function is the Gauss error function defined as

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x exp(-s^2)ds$$

. It's obvious that $\text{erf}(\infty) = 1$ and $\text{erf}(-\infty) = -1$. Also notice the property that $\frac{d\ \text{erf}(x)}{dx} = \frac{2}{\sqrt{\pi}}exp(-x^2)$.

From equation 3.21, we know

$$\begin{aligned}
&\frac{d\,K(t)}{dt} = \frac{\sqrt{2\pi}}{4\sqrt{\gamma'}}[exp(2|\beta_{ji}|\sqrt{\gamma'})\frac{d\ \mathrm{erf}(\frac{t\sqrt{2\gamma'}}{2}+\frac{|\beta_{ji}|\sqrt{2}}{2t})}{dt} + \frac{d\ \mathrm{erf}(\frac{t\sqrt{2\gamma'}}{2}-\frac{|\beta_{ji}|\sqrt{2}}{2t})}{dt}]\\
&= \frac{\sqrt{2\pi}}{4\sqrt{\gamma'}}exp(2|\beta_{ji}|\sqrt{\gamma'})\frac{2}{\sqrt{\pi}}exp(-(\frac{t\sqrt{2\gamma'}}{2}+\frac{|\beta_{ji}|\sqrt{2}}{2t})^2)(\frac{\sqrt{2\gamma'}}{2}-\frac{\sqrt{2}|\beta_{ji}|}{2t^2})\\
&+\frac{\sqrt{2\pi}}{4\sqrt{\gamma'}}\frac{2}{\sqrt{\pi}}exp(-(\frac{t\sqrt{2\gamma'}}{2}-\frac{|\beta_{ji}|\sqrt{2}}{2t})^2)(\frac{\sqrt{2\gamma'}}{2}+\frac{\sqrt{2}|\beta_{ji}|}{2t^2})\\
&= \frac{1}{2\sqrt{\gamma'}}[exp(\frac{(t\sqrt{\gamma'}-\frac{|\beta_{ji}|}{t})^2}{-2})(\sqrt{\gamma'}-\frac{|\beta_{ji}|}{t^2}) + exp(\frac{(t\sqrt{\gamma'}-\frac{|\beta_{ji}|}{t})^2}{-2})(\sqrt{\gamma'}+\frac{|\beta_{ji}|}{t^2})]\\
&= \frac{1}{2\sqrt{\gamma'}}2\sqrt{\gamma'}exp(\frac{(t\sqrt{\gamma'}-\frac{|\beta_{ji}|}{t})^2}{-2})\\
&= exp(\frac{(t\sqrt{\gamma'}-\frac{|\beta_{ji}|}{t})^2}{-2})\\
&= g(t)
\end{aligned} \tag{3.22}$$

From equation 3.22, we know that equation 3.20 can be re-written as

$$f(\beta_{ji}) = \frac{\sqrt{2\gamma'}}{\sqrt{\pi}}\int_0^\infty dK(t) \tag{3.23}$$

Recall equation 3.21 and notice the fact $\mathrm{erf}(\infty) = 1$ and $\mathrm{erf}(-\infty) = -1$. Then we have

$$\begin{aligned}
&\int_0^\infty dK(t) = \frac{\sqrt{2\pi}}{4\sqrt{\gamma'}}[(exp(2|\beta_{ji}|\sqrt{\gamma'})(1)+1) - (exp(2|\beta_{ji}|\sqrt{\gamma'})(1)+(-1))]\\
&= \frac{\sqrt{2\pi}}{2\sqrt{\gamma'}}
\end{aligned}$$

Thus we know equation 3.23 can be re-written as

$$\begin{aligned}
f(\beta_{ji}) &= \frac{\sqrt{2\gamma'}}{\sqrt{\pi}}\frac{\sqrt{2\pi}}{2\sqrt{\gamma'}}\\
&= 1
\end{aligned} \tag{3.24}$$

Put equation 3.24 into equation 3.17, We have

$$\begin{aligned} LHS &= \frac{\sqrt{\gamma'}}{2} exp(-\sqrt{\gamma'}|\beta_{ji}|) \\ &= RHS \end{aligned} \tag{3.25}$$

Thus equation 3.14 is proved.

3.8 Appendix 2

We want to prove our solution achieved using algorithm 3 is global optimal .

In each iterationin in algorithm 3, we can get the MAP estimation of $\vec{\beta}$ given ψ using formula 3.13 and the MAP estimation of ψ given $\vec{\beta}$ using formula 3.12. So it is easy to see the joint probability of $\vec{\beta}$ and ψ given the observed data increases in each iteration. Thus the iteration will converge and we can get a local maximal point in the solution space using algorithm 3. Now we will show there is only one local maximal point in the solution space in our model (thus it is also global maximal).

Note that formula 3.13 defines a curve in the solution space. Assume there are mutiple local maximal points in the solution space in our model. It's easy to see all these local maximal points in the solution space should satisfy formula 3.13 so they are all on this curve. In other words, the curve defined by formula 3.13 has mutiple local maximal points. Thus it must have at least one local minimal point. Let's use $(\psi_{low}, \vec{\beta}_{low})$ to represent this local minimal point on the curve.

There are two possible directions if we move away from the point $(\psi_{low}, \vec{\beta}_{low})$ along the curve defined by formula 3.13 . One direction is increasing ψ and the other direction is decreasing ψ. Since $(\psi_{low}, \vec{\beta}_{low})$ is the local minimal point on the curve, moving in both directions along the curve will increase the joint probability.

However, note the following two facts.

- When the value of ψ is fixed, moving an infinitly small amount $d\vec{\beta}$ away from the point $(\psi_{low}, \vec{\beta}_{low})$ in any direction will always decrease the joint probability. The reason is that the log of formula 3.11 is convex and the partial derivative of $\vec{\beta}$ is zero at the point of $(\psi_{low}, \vec{\beta}_{low})$.
- When the value of β is fixed, moving an infinitly small amount $d\psi$ away from the point $(\psi_{low}, \vec{\beta}_{low})$ will at lease decrease the joint probability in one of the two directions. The reason is that when $\vec{\beta}$ is fixed, the joint probability becomes a gamma distribution, and gamma distribution does not have a local minimal point when ψ is positive.

From the above two facts, we can see that moving away from the point $(\psi_{low}, \vec{\beta}_{low})$ along the curve will decrease the joint probability in at lease one of the two directions. This contrdicts with the assumption that $(\psi_{low}, \vec{\beta}_{low})$ is a local minimal point on the curve.

Thus there can not be mutiple local maximal points in the solution space in our model.

3.9 Appendix 3

Assume $(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P})$ follows a P dimensional multivariate Gaussian distribution with zero means $(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P}) \sim N_P(\vec{0}, \Omega^{-1})$. Assume Ω follows a wishart distribution $\Omega \sim W_p(\delta, T)$ where T is a diagonal matrix $T = diag(t, ..., t)$. Let's consider the conditional distribution $P(\mathbf{x_i}|\mathbf{x_{i+1}}, ..., \mathbf{x_P})$. From the properties of Gaussian distribution, it's easy to know $P(\mathbf{x_i}|\mathbf{x_{i+1}}, ..., \mathbf{x_P})$ is also a Gaussian distribution and let's assume $P(\mathbf{x_i}|\mathbf{x_{i+1}}, ..., \mathbf{x_P}) \sim N(\mathbf{X_{(i+1):P}}\vec{\beta}_i, \psi_i)$. We want to prove $\psi_i^{-1} \sim Gamma(\frac{\delta-i+1}{2}, \frac{t^{-1}}{2})$.

Proof:

Let Σ be a $P \times P$ matrix that $\Sigma = \Omega^{-1}$, thus we have $(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_P}) \sim N_P(\vec{0}, \Sigma)$,

From the properties of Gaussian distribution, we know $(\mathbf{x_i}, ..., \mathbf{x_P})$ also follows a $P-i+1$ dimensional multivariate Gaussian distribution $(\mathbf{x_i}, ..., \mathbf{x_P}) \sim N_{P-i+1}(\vec{0}, \Sigma_{i:P})$ where $\Sigma_{i:P}$ is the $(P - i + 1) \times (P - i + 1)$ sub-matrix of Σ.

Note that the definitions of inverse whishart distribution are slightly different in different literatures. Since we will frequently cite [37] in this proof, we will use their version, which is slightly different from the inverse whishart distributions defined in [15] and [9].

From page 110 of [37], we know that since $\Sigma = \Omega^{-1}$ and $\Omega \sim W_p(\delta, T)$, then $\Sigma \sim W_P^{-1}(\delta + P + 1, T^{-1})$. And from page 111 of [37], we know that since $\Sigma_{i:P}$ is the sub-matrix of Σ, thus $\Sigma_{i:P} \sim W_{P-i+1}^{-1}(\delta + P + 1 - 2P + 2(P - i + 1), T_{i:P}^{-1})$, which is $\Sigma_{i:P} \sim W_{P-i+1}^{-1}(\delta + P - 2i + 3, T_{i:P}^{-1})$

Let's assume $\Omega'_{i:p} = \Sigma_{i:P}^{-1}$. From page 110 of [37], we know $\Omega'_{i:P} \sim W_{P-i+1}(\delta + P - 2i + 3 - (P - i + 1) - 1, T_{i:P})$, which is $\Omega'_{i:P} \sim W_{P-i+1}(\delta - i + 1, T_{i:P})$.

Suppose Ω'_{11} is the 1×1 upper left partition of $\Omega'_{i:P}$, and suppose T'_{11} is the 1×1 upper left partition of $T_{i:P}$ (thus T'_{11} in fact is just a real number t). From page 104 of [37], we know $\Omega'_{11} \sim W_1(\delta - i + 1, T'_{11})$.

On the other hand, suppose Σ'_{11} is the 1×1 upper left partition of $\Sigma_{i:P}$. Suppose Σ'_{12}, Σ'_{21} and Σ'_{22} are other partitions of $\Sigma_{i:P}$ defined in a similar way. From the properties of Gaussian distribution, we know $\psi_i = \Sigma'_{11} - \Sigma'_{12}{\Sigma'_{22}}^{-1}\Sigma'_{21}$. And from Matrix Inverse Lemma, we know ${\Omega'_{11}}^{-1} = \Sigma'_{11} - \Sigma'_{12}{\Sigma'_{22}}^{-1}\Sigma'_{21}$ Thus $\Omega'_{11} = \psi_i^{-1}$

We have already known that $\Omega'_{11} \sim W_1(\delta - i + 1, T'_{11})$ in our previous derivation. Thus we know $\psi_i^{-1} \sim W_1(\delta - i + 1, T'_{11})$. Recall T'_{11} in fact is just a real number t. Thus we have $\psi_i^{-1} \sim Gamma(\frac{\delta-i+1}{2}, \frac{t^{-1}}{2})$.

Chapter 4

Extend Our model to Non-linear Cases

As presented in chapter 2, our approach transformed the structure learning problem into the problem of solving a set of modified Lasso regressions, which are linear models. In this chapter, we extend our approach so that it can capture non-linear relationships between variables for structure learning. As the solution, we propose a generalization of Lasso regression, named "Feature vector machine" (FVM), by exploiting the duality between SVM regression and Lasso regression and use it to replace Lasso regression in our previous structure learning model ([32]). FVM can be also used as a non-linear feature selection approach with several advantages compared to other alternatives.

In this chapter, we first give an introduction of Lasso regression and SVM regression as well as a geometric comparison between their solution hyper-planes in section 3.1 and 3.2. Motivated by their geometric similarity, we then explore the mapping between these two algorithms. By re-formulating the standard Lasso regression into a form isomorphic to SVM, we prove that the solution of Lasso regression on a given dataset can be achieved by solving linear hard-margin SVM regression on the transposed dataset in section 3.3. Based on this re-formulation, we propose a generalization of Lasso regression, which we called "Feature vector machine" (FVM), by introducing kernels and slack variables in section 3.4. Preliminary experiments are conducted in section 3.5 to verify our derivations of FVM and show its effectiveness in non-linear feature selection tasks. Section 3.6 integrates FVM in our structure learning model and section 3.7 gives the summarization.

4.1 Introduction of Lasso regression and SVM regression

4.1.1 Lasso regression and its limitations

Lasso regression ([47]) tends to assign zero weights to most irrelevant or redundant features, and hence is a promising technique for identifying the relationship between the response variable and a small subset of predictive variables. It has been widely used in shrinkage and feature selection (and our other feature selection works can be found in [30] and [29]).

The loss function of Lasso regression is defined as:

$$L = \sum_i (y_i - \sum_p \beta_p x_{ip})^2 + \lambda \sum_p \|\beta_p\|_1 \tag{4.1}$$

where x_{ip} denotes the pth predictor (feature) in the ith datum, y_i denotes the value of the response in this datum, and β_p denotes the regression coefficient of the pth feature. The norm-1 regularizer $\sum_p \|\beta_p\|_1$ in Lasso regression typically leads to a sparse solution in the feature space, which means that the regression coefficients for most irrelevant or redundant features are shrunk to zero. Theoretical analysis in [36] indicates that Lasso regression is particularly effective when there are many irrelevant features and only a few training examples.

One of the limitations of standard Lasso regression is its assumption of linearity in the feature space. Hence it is inadequate to capture non-linear dependencies from features to responses (output variables). To address this limitation, [38] proposed "generalized Lasso regressions" (GLR) by introducing kernels. In GLR, the loss function is defined as

$$L = \sum_i (y_i - \sum_j \alpha_j k(x_i, x_j))^2 + \lambda \sum_i \|\alpha_i\|_1$$

where α_j can be regarded as the regression coefficient corresponding to the jth basis in an *instance space* (more precisely, a kernel space with its basis defined on all examples), and $k(x_i, x_j)$ represents some kernel function over the "argument" instance x_i and the "basis" instance x_j. The non-linearity can be captured by a non-linear kernel. This loss function typically yields a sparse solution in the instance space, but not in feature space where data was originally represented. Thus GLR does not lead to compression of data in the feature space.

Additive models ([45]) or generalized additive model ([19]) can also be used to capture the nonlinear dependencies between the response and predictor variables. However, their solutions are not sparse in the feature space either.

[49], [6] and [24] addressed the limitation from a different angle. They introduced

feature scaling kernels in the form of:

$$K_\theta(x_i, x_j) = \phi(x_i * \theta)\phi(x_j * \theta) = K(x_i * \theta, x_j * \theta)$$

where $x_i * \theta$ denotes the component-wise product between two vectors: $x_i * \theta = (x_{i1}\theta_1, ..., x_{ip}\theta_p)$. For example, [24] used a feature scaling polynomial kernel:

$$K_\gamma(x_i, x_j) = (1 + \sum_p \gamma_p x_{ip} x_{jp})^k,$$

where $\gamma_p = \theta_p^2$. With a norm-1 or norm-0 penalizer on γ in the loss function of a feature scaling kernel machine, a sparse solution is supposed to identify the most influential features. Notice that in this formalism the feature scaling vector θ is inside the kernel function, which means that the solution space of θ could be non-convex. Thus, estimating θ in feature scaling kernel machines is a much harder problem than the convex optimization problem in conventional SVM of which the weight parameters to be estimated are outside of the kernel functions.

What we are seeking for here is an alternative approach that guarantees a sparse solution in the feature space, that is sufficient for capturing both linear and non-linear relationships between features and the response variable, and that does not involve parameter optimization inside of kernel functions. The last property is particularly desirable in the sense that it will allow us to leverage many existing works in kernel machines which have been very successful in SVM-related research.

We notice that [21] has recently developed an interesting feature selection technique named "potential SVM", which has the same form as the basic version of FVM (with linear kernel and no slack variables). However, they did not explore the relationship between "potential SVM" and Lasso regression. Furthermore, their method does not work for feature selection tasks with non-linear models since they did not introduce the concepts of kernels defined on feature vectors.

4.1.2 SVM regression

Suppose we are given training data $\{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$. In the ϵ-SVM regression [48], the goal is to find a function $f(\vec{x})$ that has at most ϵ deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible. Consider a simple case as a linear function without the constant term, i.e., $f(x) = \vec{\beta}\vec{x}$ where $\vec{x}_i = (x_{i1}, ..., x_{iP})$ is the vector representation of an example, and $\vec{\beta} = (\beta_1, ..., \beta_P)$ are the model parameters (feature weights). The problem is defined as to learn the parameter vector using the training set with

the following objective:

$$\begin{cases} \underline{\min}_{\beta} & \frac{1}{2}\|\vec{\beta}\|^2 \\ \underline{\text{s.t.}} & \vec{\beta}\vec{x}_i - y_i \leq \epsilon \quad \forall i \\ & \vec{\beta}\vec{x}_i - y_i \geq -\epsilon \quad \forall i \end{cases}, \tag{4.2}$$

This formula is often referred as the primal objective function. By introducing a set of non-negative Lagrange multipliers α_i'' and α_i', this objective function can be re-formulated as the equivalent dual optimization problem [*] :

$$\begin{cases} \underline{\max}_{\alpha} & \frac{-1}{2}\sum_{i,j=1}^{n}(\alpha_i'' - \alpha_i')(\alpha_j'' - \alpha_j')\vec{x}_i\vec{x}_j \\ & -\epsilon\sum_{i=1}^{n}(\alpha_i'' + \alpha_i') + \sum_{i=1}^{n} y_i(\alpha_i'' - \alpha_i') \\ \underline{\text{s.t.}} & \alpha_i'' \geq 0 \quad \forall i \\ & \alpha_i' \geq 0 \quad \forall i \end{cases}, \tag{4.3}$$

Solving the problem for α_i'' and α_i' with $i = 1, ..., n$ typically yields a sparse solution in the sense that the weight $\alpha_i = \alpha_i'' - \alpha_i'$ can be exactly zero for many training examples $\vec{x}_i$.

4.2 Geometric similarities between the solution hyper-planes in SVM and Lasso regression

Suppose there are N examples and P features. Let's use X to denote the $N \times P$ data matrix, where each row corresponds to a training example and each column corresponds to a feature. In matrix X, the (i, j)th element x_{ij} represents the value of the jth feature in the ith example. Let's use $Y = (y_1, y_2, ..., y_N)^T$ to denote a $N \times 1$ matrix, where element y_i represents the value of the response variable in the ith example.

The shape of the solution hyper-planes of SVM regression is easy to get from formula 4.2. It is shown in figure 4.1. The space is spanned by features and each training example is represented as a data point in the space. The two hyper-planes in the graph correspond to the $epsilon$-tube, whose directions are decided by $\vec{\beta}$ in formula 4.2. We can see all the data points are either on a hyper-plane or between the two hyper-planes, due to the constraint in formula 4.2. The data points on the hyper-planes are called support vectors and have non-zero α_i weights. The data points between the hyper-planes have zero α_i weights and do not influence the shape of the solution hyper-plane.

In order to investigate the solution hyper-plane of Lasso regression, we first have a look

[*]Notice that we are assuming $f(x) = \vec{\beta}\vec{x}$, instead of $f(x) = \vec{\beta}\vec{x} + b$. Thus in formula 4.3, the constraint $\sum_i(\alpha_i'' - \alpha_i') = 0$ does not exist. If we assume $f(x) = \vec{\beta}\vec{x} + b$, then this constraint will appear.

at two propositions about Lasso regression.

Proposition 1: For a Lasso regression problem with the following objective function,

$$\begin{aligned} L &= \|X\beta - Y\|^2 + \lambda|\beta_p|_1 \\ &= \sum_i (\sum_p x_{ip}\beta_p - y_i)^2 + \lambda \sum_p |\beta_p| \end{aligned} \tag{4.4}$$

its solution $P \times 1$ column matrix β satisfies the following property:

$$\begin{cases} \text{if } \beta_q < 0 \text{ then } \sum_i (\sum_p \beta_p x_{ip} - y_i) x_{iq} = \frac{\lambda}{2} \\ \text{if } \beta_q = 0 \text{ then } |\sum_i (\sum_p \beta_p x_{ip} - y_i) x_{iq}| < \frac{\lambda}{2} \\ \text{if } \beta_q > 0 \text{ then } \sum_i (\sum_p \beta_p x_{ip} - y_i) x_{iq} = \frac{-\lambda}{2} \end{cases} \tag{4.5}$$

Proof: see [42]. [†] ■

Proposition 2 is the inverse proposition of proposition 1.

Proposition 2: For a Lasso regression problem in formula 4.4, if some β satisfies the properties in formula 4.5, then β is the solution of the Lasso regression.

Proof: see [42]. ■

Proposition 1 and Proposition 2 give us an intuitive hint how the solution hyper-plane of Lasso regression looks like. In order to make things clearer, we further re-write formula 4.5 as following

$$\begin{cases} \text{if } \beta_q < 0 \text{ then } f_q^T (X\beta - Y) = \frac{\lambda}{2} \\ \text{if } \beta_q = 0 \text{ then } \frac{-\lambda}{2} < f_q^T (X\beta - Y) < \frac{\lambda}{2} \\ \text{if } \beta_q > 0 \text{ then } f_q^T (X\beta - Y) = \frac{-\lambda}{2} \end{cases} \tag{4.6}$$

where the $N \times 1$ matrix $f_q = (x_{1q}, ..., x_{Nq})^T$ corresponds to the qth column of matrix X and represents the qth feature vector.

By comparing formula 4.6 and formula 4.2, we can see some similarities. In fact, let's consider a transposed space spanned by examples and each features is represented as a point in this space. Then we can clearly see the geometric meaning of the solution in Lasso regression. There would also be two parallel hyper-planes in this transposed space, whose direction is $X\beta - Y$, according to formula 4.6. All the non-zero weighted feature vectors are on two parallel hyper-planes. These feature vectors, together with the response variable,

[†] Although [42] did not give a formal mathematical proof, they have given detailed descriptions in words thus we can construct such a proof easily. The key step is to show the following fact: the argument that β satisfies formula 4.5 is equivalent to the argument that β is local optimal in terms of minimizing Lasso regression's objective function in formula 4.4. Also notice that Lasso regression is convex, which means if β is local optimal, then it is just the solution of Lasso regression.

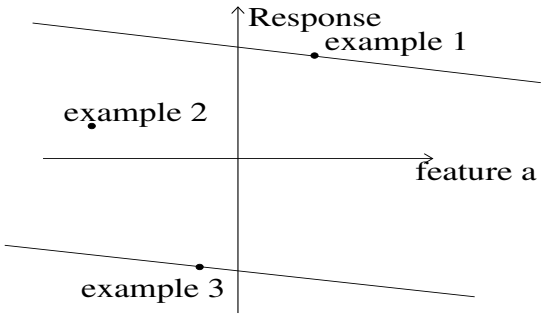

Figure 4.1: The solution of SVM in the space spanned by features. Each training example is represented as a data point in the space. Example 1 and example 3 are support vectors with non-zero weights. Example 2 has a zero weight.

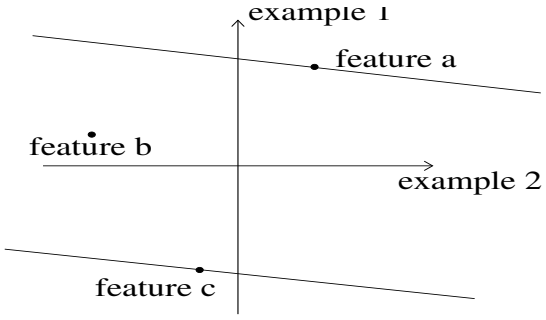

Figure 4.2: The solution of Lasso regression in a space spanned by training examples. Each feature is represented as a data point in the space. Feature a and feature c are "support features" with non-zero weights. Feature b has a zero weight.

determine the directions of these two hyper-planes. Thus these feature vectors correspond to the support vectors in SVM. Other feature vectors are between these two hyper-planes. They have zero weights and will not influence the solution hyper-plane. The figure 4.2 shows these two hyper-planes and the feature vector points.

As a summary, the solution hyper-planes in SVM and Lasso regression share many geometric similarities, although in different spaces. This motives us to further explore the internal relationship between these two methods.

4.3 The proof of the mapping between the solution of SVM and Lasso regression

The mapping between the solution of SVM and Lasso regression is described in theorem 1.

Theorem 1: Let's consider two problems. The first problem is applying Lasso regression on the $N \times P$ data matrix X and the $N \times 1$ response matrix Y. Let's use the $P \times 1$ column matrix β to represent the learned weights for features. The second problem is applying SVM regression on the $P \times N$ data matrix X^T and the $P \times 1$ response matrix $X^T Y$. Let's use the $P \times 1$ column matrix α to represent the learned weights for examples. Then $\beta = \alpha$.

proof: In order to be consistent, we first re-write the objective function of SVM in formula 4.2 into matrix form . In order to avoid confusion with the symbols X, Y and β already defined in theorem 1, we use X' to represent the input data matrix and use Y' to represent the response matrix for SVM. We use β' to represent the column weight matrix for features learned in SVM. Without loss of generality, we use $\frac{\lambda}{2}$ to replace ϵ. We get

$$\begin{cases} \underline{\min}_{\beta'} & \frac{1}{2}\|\beta'\|^2 \\ \underline{\text{s.t.}} & X'\beta' - Y' \leq \frac{\lambda}{2}e \\ & X'\beta' - Y' \geq \frac{-\lambda}{2}e \end{cases} , \tag{4.7}$$

where e is a one-vector whose dimensionality is equal to the number of examples in the SVM setting.

According to the conditions given in theorem 1, there are P examples and N features in the current SVM setting. Thus SVM's primary solution β' is a $N \times 1$ matrix and can be written as $\beta' = (\beta'_1, ..., \beta'_N)^T$. Since we know SVM's primary solution can be represented in the dual form, we can write

$$\beta'_i = \sum_p \alpha_p x'_{pi}$$

where α_p is the weight that SVM assigned to the pth example. Here x'_{pi} is the (p, i) element in matrix X' and represents the value of the ith feature in the pth example in input data matrix X'. Thus, using matrix form, we can write $\beta' = X'^T \alpha$.

According to the conditions given in theorem 1, we have $X' = X^T$ and $Y' = X^T Y$. Thus we have $\beta' = X'^T \alpha = X\alpha$. Put these stuffs into formula 4.7. We get

$$\begin{cases} \underline{\min}_{\alpha} & \frac{1}{2}\|X\alpha\|^2 \\ \underline{\text{s.t.}} & X^T X\alpha - X^T Y - \frac{\lambda}{2}e \leq 0 \\ & X^T X\alpha - X^T Y + \frac{\lambda}{2}e \geq 0 \end{cases} , \tag{4.8}$$

This is a standard optimization problem with inequalities. Following the standard con-

strained optimization procedure, we can derive the dual of this optimization problem. We can introduce Lagrange multipliers α_+ for the inequalities $X^TX\alpha - X^TY + \frac{\lambda}{2}e \geq 0$ and introduce Lagrange multipliers α_- for the inequalities $X^TX\alpha - X^TY - \frac{\lambda}{2}e \leq 0$. α_+ and α_- are two $P \times 1$ column matrixs with non-negative elements. Then we get the Lagrange L as:

$$\begin{aligned} L = \quad & \frac{1}{2}\alpha^T X^T X\alpha - \alpha_+^T(X^T(X\alpha - Y) + \frac{\lambda}{2}e) \\ & +\alpha_-^T(X^T(X\alpha - Y) - \frac{\lambda}{2}e) \end{aligned}$$

The optimizer satisfies:

$$\nabla_\alpha L = \quad X^TX\alpha - X^TX(\alpha_+ - \alpha_-) = 0$$

When X^TX is not inversible, there could be multiple solutions. Let's assume there is only one solution for SVM regression. Then it's easy to know $\alpha = \alpha_+ - \alpha_-$ is the solution of this loss function.

Notice that α, α_+, and α_- are all column matrices as $\alpha = (\alpha_1, ..., \alpha_P)^T$, $\alpha_+ = (\alpha_{+1}, ..., \alpha_{+P})^T$ and $\alpha_- = (\alpha_{-1}, ..., \alpha_{-P})^T$.

- For any element $\alpha_q > 0$, obviously α_{+q} should be larger than zero. From the KKT condition, we know
$$\sum_{i=1}^{N}(\sum_{p=1}^{P}\alpha_p x_{ip} - y_i)x_{iq} = -\frac{\lambda}{2}$$
holds at this time.

- For the same reason we know when $\alpha_q < 0$, α_{-q} should be larger than zero thus
$$\sum_{i}(\sum_{p}\alpha_p x_{ip} - y_i)x_{iq} = \frac{\lambda}{2}$$
holds.

- When $\alpha_q = 0$, α_{+q} and α_{-q} must both be zero (it's easy to see they can not be both non-zero from KKT condition), thus from KKT condition, both
$$\sum_{i}(\sum_{p}\alpha_p x_{ip} - y_i)x_{iq} > -\frac{\lambda}{2}$$

and

$$\sum_i (\sum_p \alpha_p x_{ip} - y_i) x_{iq} < \frac{\lambda}{2}$$

hold now, which means

$$|\sum_i (\sum_p \alpha_p x_{ip} - y_i) x_{iq}| < \frac{\lambda}{2}$$

at this time.

As a summary, α satisfy properties in formula 4.5. According to proposition 2, α is the solution of the Lasso regression in formula 4.4. Thus $\alpha = \beta$

Proof finished.

4.4 The Feature vector machine

4.4.1 FVM for non-linear feature selection

Theorem 1 allows us to get the solution of Lasso regression on a given dataset just by solving linear SVM regression on a transposed dataset. In other words. Lasso regression defined in formula 4.1 can be re-formulated as formula 4.8 and solved by standard linear SVM regression solvers. To clean up a little bit, we rewrite formula 4.8 in vector format:

$$\begin{cases} \underline{\min}_\beta & \frac{1}{2}\|[f_1, \ldots, f_K]\beta\|^2 \\ \underline{\text{s.t.}} & |f_q^T(y - [f_1, \ldots, f_K]\beta)| \leq \frac{\lambda}{2}, \quad \forall q. \end{cases} \tag{4.9}$$

In many cases, the dependencies between feature vectors are non-linear. Analogous to the SVM, we can introduce kernels that capture such non-linearity. Note that unlike SVM, our kernels are defined on feature vectors instead of the sampled vectors (i.e., the rows rather than the columns in the data matrix). Such kernels can also allow us to easily incorporate certain domain knowledge into the classifier.

Suppose that two feature vectors f_p and f_q have a non-linear dependency relationship. In the absence of linear interaction between f_p and f_q in the the original space, we assume that they can be mapped to some (higher dimensional, possibly infinite-dimensional) space via transformation $\phi(\cdot)$, so that $\phi(f_q)$ and $\phi(f_q)$ interact linearly, i.e., via a dot product $\phi(f_p)^T\phi(f_q)$. We introduce kernel $K(f_q, f_p) = \phi(f_p)^T\phi(f_q)$ to represent the outcome of this operation.

Replacing f with $\phi(f)$ in Problem (4.9), we have

$$\begin{cases} \underline{\min}_{\beta} & \frac{1}{2}\sum_{p,q}\beta_p\beta_q K(f_p, f_p) \\ \underline{\text{s.t.}} & \forall q, \quad |\sum_p \beta_p K(f_q, f_p) - K(f_q, y)| \leq \frac{\lambda}{2} \end{cases} \tag{4.10}$$

We refer to the optimization problem in formula 4.9, and its kernelized extensions in formula 4.10, as **Feature vector machine** (**FVM**).

Now, in Problem 4.10, we no longer have $\phi(\cdot)$, which means we do not have to work in the transformed feature space, which could be high or infinite dimensional, to capture non-linearity of features. The kernel $K(\cdot,\cdot)$ can be any symmetric semi-positive definite matrix. When domain knowledge from experts is available, it can be incorporated into the choice of kernel (e.g., based on the distribution of feature values). When domain knowledge is not available, we can use any kernels that can detect non-linear dependencies without any distribution assumptions. In the following we give one such example.

One possible kernel is the mutual information [8] between two feature vectors: $K(f_p, f_q) = MI(f_p, f_q)$. This kernel requires a pre-processing step to discritize the elements of features vectors because they are continuous in general. In this paper, we discritize the continuous variables according to their ranks in different examples. Suppose we have N examples in total. Then for each feature, we sort its values in these N examples. The first m values (the smallest m values) are assigned a scale 1. The $m+1$ to $2m$ values are assigned a scale 2. This process is iterated until all the values are assigned with corresponding scales. It's easy to see that in this way, we can guarantee that for any two features p and q, $K(f_p, f_p) = K(f_q, f_q)$, which means the feature vectors are normalized and have the same length in the ϕ space (residing on a unit sphere centered at the origin).

Mutual information kernels have several good properties. For example, it is symmetric (i.e., $K(f_p, f_q) = K(f_q, f_p)$, non-negative, and can be normalized. It also has intuitive interpretation related to the redundancy between features. Therefore, a non-linear feature selection using generalized Lasso regression with this kernel yields human interpretable results.

Note that we choose mutual information kernel mainly because it is simple and effective. Other choices are also possible and it could be interesting to explore them. we have not done thorough investigations in this line yet.

4.4.2 Further discussion about FVM

As we have shown, FVM is a straightforward feature selection algorithm for nonlinear features captured in a kernel; and the selection can be easily done by solving a standard SVM problem in the feature space, which yield an optimal vector β of which most elements are

zero. It turns out that the same procedure also seemlessly leads to a Lasso-style regularized nonlinear regression capable of predicting the response given data in the original space.

In the prediction phase, all we have to do is to keep the trained β fixed, and turn the optimization problem (4.10) into an analogous one that optimizes over the response y. Specifically, given a new sample x_t of unknown response, our sample matrix grows by one column $\rightarrow [, x_t]$, which means all our feature vectors gets one more dimension. We denote the newly elongated features by $F' = \{f'_q\}_{q\in A}$ (note that A is the pruned index set corresponding to features whose weight β_q is non-zero). Let y' denote the elongated response vector due to the newly given sample: $y' = (y_1, ..., y_N, y_t)^T$, it can be shown that the optimum response y_t can be obtained by solving the following optimization problem [‡]:

$$\min_{y_t} K(y', y') - 2\sum_{p\in A} \beta_p K(y', f'_p) \tag{4.11}$$

When we replace the kernel function K with a linear dot product, FVM reduces to Lasso regression. Indeed, in this special case, it is easy to see from Eq. (4.11) that $y_t = \sum_{p\in A} \beta_p x_{tp}$, which is exactly how Lasso regression would predict the response. In this case one predicts y_t according to β and x_t without using the training data . However, when a more complex kernel is used, solving Eq. (4.11) is not always trivial. In general, to predict y_t, we need not only x_t and β, but also the non-zero weight features extracted from the training data.

As in SVM, we can introduce slack variables into FVM to define a "soft" feature surface. But due to space limitation, we omit details here. Essentially, most of the methodologies developed for SVM can be easily adapted to FVM for nonlinear feature selection.

One thing to be noticed is that in FVM regression, the computational cost is quadratic to the number of features and linear to the number of samples. Thus when the number of features is huge and the number of samples is small, the computational cost of FVM regression is larger than that of SVM.

4.5 Experiments

4.5.1 Experiments to verify our derivations and proofs

We conducted experiments to verify our derivations and proofs for theorem 1. A 100×30 data matrix X, with 100 examples and 30 features, was randomly generated. The values of

[‡]For simplicity we omit details here, but as a rough sketch, note that Eq. (4.10) can be reformed as

$$min_\beta \|\phi(y') - \sum_p \beta_p \phi(f'_p)\|^2 + \sum_p \|\beta_p\|_1.$$

Replacing the opt. argument β with y and dropping terms irrelevant to y_t, we will arrive at Eq. (4.11).

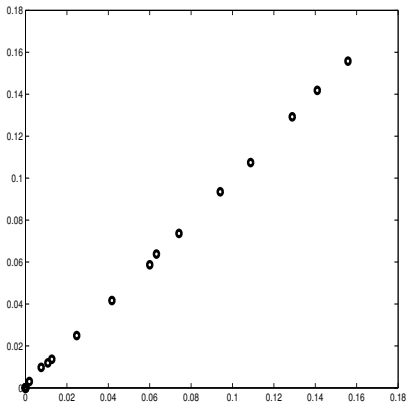

Figure 4.3: comparison between the weights estimated by SVM (in the X axis) and the weights estimated by Lasso codes in [16] (in the Y axis). We can see that the two estimations are very close.

the elements in X are sampled from the uniform distribution between 0 and 1. A 100×1 response matrix Y is also randomly generated in the same way. Our task is to learn the 30×1 weight matrix β by applying Lasso regression in formula 4.4 on X and Y.

According to theorem 1, we can get β by feeding the data matrix X^T and the response matrix $X^T Y$ into SVM regression solver. We chooses the SVM-light package [23] as our toolkit, due to its large-scale processing ability and the widespread popularity. [§] SVM-light returned us an estimated 30×1 weight matrix $\hat{\beta}$.

We also used the matlab codes of Lasso regression implemented according to [16] to get another estimated $\hat{\beta}'$. We use it as the baseline to verify the correctness of the $\hat{\beta}$ returned by SVM.

Figure 4.3 shows the comparison between the two weight vectors, learned by SVM-light and by Lasso matlab toolkit respectively. Each circle in the figure represents a feature. The

[§]It's important to choose the correct parameter options when we use SVM-light to calculate solutions for Lasso regression. Let's suppose we want to set the regularization parameter λ in Lasso regression to be 0.7. Then the command line we use for SVM-light is as following:

$$svm\_learn \ -a \ fa \ -z \ r \ -b \ 0 \ -c \ 9999 \ -w \ 0.7 \ fd \ fm$$

Here "-b 0" option is necessary to guarantee that the unbiased hyperplane $y = \vec{w}\vec{x}$ instead of the biased hyperplane $y = \vec{w}\vec{x} + b$ is used in the model. Another thing is that the line number in file "fa" will be $2 \times P$ if there are P lines in the file "fd". From the first line to the Pth line in the file "fa", the value in the ith line is the $alpha_+$ value of the ith example. However, from the $P + 1$th line to the $2 \times P$ line in the file "fa", the value in the $P + i$th line is corresponding to the $-\alpha_-$ value for the $P + 1 - i$th example. This rather unnatural format in svm-light need be noticed otherwise the results will be wrong.

value on the X axis of a feature represents the weight of this feature learned by SVM-light according to theorem 1. The value on the Y axis of a feature represents the weight of this feature learned by Lasso matlab toolkit. Notice that many features have been zero-weighted by both methods thus overlap together. We can see that the points in the figure almost form a straight line, which means that the two estimations are very close.

4.5.2 Experiments of non-linear feature selection by FVM

We test FVM on a simulated dataset with 100 features and 500 examples. The response variable y in the simulated data is generated by a highly nonlinear rule:

$$y = sin(10 * f_1 - 5) + 4 * \sqrt{1 - f_2^2} - 3 * f_3 + \xi.$$

Here feature f_1 and f_3 are random variables following a uniform distribution in $[0, 1]$; feature f_2 is a random variable uniformly distributed in $[-1, 1]$; and ξ represents Gaussian noise. The other 97 features $f_4, f_5, ..., f_{100}$ are conditionally independent of y given the three features f_1, f_2 and f_3. In particular, $f_4, ..., f_{33}$ are all generated by the rule $f_j = 3 * f_1 + \xi$; $f_{34}, ..., f_{72}$ are all generated by the rule $f_j = sin(10 * f_2) + \xi$; and the remaining features $(f_{73}, ..., f_{100})$ simply follow a uniform distribution in $[0, 1]$. Fig. 4.4 shows our data projected in a space spanned by f_1 and f_2 and y.

We use a mutual information kernel for our FVM. For each feature, we sort its values in different examples and use the rank to discritize these values into 10 scales (thus each scale corresponds to 50 data points). We apply both standard Lasso regression and FVM with mutual information kernel on this dataset. The value of the regularization parameter λ can be tuned to control the number of non-zero weighted features. In our experiment, we tried two choices of the λ, for both FVM and the standard Lasso regression. In one case, we set λ so that only 3 non-zero weighted features are selected; in another case, we relaxed a bit and allowed 10 features.

The results are very encouraging. As shown in Fig. (4.5), under stringent λ, FVM successfully identified the three correct features, f_1, f_2 and f_3, whereas Lasso regression has missed f_1 and f_2, which have non-linear dependencies with y. Even when λ was relaxed, Lasso regression still missed the right features, whereas FVM was very robust.

4.6 Integrate FVM in our structure learning model

In our previous structure learning model, the P node variables $\mathbf{x_1}, ..., \mathbf{x_P}$ are assumed to follow a multivariate normal distribution. However, such a model can not capture the possible non-linear dependencies among these variables.

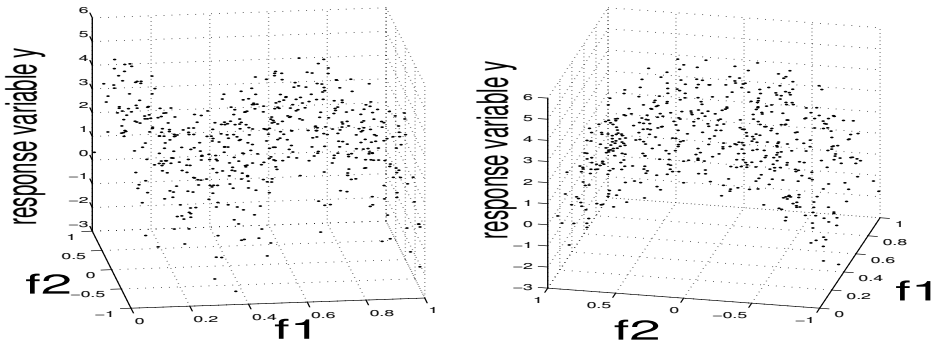

Figure 4.4: The responses y and the two features f_1 and f_2 in our simulated data. Two graphs from different angles are plotted to show the distribution more clearly in 3D space.

Following the idea of FVM, suppose that two variables $\mathbf{x_p}$ and $\mathbf{x_q}$ have a non-linear dependency relationship. In the absence of linear interaction between $\mathbf{x_p}$ and $\mathbf{x_q}$ in the the original space, we assume that they can be mapped to some (higher dimensional, possibly infinite-dimensional) space via transformation $\phi(\cdot)$, so that $\phi(\mathbf{x_p})$ and $\phi(\mathbf{x_q})$ interact linearly, i.e., via a dot product $\phi(\mathbf{x_p})^T\phi(\mathbf{x_q})$. We introduce kernel $K(\mathbf{x_q}, \mathbf{x_p}) = \phi(\mathbf{x_p})^T\phi(\mathbf{x_q})$ to represent the outcome of this operation.

We assume the variables $\phi(\mathbf{x_1}), ..., \phi(\mathbf{x_P})$ follow a multivariate normal distribution. $\phi(\mathbf{X}) = [\phi(\mathbf{x_1}), ..., \phi(\mathbf{x_P})]'$ represents the $P \times 1$ column matrix. Then we have

$$P(\phi(\mathbf{x_1}), ..., \phi(\mathbf{x_P})) = \frac{1}{(2\pi)^{\frac{P}{2}}|\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}\phi(\mathbf{X})'\Sigma^{-1}\phi(\mathbf{X})) \tag{4.12}$$

Following the derivations in chapter 2, we can estimate Σ^{-1} by solving a set of modified Lasso regressions. Formula 3.13 in chapter 2 now becomes:

$$\hat{\beta}_i = argmin\|\phi(x_i) - \sum_{j=i+1}^{p} \beta_{ji}\phi(x_j)\|^2 + \sqrt{\gamma\psi_i}\sum_{j=i+1}^{p}|\beta_{ji}| \tag{4.13}$$

As we discussed in this chapter , formula 4.13 can be re-formulated as formula 4.10 and solved by SVM regression solvers.

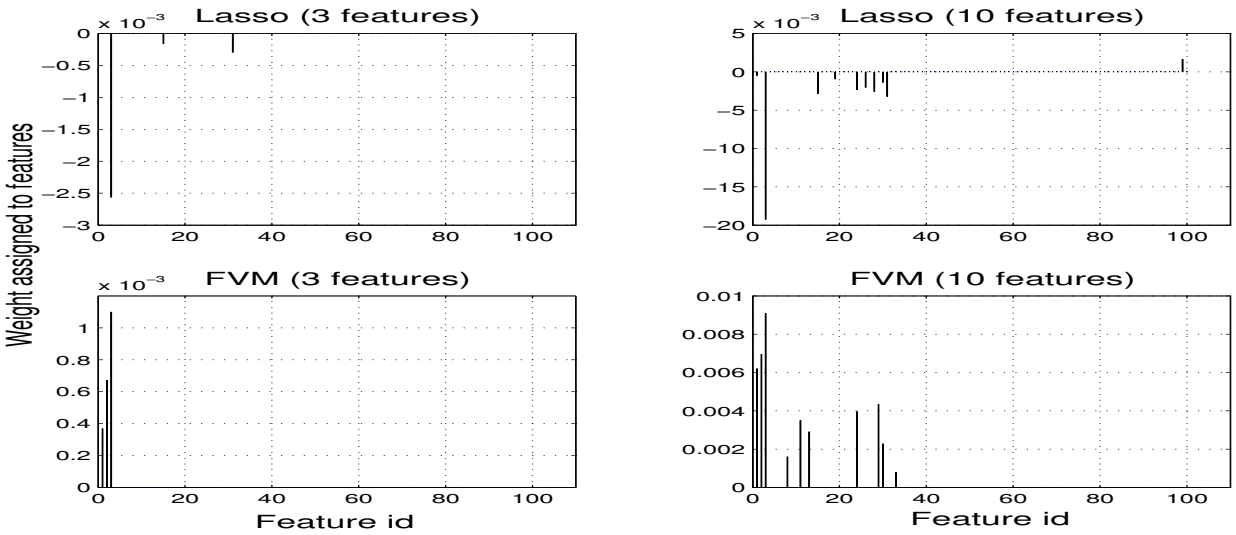

Figure 4.5: Results of FVM and the standard Lasso regression on this dataset. The X axis represents the feature IDs and the Y axis represents the weights assigned to features. The two left graphs show the case when 3 features are selected by each algorithm and the two right graphs show the case when 10 features are selected. From the down left graph, we can see that FVM successfully identified f_1,f_2 and f_3 as the three non-zero weighted features. From the up left graph, we can see that Lasso regression missed f_1 and f_2, which are non-linearly correlated with y. The two right graphs show similar patterns.

4.7 Summary

In this chapter, we have explored the mapping between the solution of SVM regression and Lasso regression and found that the solution of one model on a given dataset can be achieved by solving the other model on the transposed dataset. This finding sets up a bridge that allows the optimization techniques and algorithmatic variants developed for these two models to communicate with each other. Based on this exploration, we propose a generalization of Lasso regression named FVM. which extends our structure learning model to non-linear cases and enriches the current approaches for feature selection .

Chapter 5

Learning Gene Regulatory Networks

5.1 Background

A *gene regulatory network* usually refers the ensemble of DNA segments (e.g., genes, motifs) and proteins in a cell and the causal schemes (e.g., regulatory dependencies) according these elements interacting with each other and with other substances in the cell, thereby governing the rates at which genes in the network are transcribed into mRNA (Fig. 7.1). Automated induction of large genetic regulatory networks has been an open challenge in machine learning and computational biology. In recent years, advances in microarray technology have made possible the simultaneous monitoring of the mRNA levels of tens of thousands of genes from the entire genome under various conditions. If two genes are co-regulated by the same Transcriptional Factors (TFs), they will tend to have similar patterns of mRNA levels in different conditions in the microarray data. Thus it is possible to infer the structure of the gene regulatory network from microarray data if we treat each gene as a variable and treat the expression levels of all studied genes under a single condition as a (multivariate) sample.

Clustering-based algorithms are widely applied in microarray data analysis. The two most popular clustering methods, hierarchical clustering algorithm ([10]) and K-means clustering algorithm ([11]), have been used to find biologically meaningful gene clusters from expression data by many previous researchers. [2] and [41] have developed more advanced clustering-based approaches to discover genetic regulatory networks from multiple data sources.

Although clustering analysis is effective and straightforward, it has several limitations. It is often tricky to determine the number of clusters and the partial correlation (conditional independence relationship) between mRNA levels of genes cannot be modeled.

One solution to overcome this is to use a Bayesian network (BN), which is defined as a

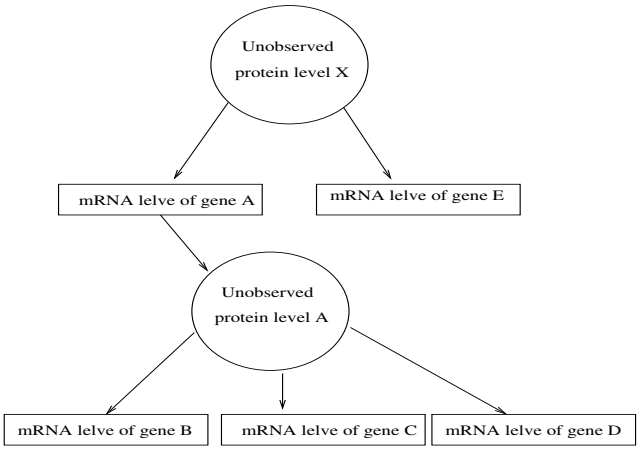

Figure 5.1: A typical sample of gene regulatory network

directed acyclic graph associated with local conditional distributions, as the model of gene network ([13], [18], [4]). Each node in a BN represents a gene and each directed edge represents an interaction between two genes. Several algorithms (usually called ”structure learning algorithms”), like Sparse Candidate Hill Climbing and Grow-Shrinkage algorithm, have been developed to learn the structures of BNs with thousands of nodes from data.

One problem of modeling a regulatory network as a BN lies on the fact that the protein levels are unobservable from microarray data (Fig. 7.1). Most BNs proposed so far only include mRNA levels of genes as nodes, but not protein levels. While the mRNA levels of co-regulated genes are often correlated, the mRNA levels of the regulated genes and regulator genes may not always correlate to each other because the mRNA expression levels of regulator genes are not always good indicators of their protein levels and activities. Even when the mRNA levels of regulator genes and regulated genes do correlate, the mRNA levels of co-regulated genes are generally not conditionally independent to each other given the mRNA levels of the regulator genes. Generally speaking, there are two kinds of dependencies among mRNA levels of genes in microarray data. One is the dependency between regulator genes and regulatee genes. The other is the dependency among co-regulated genes. A BN directly inferred from mRNA profiles may confound these two kinds of dependencies.

one way to distinguish these two kinds of dependencies is to use location analysis data in addition to microarray data. Location analysis identifies physical interactions between regulators and motifs on the regulatory regions of the genes. However, physical binding is not equivalent to regulatory interaction. Furthermore, location analysis itself is also very

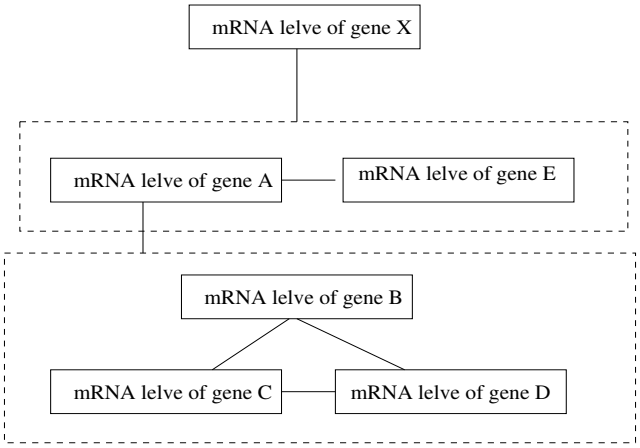

Figure 5.2: An undirected graph learned in our approach corresponding the regulatory network in Fig 1. The dot-lined boxes represent modules of co-regulated genes. The gene connecting to a module encodes the TF that regulates the genes in that module.

noisy. Thus, by combining these two data sources, we may recover modules more accurately than using either one alone. [18] and [4] have used genome wide location analysis data as priors to constrain the structures of BNs so that the edges corresponding to the dependencies of co-regulated genes are filtered out. Such an approach implicitly assumes the presence of (relatively strong) correlations between mRNA levels of the regulator genes and the regulated genes, whereas ignores the dependencies among genes that are co-regulated, which are often the strongest signals in microarray data and may be very useful.

5.2 Our approach

In this thesis, we integrate the microarray data and location analysis data in a different way. Our approach learns the genetic regulatory network in two steps.

- In the first step, instead of directly modeling the regulator-regulatee dependencies as in BNs, we only attempt to model the dependencies among co-regulated genes using graphical Gaussian model (GGM). Each edge of the undirected graph encoded by the GGM corresponds to the dependencies between a pair of co-regulated genes. Fig 5.2 shows an example of such a model. We use the structure learning approach in chapter 2 to learn a GGM from microarray data. Of course in practice our algorithm can not completely exclude edges that may correspond to the dependencies between regula-

tors and regulatees. In this case, we rely on the location analysis data to distinguish the two kinds of edges in the next stage.

- In the second step, we take the learned undirected graph together with a genome-wide location analysis dataset (i.e., the ChIp-ChIp data) as the input. A post-processing algorithm is developed to calculate the probability that genes connected in the estimated undirected graph have some shared TF set in location analysis data by chance. If this probability is less than a p-value threshold, we would infer these genes be regulated by the shared TF set.

The advantage of our approach lies on the fact that the dependencies among co-regulated genes are often much stronger and more robust than the dependencies between regulator genes and regulatee genes in microarray data. Thus by exploiting the co-regulation dependency information, which is not used in BNs, we may discover more regulatory patterns and we no longer need to assume the presence of detectable statistical correlations between mRNA levels of regulator genes and regulated genes.

5.2.1 The first step: Learning GGM from microarray data

In the first step we learn an undirected graph encoded by the GGM from microarray data. The edges in the undirected graph are supposed to correspond to the co-regulated gene pairs. The technical details of this step is presented in chapter 2.

5.2.2 The second step: the post-processing process

In the second step, we outline a method to identify the regulator genes and the modules of co-regulated genes from the estimated GGM with the help of genome-wide location analysis data (i.e., the ChIp-ChIp data).

Location analysis identifies physical interactions between regulators and motifs on the regulatory regions of the genes. It consists of a matrix whose rows are all the possible regulatee genes and columns are all the possible regulators. The elements of the matrix are the P-values reflecting the chance that a physical interaction happens between the regulator and regulatee gene.

For any gene x_i, let's assume there are K genes connecting to it in the previously estimated undirected graph. Let's suppose in these K genes, there are L genes that share some regulator set with gene x_i in the location analysis data. We use R to represent the shared regulator set. Now we want to calculate the probability that the $L + 1$ genes (L genes plus gene x_i itself) share the same regulator set R by chance. In this paper, we use a binomial distribution to approximate this probability.

Let's use N to represent the total number of genes. Let's use M to represent the number of genes which are possibly regulated by the regulator set R in the location analysis data. Then, the probability of a randomly selected gene being regulated by regulator set R is $\frac{M}{N}$.

Thus, in the $K+1$ genes (K genes connecting to gene x_i plus gene x_i itself), the probability that $L+1$ or more than $L+1$ genes share the same regulator set R by chance is

$$score = \sum_{j=L+1}^{K+1} \frac{(K+1)!}{(K+1-j)!j!}(\frac{M}{N})^j(1-\frac{M}{N})^{K+1-j}. \tag{5.1}$$

If this score is lower than a pre-defined p-value threshold (we use 0.05 in this paper), we will infer regulator-regulatee dependencies between regulator set R and these $L+1$ genes.

The pseudo code of the post-processing algorithm is shown as Algorithm 3.

Algorithm 4 Pseudo code of the post-processing algorithm

1. For each gene x_i
   (a) Get all the genes connecting to gene x_i in the estimated undirected graph. Use Neighbor(x_i) to represent this gene subset. Suppose the size of Neighbor(x_i) is K.
   (b) From the protein-binding location analysis data, find all the TFs that are shared by x_i and one or more genes in Neighbor(x_i). For each shared TF set R
      i. Suppose there are N genes in total. Suppose there are M genes regulated by TF set R. Suppose in Neighbor(x_i), there are L genes regulated by TF set R. Use $S(R, \text{Neighbor}(x_i))$ to represent these L genes. Use formula 5.1 to calculate a p-value. If this p-value is less than a p-value threshold, then we would infer that the gene x_i and genes in $S(R, \text{Neighbor}(x_i))$ form a co-regulated module, which is regulated by genes in TF set R

5.3 Experiments

We applied our method on the Yeast Saccharomyces cerevisiae dataset. The expression microarray data we used comes from [43]. The mRNA expression levels of 6177 genes are measured under 76 conditions. The location analysis data we used comes from [27]. It gives the p-values of genes regulated by each of the 106 TFs. When a p-value is lower than 0.05, we would assume it is possible that the gene is regulated by the TF.

In our experiment, the value of the hyper-prior γ is assigned to be 2.31 so that the correlation coefficients between zero weighted variables and the regression residue are no larger

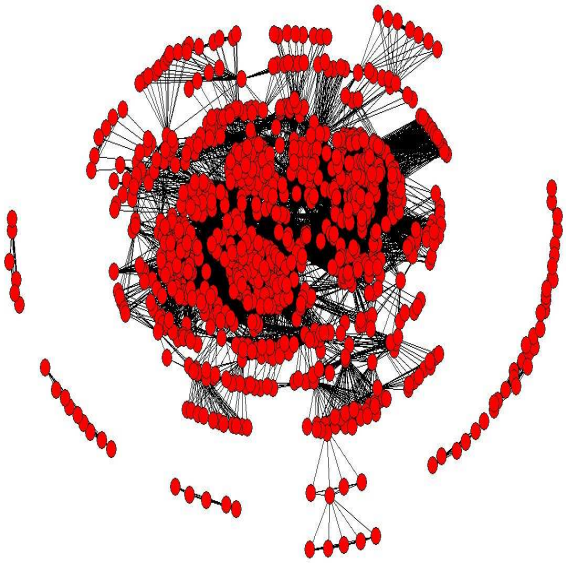

Figure 5.3: This is an undirected graph that is corresponding to all the gene regulatory modules(whose sizes are required to be no less than five) found by our algorithm. Each node represents a gene and each edge represents a co-regulated relationship between two genes. Each regulatory module corresponds to a clique. Notice that one gene can be in multiple regulatory modules. That's why many cliques are connected in the figure. The graph clustering coefficient is 0.433.

than 0.01. Another parameter is the p-value threshold of the post-processing algorithm, which can be used to control the number of output gene modules. Here we also set the p-value threshold to be 0.05.

Directly evaluating the estimated gene regulatory network is difficult because the true gene network is usually unknown. One possible solution is that, instead of directly evaluating the estimated gene regulatory network, we evaluate the gene regulatory modules in the estimated gene network. A gene regulatory module is defined to be a set of co-regulated genes sharing the same set of regulator genes. Genes in the same module are likely to attend the same cellular processes and have similar biological functions. Most yeast genes have been annotated with certain cellular processes or functions in GO database([1]). Thus it is possible to use this information as an external evidence to evaluate the estimated gene regulatory modules. In fact, such evaluation strategies have been widely used in recent years ([41]).

The details of the evaluation strategy are summarized as following. For each gene module and each GO annotation, we calculated the fraction of genes in that module associated

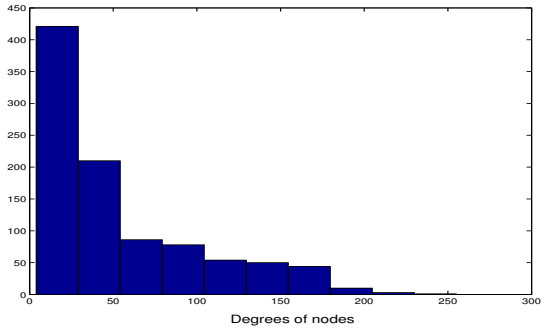

Figure 5.4: This is the histogram that shows the distribution of node degrees in figure 5.3. The X axis represents the degree of nodes. The Y axis shows the number of nodes with such degrees in each bin. We can see that the histogram approximately follows power law distribution.

with that GO annotation and used the hypergeometric distribution to calculate a p-value for this fraction (which can be done using SGD Term Finder Toolkit([1]), and took p-value < 0.05 to be significant. We compared the enrichment of modules for GO annotations between results achieved by our undirected graph structure learning algorithm and results achieved by other baseline algorithms. In this paper, we only consider GO annotations in "Process" category.

5.3.1 Co-regulated gene modules recovered by our method

We list the regulatory modules found by our approach with at least 2 TFs and five regulated genes in table 5.1.

Each line in table 5.1 represents a regulatory module found by our approach. The first column gives the regulator set of this module. The second column gives the number of regulated genes with annotations in GO database VS the number of all the regulated genes in this module. The third column gives the previous reference of the interactions among regulator genes in this module. The last column gives the GO annotations of processes shared by the regulated genes in this module (the names of the regulated genes are not shown to save space). We also give the P-values for these GO annotations, which are calculated by SGD Gene Ontology Term Finder Toolkit([1]). This value reflects the probability that the genes in the module share the GO annotation by chance. Since there are generally multiple GO annotations shared by a gene cluster, we only show the GO annotation with the smallest P-value for each module in the last column. We can see many of the co-regulators in our recovered modules have already been reported as co-TFs or have interactions by previous references. Some well studied co-TFs like (MBP1, SWI6) and (FKH2, MCM1, NDD1) can

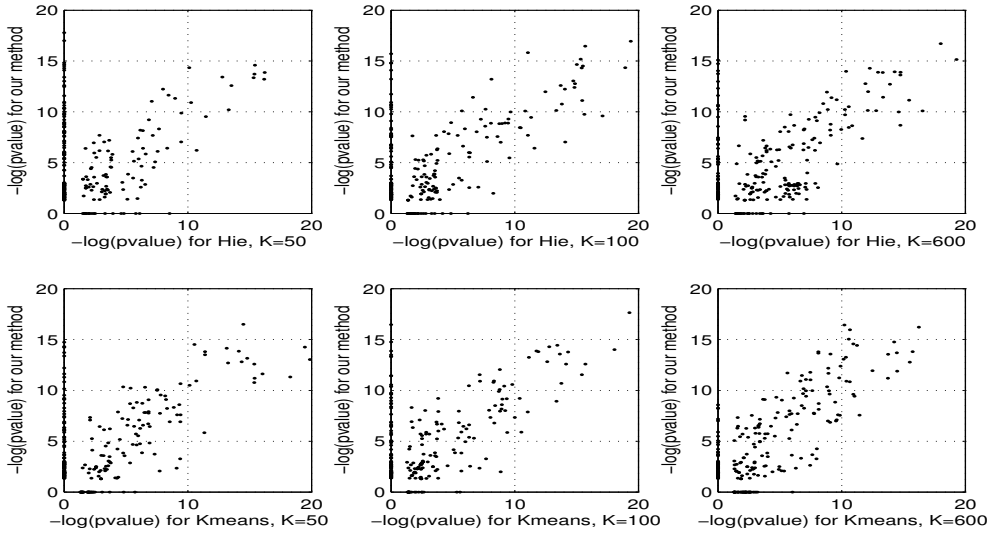

Figure 5.5: Comparison between our approach and Hierarchy clustering (in the first row) and K-means clustering (in the second row), with k=50 (first column), 100 (second column), 600 (third column) respectively. In the up left figure, there are 109 dots on the Y axis and 22 dots on the X axis. In the up middle figure there are 121 dots on the Y axis and 19 dots on the X axis. In the up right figure, there are 91 dots on the Y axis and 22 dots on the X axis. In the down left figure, there are 119 dots on the Y axis and 29 dots on the X axis. In the down middle figure, there are 112 dots on the Y axis and 27 dots on the X axis. In the down right figure, there are 92 dots on the Y axis and 39 dots on the X axis.

also be found in table 5.1.

There are a lot of other interesting patterns found by our approach that are not listed in table 5.1. For example, our approach found the module with regulator set (CAD1,PHO4) and the regulated gene set (CUP1-1,CUP1-2). This small module corresponds to the process "response to copper ion" on a significance level with $pvalue = 3.02e-07$ (both CUP1-1 and CUP1-2 belong to this process, while in all of the 7276 annotated genes in the GO database, only 4 genes belong to this process).

In figure 5.3, we plot the undirected graph that is corresponding to all the gene regulatory modules(whose sizes are required to be no less than five) found by our algorithm. There are 957 nodes in this figure. Each node represents a gene and each edge represents a co-regulated relationship between two genes. Thus each regulatory module corresponds to a clique. Notice that one gene can be in multiple regulatory modules. That's why many cliques are connected. The graph clustering coefficient is also given in figure 5.3. In figure 5.4, we plot the histogram that shows the distribution of node degrees for the undirected

Table 5.1: **modules with multiple regulators and more than 5 regulated genes.**

regulator set	annotated/ALL	reference of co-regulators	common processes or function of regulated genes
ACE2,SWI5	6/6		4/6 cell proliferation, p=0.023
ASH1,FKH2	4/5		2/4 cell wall organization and biogenesis, p=0.002
ASH1,SWI4	5/5		3/5 cell organization and biogenesis(p=5.94e-05),
CIN5,MET4	6/8		2/6 copper ion import, p=0.0002
DIG1,STE12	10/10	functional and physical	10/10 cellular process(p=1.45e-05),
FHL1,RAP1	100/100	share motif(Davide et al)	100/100 structural constituent of ribosome, $p < 1e-4$
FKH1,FKH2	4/5	co-TFs(Kato et al 2004)	2/4 covalent chromatin modification, p=0.0003
FKH2,MCM1,NDD1	12/12	co-TFs(CYGD)	6/12 cell proliferation, p=0.01
GAT3,RGM1	3/13		2/3 telomerase-independent telomere maintenance, p=9.56e-06
GCR1,GCR2,RAP1	6/6		6/6 energy pathways, p=1.65e-08
HIR1,HIR2	6/6	co-TFs(Spector et al 1997)	6/6 chromatin assembly or disassembly, p=1.23e-14
HSF1,MSN4	6/8	co-TFs(Jeffrey et al, 2002)	4/6 response to stress, p=9.62e-05
INO2,INO4	6/7	co-TFs(BRIAN et al 1995)	2/6 translation elongation factor activity, p=4.76e-05
MAL13,MSN4,RGM1	5/7		3/5 telomerase-independent telomere maintenance, p=1.05e-6
MBP1,SWI4	8/8		3/8 microtubule cytoskeleton organization and biogenesis, p=0.005
MBP1,SWI6	32/37	fun-phy interact(CYGD)	21/32 cell cycle, p=4.74e-17
MET31,MET4	8/8	same complex(Pierre et al 1998)	6/8 sulfur metabolism, 9.78e-11
MET4,RAP1	6/6		6/6 macromolecule biosynthesis, p=4.24e-06
NDD1,SKN7	5/5		5/5 cell organization and biogenesis, p=9.35e-05
NDD1,SWI4	5/5		2/5 cytoskeleton organization and biogenesis, p=0.012
SWI4,SWI6	15/19	fun-phy interact(CYGD)	7/15 cell cycle p=3.57e-05

graph in figure 5.3. We can see that the histogram approximately follows power law distribution.

5.3.2 Comparison between our approach and other approaches

[2] has used GRAM algorithm, which can be thought as a kind of clustering algorithms, to extract gene regulatory modules from expression microarray data and location analysis data. In particular, they have used the same genome-wide location analysis data as the one we used. However, their expression microarray data is over 500 experimental conditions

Table 5.2: **The comparison between our method and several baseline approaches.**

	assigned by our method but missed by baselines	assigned by baselines but missed by our method
Gram algorithm	93	61
Bayesian Network	88	29
K-means Clustering	109	22
Hierarchical Clustering	119	29

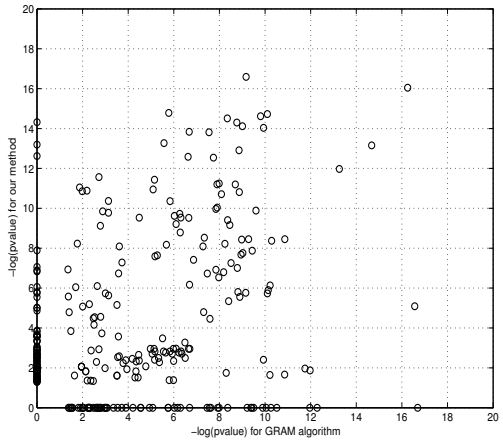

Figure 5.6: Comparison between our method and Gram algorithm in GRAM algorithm. There are 93 dots on the Y axis and 61 dots on the X axis.

and our microarray data, with 76 conditions, is only a subset of theirs. * We will use their result as a baseline result.

We set the requirement that the size of a gene module should be no less than five. The p-value threshold of our post-processing algorithm, which can be used to control the number of output gene modules, is set to be 0.04 so that the number of output gene modules is 106, which is exactly the same as the number of gene modules in the results reported by [2]. In this way, we can compare our results with [2] conveniently.

We also compared our results to the results achieved by hierarchical clustering ([10]) and k-means clustering algorithms ([11]). Since the number of clusters is a parameter that needs to be pre-defined in these two clustering algorithms, we tried several settings (k=50, 100, and 600) for this parameter. For the Hierarchical clustering method, we used the agglomerative average linkage version. For the K-means clustering method, we tried 5 random restarts and only reported the best result. Notice that clustering algorithms can be only applied on expression microarray data to group correlated genes. They cannot make use of location analysis data themselves. In order to be fair in comparing our method with these two clustering methods, we need some post-processing algorithms that can combine the clustering results with location analysis data. In fact, we can simply treat the clustering results as a disjoint undirected graph. Genes in same clusters form complete sub-graphs and

*[2] combined several microarray data sources in different conditions to get a big microarray data over 500 conditions in their paper. Thus the cell cycle microarray data from [43], which we used in our paper, is a only a subset of it.

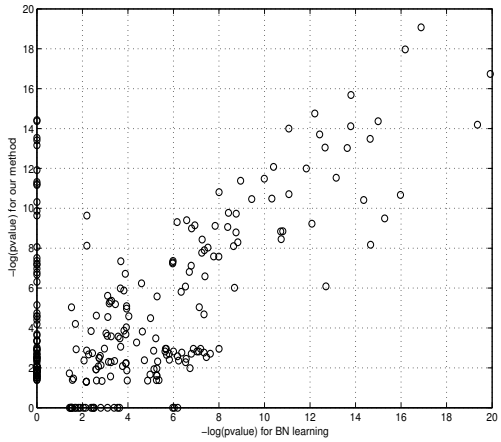

Figure 5.7: Comparison between our method and BN learning algorithm. There are 88 dots on the Y axis and 29 dots on the X axis.

genes in different clusters are disconnected with each other. Thus we can directly use the post-processing algorithm we described before to post-process clustering results. In order to be fair in comparing the two module extraction methods using this evaluation strategy, the number of gene regulatory modules found by these two methods should be the same. Thus for the two clustering methods, we tuned the p-value threshold of the post-processing algorithm so that it also returns 106 modules. In this way, we can give a fair comparison.

We further compared our approach with the BN learning approaches that used the location analysis data as priors to constrain the BN structures (([13], [18] and [4])). We used sparse candidate hill climbing as the search algorithm for BN learning and only allow edges from a regulator gene to a regulatee gene when the p-value between these two genes is lower than 0.05 in location analysis data. We tune the weight of the penalizer for the number of edges in the BN so that it will also generate 106 regulatory modules.

The results are shown in figure 5.5, 5.6 and 5.7. In these figures, each dot represents a GO annotation. If it is on the Y-axis, it means that the GO annotation has been enriched in our results but not in the baseline results. If it is on the X-axis, it means that the GO annotation has been enriched in the baseline results but not in our results. We can see that there are many more dots on the Y-axis than on the X-axis in figure 5.5, 5.6 and 5.7, which implies our method achieved a much better performance. Notice the microarray data used in [2] contains richer information than our data, but their results are still not as good as ours, as reflected in figure 5.6.

We also generated the table 5.2, which gives the comparison between our method and

several baseline approaches. The first column in the table shows the names of the baseline method. The second column shows the number of GO annotatoins correctly assigned by our method but missed by the baseline methods. The third column shows the number of GO annotatoins correctly assigned by the baseline methods but missed by our method.

5.4 Summary

In this chapter, we proposed a new formalism based on Graphical Gaussian models to learn gene regulatory networks from microarray data and location analysis data. Overall, our approach provides an efficient and biological more informative and comprehensible (i.e., yielding sparse network and gene modules) solution to the problem of gene network inference. We applied our approach to a real microarray dataset and a protein-binding location analysis dataset. An evaluation on the basis of consistency with the GO annotations shows our results significantly better than the results of clustering-based methods and BN learning methods.

Chapter 6

Learning Category Network to Help Text Classification Tasks

6.1 Background

In the multi-class text categorization problem, each document may belong to more than one categories. In this chapter, we use the phrase "category network" to represent the dependencies among different category labels. Each node in the network corresponds to a category and each edge corresponds to the dependency between categories. When category labels are not independent to each other, it is possible to exploit the category network to improve text categorization performance.

Sometimes documents are organized in a large number of categories and the categories are arranged in a hierarchal tree. If a document belongs to a node in this tree, then it must also belong to all the ancestor nodes in the hierarchal structure. And the category nodes in the same level are mutually exclusive. The U.S. patent database and Yahoo hierarchy are two such examples. Many previous researchers have studied how to use such pre-existing hierarchal structure to help text categorization.

In more general scenarios, the connections among categories may go beyond a hierarchal tree. For example, each paper published by ACM Press has mutiple "category codes" that help to identify the area of the paper. A paper could belong to two categories on two different taxonomies. In this case, the category network becomes a general graph instead of a hierarchical tree. The Reuters 2001 dataset is another example, where documents are classified into three taxonomies: Topic categories, Industrial categories and Geographical categories respectively. One can view the dependencies among these categories as in a graph where the structures of the networks are not observed in general and can only be

learned from data.

In this chapter, we focus on the scenarios where the category networks are general graphs rather than pre-defined hierarchal trees. Our task is to first extract a category network from textual corpus and then use this category network to help text categorization:

- **The first step: Learning category network from textual corpus.** Suppose there are N documents and P category labels in the dataset. Each document has one or more category labels. We treat each document as a sample and each category label as a node variable. Thus we get a $N \times P$ data matrix. If the ith document belongs to the jth category, then the (i, j)th element in the matrix is set to be 1. Otherwise it is set to be zero. We use the approaches in chapter 3 to learn an undirected graph from the textual corpus. The edges in the undirected graph are supposed to correspond to the correlated category label pairs.
- **The second step: Use the category network to help text classification.** Once a category network is learned, we can exploit it to help text classification. Suppose one document is assigned a label A in the classification process, then this document would tend to be assigned other labels that are connected with category A in the category network.

We will discuss how to conduct text classification with category networks in detail in the next section.

6.2 Text classification with category networks

.

6.2.1 Review of graph-based semi-supervised learning approaches

Many researchers studied the problem of classification with graph structures. For example, graph-based semi-supervised learning approaches have been widely used for classification tasks on textual data ([3], [51], [46], and [52]). This kind of approaches have several important advantages compared to other alternatives. First, their objective functions can be solved easily by gradient based methods, no matter how complex the graph structure is. Second, these approaches assign real-value scores to examples in the test set, which makes them easy to be tuned to maximize metrics like F1 performance.

The key idea of graph-based semi-supervised learning approaches lies on an additional term (often called graph regularizer) in their objective functions, compared with traditional classifiers like logistic regression or ridge regression. This term encourages the examples connected to each other in a graph to receive similar scores and be assigned same labels.

Text classification with document networks

Consider the problem of predicting unknown labels for the testing examples based on their input vectors. The observed data includes a training set $(\vec{x}_i, y_i)$ for $i = 1, ..., n$ and a test set $(\vec{x}_j)$ for $j = n+1, ..., m$. Here $\vec{x}_i$ is the input vector of the ith document and scalar $y_i \in \{-1, 1\}$ is the class label of the ith example. The true labels y_j for the test set $(\vec{x}_j)$ are unknown and to be predicted.

[52] applied the graph-based semi-supervised learning approach on Reuters corpus for text classification tasks. They first constructed a network using word level similarity among documents. Each node in the network represents a document (in the training or testing set) and each edge represents a document pair whose cosine similarity is larger than a threshold. After the document network was induced, [52] reinforced continuity among category labels of connected documents by using the following objective function:

$$\hat{f} = \arg\min_{f \in R^m} \sum_{i=1}^{n} loss(f_i, y_i) + \lambda f^T \Delta f \tag{6.1}$$

In formula 6.1, f_i represents the predictive score assigned to the ith document. These scores are the parameters to be solve to minimize the objectuive function. If f_i is larger than a threshold, then the ith document would be assigned to the positive category. Otherwise it will be assigned to the negative category. $loss()$ represents the loss function measuring how good the values fit the training data. A detailed analysis of loss functions can be found in [28]. It is set as $loss(f_i, y_i) = (f_i - y_i)^2$ in [52]. Matrix Δ, namely the graph-Laplacian matrix, is an $m \times m$ matrix that encodes the document network structure, defined as $\Delta = D - W$. Matrix $W = [w_{ij}]$ is the $m \times m$ matrix with each w_{ij} representing the weight of the edge connecting document i and document j. If there is no edge between these two documents, then $w_{ij} = 0$. Matrix D is defined as the $m \times m$ diagonal matrix whose diagonal elements are $d_{ii} = \sum_{k=1}^{m} w_{ik}$.

More explicitly, formula (6.1) can be re-written as

$$\hat{f} = \arg\min_{f} \sum_{i=1}^{n} loss(f_i, y_i) + \lambda \sum_{(i,j) \in E} w_{ij}(f_i - f_j)^2. \tag{6.2}$$

It's obvious that the graph regularizer $\lambda \sum_{(i,j) \in E} w_{ij}(f_i - f_j)^2$ encourages f_i and f_j to be close to each other if document i and document j are connected.

Text classification with hyperlink networks

[51] applied the graph-based semi-supervised learning approach on Yahoo corpus and We-

bKB data for web page classification tasks. They reinforced continuity among category labels of connected web pages by the following objective function:

$$\hat{f} = \arg\min_{f \in R^m} \sum_{i=1}^{n} loss(f_i, y_i) + \lambda' f^T K^{-1} f + \lambda f^T \zeta f \tag{6.3}$$

ζ is the graph-Laplacian matrix and it plays similar roles as the Δ matrix in formula 6.1. The only difference is that ζ used in [51] encodes the hyperlink network while Δ used in [52] encodes the document network based on word-level similarities.

Compared to formula 6.1, formula 6.3 has one additional term $\lambda' f^T K^{-1} f$. The matrix K is an $m \times m$ kernel gram matrix that is constructed using the inner-products of the "bag of words" representation of web pages. It is obvious that this term encourgages the web pages with similar content to receive similar predictive scores.

Also note that without the graph regularizer term $\lambda f^T \zeta f$, formula 6.3 will reduce to standard ridge regression classifiers or logistic regression classifiers (depending on which loss function is actually used).

6.2.2 Our method

Both [52] and [51] transferred the multi-class categorization problem into a set of one-vs-all binary classification problems. Each binary classifier makes decisions separately and the dependencies among category labels have not been considered.

In this section, we use the graph-based semi-supervised learning approach to classify documents with learned category networks. Instead of constructing multiple one-vs-all binary classifiers, we predict document labels for all the categories simultaneously. This is our unique contribution compared to the previous works including [52] and [51].

Consider the multi-class text categorization problem. Suppose there are L categories in total. The observed data includes a training set $(\vec{x}_i, \vec{y}_i)$ for $i = 1, ..., n$ and a test set $(\vec{x}_j)$ for $j = n+1, ..., m$. Vector $\vec{y}_i = (y_{i1}, ..., y_{iL})$, where scalar $y_{ik} \in \{-1, 1\}$ is the class label of the ith example for the kth category. The true labels $\vec{y}_j$ for the test set $(\vec{x}_j)$ are unknown and to be predicted. In the multi-class text categorization task, people often estimate a predictive score f_{ik} for each category k and each example $\vec{x}_i$ in the test set. If this score is larger than a threshold, then the corresponding example would be assigned to the kth category. We use the $n \times 1$ matrix $f_k = [f_{1k}, ..., f_{nk}]^T$ to represent the predictive scores for the kth category and use the $L \times 1$ matrix $f_i = [f_{i1}, ..., f_{iL}]^T$ to represent the predictive scores for the ith example.

We are interested in the case that in addition to the input vectors, a category network is also given. Each node in the network corresponds to an category and each edge in the network corresponds to some dependency between these two categories. Note that our

category networks are very different from the document networks or hyperlink networks in [52] and [51] since in document networks or hyperlink networks, each node corresponds to a document and each edge corresponds to the dependency between two documents.

The objective function of our method is defined as:

$$\hat{f} = \arg\min_{f} \sum_{k=1}^{L} \sum_{i=1}^{n} loss(f_{ik}, y_{ik}) + \lambda \sum_{k=1}^{L} f_k^T K^{-1} f_k + \lambda' \sum_{i=1}^{n} f_i^T \zeta f_i. \tag{6.4}$$

The matrix K is an $m \times m$ kernel gram matrix that is constructed using the inner-products of the "bag of words" representation of documents. The matrix ζ is an $L \times L$ graph-Laplacian matrix that encodes the category network obtained by applying our graph learning method in chapter 3. The construction and the role of ζ is similar to what we described in section 6.2.

Constants λ and λ' are regularization parameters that are treated as constants here. Equation 6.4 can be reduced to a standard ridge regression classifier when λ' is set to be zero because K is a linear kernel and $loss(f_{ik}, y_{ik})$ is the least squares loss.

Equivalently, we can also write formula (6.4) as

$$\hat{f} = \arg\min_{f} \sum_{k=1}^{L} \sum_{i=1}^{n} loss(f_{ik}, y_{ik}) + \lambda \sum_{k=1}^{L} f_k^T K^{-1} f_k + +\lambda' \sum_{i=1}^{n} \sum_{(k,q)\in E} w_{kq}(f_{ik} - f_{iq})^2 \tag{6.5}$$

Here E represent the set of edges in the category network. w_{kq} represent the edge weights, which are the off-diagonal elements in matrix W.

There are three terms in the objective function in formula 6.5. The first two terms do not introduce the interaction between different categories. Without considering the third term, formula 6.5 will reduce to a standard classifier (logistic regression or ridge regression classifier, depending on which loss function is used). The third term in formula 6.5 encourages the predictive score assigned for connected categories to be close to each other.

6.3 Experiments

We conducted experiments on Reuters 21578 corpus, which has been a benchmark in text classification evaluations. Note that there are many other datasets like OSHMED and RCV1 corpus, which may also be suitable for our approach. We leave the exploration of these datasets to future works.

The Reuters 21578 corpus contains 7769 training documents and 3019 testing documents. There are 90 categories in the dataset in total and each document may belong to one or more categories. We first applied our structure learning algorithm on the corpus and

learned a category network with 90 nodes and 20 edges. This network is then exploited to help text categorization using the approach described in the previous section. There are 23 categories linked to each other by the edges in the learned category network. We compared the F1 classification performance of these categories achieved by our approach with baselines achieved by standard ridge regression classifier.

In our experiments, we set w_{kq} in formula 6.5 as $w_{kq} = \lambda_k \lambda_q e_{kq}$. Here e_{kq} represents the edge weight we learned from our structure learning algorithm. λ_k and λ_q are the scaling parameters for the kth and qth category. It's obvious that when λ_k is zero, then the classification decisions for the kth category will not be influenced by other categories.

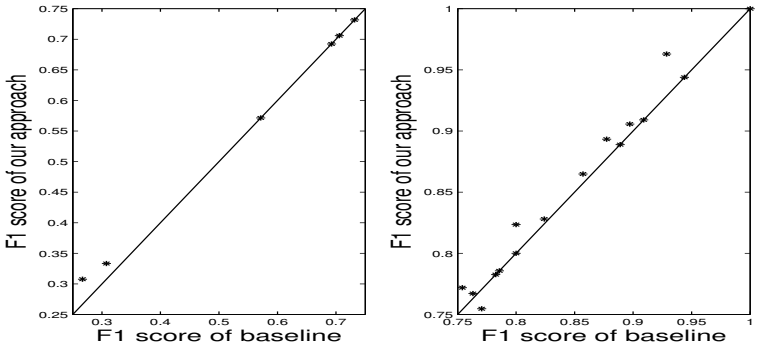

Figure 6.1: Each star in the graph represents a category in our experiments. The Y axis value of a star represents the F1 score of this category achieved by our approach and the X axis value represents the F1 score achieved by the baseline ridge regression classifier. In order to show the graph clearly, we split the graph into two parts. The left subgraph shows the categories with F1 score from 0.25 to 0.75 and the right subgraph shows the categories with F1 scores larger than 0.75. There are 23 categories in these two graphs. We can see that 11 stars are above the diagonal line and only 1 star is below the diagonal line.

We tuned the the parameter λ_k for every category k in formula 6.5 on the training data by two-fold cross-validation. On some categories, this λ_k values could be tuned to zero. For these categories, our approach will reduce to standard ridge regression and have the same classification performance as the baseline. Only for the categories with non-zero λ_k values, our results are different from the baseline.

In our experiments, 11 of the 23 categories have non-zero tuned λ_k values thus they have different F1 scores compared to the baseline. Our approach has outperformed the baseline on 10 out of these 11 categories. The T-test shows that our improvement is significant with a P-value less than 0.01. We plot the results in figure 6.1 and 6.2.

In figure 6.1, each star in the graph represents a category in our experiments. The Y axis value of a star represents the F1 score of this category achieved by our approach and the X

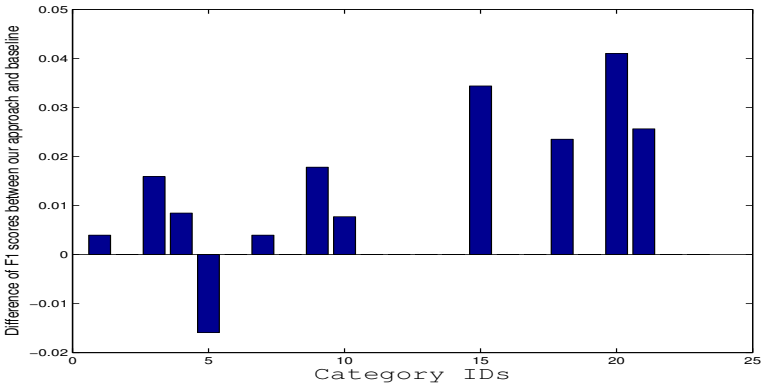

Figure 6.2: Each bar in the graph represents a category. The Y axis value of a bar represents the difference between the F1 score achieved by our approach and by the baseline ridge regression. We can see there are 23 bars in the graph. 11 of them have positive Y axis values and only 1 has negative value.

axis value represents the F1 score achieved by the baseline ridge regression classifier. In order to show the graph clearly, we split the graph into two parts. The left sub-graph shows the categories with F1 score from 0.25 to 0.75 and the right sub-graph shows the categories with F1 scores larger than 0.75. There are 23 categories in these two graphs. We can see that 11 stars are above the diagonal line and only 1 star is below the diagonal line.

In figure 6.2, each bar in the graph represents a category. The Y axis value of a bar represents the difference between the F1 score achieved by our approach and by the baseline ridge regression. We can see there are 23 bars in the graph. 11 of them have positive Y axis values and only 1 has negative value.

6.4 Summary

In this chapter, we proposed a multi-class text categorization method that can explore the relationships among category labels and outperform binary classifiers using one-vs-all strategy. The classification process includes two steps. In the first step, we learn a category network from textual corpus using our structure learning algorithm, in which each node represents a category label and each edge represents the correlation between two categories. In the second step, we use the network to help text categorization in a graph regularization

framework.

Overall, our approach provides an effective way to improve text categorization performance by exploiting the rich relationship among category labels. We applied our approach to the Reuter 21578 dataset and observed significant improvement compared to the baseline.

Chapter 7

Structure based Classification with Graphs containing Annotated Edges

7.1 Background

Automatic classification with graphs containing annotated edges is an interesting problem and has many potential applications. For example, web-graph hyperlink structures can be used in web page categorization. Hyperlinks can be thought as edges annotated by their associated anchor text and can be used to improve web-page classification performance. In social networks, participants often annotate their network connections thus it is natural to study learning methods that can take advantage of graphs with annotated edges. In gene regulatory networks, co-regulated gene pairs are connected by undirected edges, which are annotated by the proteins that regulate these genes. These edges, together with their annotation information, can help us to predict functions for unknown genes. Other examples include protein networks and citation networks, etc.

Graph-based semi-supervised learning approaches, which have been discussed in the previous chapter, are widely used for classification tasks with graph structures. However, these approaches did not exploit the annotation information associated with the edges in the graph. For example, in [51], it is assumed that all the edges in hyperlink structures have the same positive weight. This assumption, however, may not always yield the best result from a classification standpoint. The importance of these edges could be very different in terms of classification, depending on their associated annotations. Furthermore, sometimes the edges in hyperlink structures may have negative weights. For example, in the catalog and product web page classification task, the catalog pages often connect to product pages but seldom connect to other catalog pages.

A simple way to use annotation information is to pass edge annotations to node features. That is to treat the annotations of an edge as additional features of the two node connected by this edge. Although, by doing so, we have merged the edge features into node features and lost part of the useful information, it is nevertheless non-trivial to devise a method that can perform better than this simple baseline. In order to derive a better learning method, we propose a novel objective function in the graph regularization framework to exploit the annotations on the edges. The underlying idea is to create regularization for the graph with the assumption that the likelihood of two nodes to be in the same class can be estimated using annotations of the edge linking the two nodes. In our model, the importance of each edge is measured by the sum of the weights of edge annotation features associated with this edge. The edge annotation feature weights are parameters in our objective function and can be learned efficiently in this framework.

Our main contribution in this line is to derive a new formulation that could learn the edge importance from the annotations on the edges within a graph regularization framework. A scalable numerical algorithm is then presented to solve the problem. We have conducted experiments on several datasets and compared with several baseline approaches. The empirical results suggest that our approach performs significantly better than other approaches. Although we focus on web-page categorization in our experiments, our approach can be directly applied to general graph classification problems with annotated edges.

7.2 The Algorithm

Consider the problem of predicting unknown labels for the testing examples based on their input vectors. We are interested in the case that in addition to the training set $(\vec{x}_i, y_i)$ for $i = 1, ..., n$ and the test set $(\vec{x}_j)$ for $j = n + 1, ..., m$, a graph is also observed on the whole dataset. Graph-based semi-supervised learning approaches have been widely used in such tasks. Let's recall the objective function used in [51], which we have discussed in the previous chapter.

$$\hat{f} = \arg\min_f \sum_{i=1}^{n} loss(f_i, y_i) + \lambda f^T K^{-1} f + \lambda' \sum_{(i,j)\in E} w_{ij}(f_i - f_j)^2. \tag{7.1}$$

Here w_{ij} represent the edge weights, which are set to be the constant 1 in [51]. As we discussed before, this objective function has several limitations. Importantly, the edge weights w_{ij} in formula (7.1) have to be chosen a priori. In practice, they are often set to be a constant. We can not directly treat w_{ij} as learnable parameters in our model because otherwise, the trivial solution is $w_{ij} = 0$ (that is, the graph information is ignored). Moreover, w_{ij} can not be set as a negative number because otherwise, the solution of $[f_i]$ could become

infinity. However, in some problems, negative edge weights can be helpful when values of two connecting nodes are anti-correlated.

In order to overcome the limitations of (7.1), we propose a novel objective function in this paper so that the edge importance (or weights) can be automatically learned. Our method is based on the idea that the likelihood of two nodes to be in the same class can be estimated from annotations of the edge linking the two nodes. We use this idea to design regularization conditions with hyper-parameters depending on the annotations, and the hyper-parameters can be jointly optimized with f in a unified optimization problem. Specifically, we consider the following objective function as an instantiation of the above general idea:

$$\begin{aligned} \hat{f} &= \arg\min_{f,\vec{\gamma}} \sum_{i=1}^{n} loss(f_i, y_i) + \lambda f^T K^{-1} f + \lambda' \sum_{(i,j)\in E} (f_i f_j - w_{ij})^2 \\ &\quad \text{such that } w_{ij} = \vec{\gamma} \cdot \vec{a}_{ij}. \end{aligned} \tag{7.2}$$

Here w_{ij} are the edge importance to be learned from the edge annotation features. Vector $\vec{a}_{ij} = [a_{ij,1}, \ldots, a_{ij,p}]$ represents the edge-annotation features associated with the edge (i, j). Vector $\vec{\gamma} = [\gamma_1, \ldots, \gamma_p]$ is the hyper-parameter to be learned from the data, , which represents the weights of the annotation features (for predicting edge importance w_{ij}). Note that we are only considering the case when the $loss()$ is set as the least squares loss. If other loss functions are used, then the third term of formula 7.2 need to be modified accordingly.

Since we are using the least squares loss function with $y_i \in \{\pm 1\}$, the value of $f_i f_j$ is encouraged to be close to ± 1. In this case, formula (7.2) has a very clear intuitive meaning. If we consider regularization of the form $(f_i f_j - w_{ij})^2$ with unknown annotation dependent parameter w_{ij}, then the optimal value of w_{ij} is the conditional expectation: $w_{ij} = E(f_i f_j | \vec{a}_{ij})$. In the proposed method, this value is estimated using a linear estimator of the form $\vec{\gamma} \cdot \vec{a}_{ij}$. If f_i and f_j only take values in $\{\pm 1\}$, then $E(f_i f_j | \vec{a}_{ij}) = 2P(f_i f_j = 1 | \vec{a}_{ij}) - 1$. This has the following desirable effect: given the edge annotation, if f_i and f_j are likely to have the same sign, then the objective function will strengthen this trend with a positive w_{ij}; if f_i and f_j are likely to have different signs, the objective function will strengthen the trend with a negative w_{ij}.

With the above interpretation in mind, we can simplify (7.2) as follows.

$$\hat{f} = \arg\min_{f,\vec{\gamma}} \sum_{i=1}^{n} loss(f_i, y_i) + \lambda f^T K^{-1} f + \lambda' \sum_{(i,j)\in E} (f_i f_j - \vec{\gamma} \cdot \vec{a}_{ij})^2. \tag{7.3}$$

The first term of the formula encourages f_i values for the training examples to be close to y_i (either 1 or -1). The third term in the above formula encodes the graph structure and

plays the role as a graph regularizer. The difference between formula (7.3) and the original objective function in formula (7.1) is only the third term, which is the graph regularizer. The new formulation enables us to find non-trivial solutions for the edge importance w_{ij}, which is estimated using a linear combination of edge annotation features. The edge importance $w_{ij} = \vec{\gamma} \cdot \vec{a}_{ij}$ is positive when f_i and f_j have the same signs (both close to 1 or both close to -1), and negative when f_i and f_j have different signs (one is close to 1 and the other is close to -1). The weights $\vec{\gamma}$ can be learned from the data through joint optimization in our new method. If the appearance of some annotation features always imply an within-class edge connection, then these features will get positive weights. On the other hand, if some annotation features always appear in between-class connections, then these features are more likely to get negative weights. The learned annotation weights help us to better predict the f_i values on the test examples.

We want to emphasize that there can be other methods to achieve similar effect and the particular formulation in our proposal represents an instance of a general framework to use annotation information in an effective way.

7.3 Numerical Solution

Our method is closely related to [51], where an efficient algorithm was proposed to solve (7.1). In this paper, a modification of that algorithm is used to solve (7.3). Similar to [51] (see derivations there), we rewrite (7.3) as

$$\begin{aligned} \hat{f} &= \arg\min_{f,\vec{\gamma},\vec{v},\vec{w}} \sum_{i=1}^{n} loss(f_i, y_i) + \lambda(\|\vec{w}\|^2 + \|\vec{v}\|^2) + \lambda' \sum_{(i,j)\in E} (f_i f_j - \vec{\gamma}\vec{a}_{ij})^2 \\ &\quad \text{s.t. } f_k = \vec{w}\vec{x}_k + v_k\sqrt{u} \quad (k = 1, ..., m). \end{aligned} \tag{7.4}$$

Here u plays the role as a stabilizing parameter and we set it to be a small constant 0.1. By reformulating the objective function as in (7.4), the construction of the dense kernel matrix K is avoided. We can then solve the problem iteratively using the gradient descent algorithm. In each iteration, we first fix $\vec{\gamma}$ and solve for $\vec{v}$ and $\vec{w}$; we then fix $\vec{v}$ and $\vec{w}$ and solve for $\vec{\gamma}$. This process is iterated until the algorithm converges. The initial values of these parameters are all set to be zero. The pseudo code of the optimization method is given in Algorithm 5. The detailed implementation of steps (a) and (b) will be skipped due to the limitation of space.

Algorithm 5 Pseudo code of the optimization algorithm corresponding to formula 7.4

1. Initialize the parameters $\vec{w}$, $\vec{v}$ and $\vec{\gamma}$.
2. Loop
   (a) Use gradient descent to search values of $\vec{w}$ and $\vec{v}$ with fixed $\vec{\gamma}$.
   (b) Use gradient descent to search values of $\vec{\gamma}$ with fixed $\vec{w}$ and $\vec{v}$.
3. Until convergence

7.4 Experiment

We evaluated our method on two real hyper-linked datasets: WebKB (*http://www.cs.cmu.edu/webkb/*) and Yahoo! Directory (*http://www.yahoo.com/*). The graph regularization approach in [51] was implemented as our baseline approach and was named "Baseline". We also included an extension of the baseline by putting the annotation features associated with each edge into the local features of nodes connected by this edge, and we called this extension "Baseline with edge-feature".

For the WebKB dataset, we constructed not only the original directed hyper-linked web-graph, but also the co-citation graph derived from the original directed graph. In a co-citation graph, two pages are connected by an undirected edge if both pages are linked to by a third page. Experiments are conducted using the original (which we treated as undirected) and co-citation graphs separately. For the Yahoo! Directory dataset, we only constructed the co-citation graph because its directed graph is too sparse. The co-citation graphs are often much denser and can be used to achieve better categorization performance in our experiments compared to their corresponding directed graphs.

For the directed graph, we simply use the anchor text associated with each hyperlink as the edge annotation features. For the co-citation graph, when two pages are linked by the same web-page, we use the intersection of the anchor text associated with the incoming links of these two pages (in the original directed graph) as the edge annotation features. For convenience, we still call these features "anchor text of the edge". The WebKB dataset consists of 8275 web-pages crawled from university web sites. The vocabulary consists of 20000 most frequent words. The number of edges in the original directed graph is 10721. In order to form a denser co-citation graph, we also used additionally crawled web-pages so that pages used to derive the co-citation graph do not necessarily belong to the set of WebKB pages. In this way, we get a co-citation graph with 1143716 edges. The Yahoo! Directory dataset consists of 22969 web pages. Each page of this collection belongs to one

of the 13 top level topical directory categories (for example, arts, business and education). The vocabulary consists of 50000 most frequent words. The number of edges in its co-citation graph is 1170029.

We randomly split the labeled data into two parts: 50 percent for training and another 50 percent for testing. We draw five runs and report test set averages and standard deviations. The least squares loss $loss(f, y) = (f - y)^2$ is used in our experiment. The best λ can be found through cross validation in the training data without the graph structure. We fix this λ for all other configurations (with graph structure). Our approach easily scales to the datasets used in this paper. A typical run for each dataset takes well under an hour on a standard PC. The micro-F1 and macro-F1 performance of different approaches on multiple datasets with co-citation graphs are listed in Table 7.1 and Table 7.2. The F1 performance of individual categories is shown in figure 7.1.

Table 7.1: The micro-averaged F1 performance using co-citation graphs (mean ± std-dev %)

	Yahoo Directory Data	WebKB Data
Baseline	57.9 ± 0.5	86.5 ± 0.3
Baseline with edge-feature	60.5 ± 0.4	87.6 ± 0.6
Our approach	64.8 ± 0.4	89.0 ± 0.5

Table 7.2: The macro-averaged F1 performance using co-citation graphs(mean ± std-dev %)

	Yahoo Directory Data	Webkb Data
Baseline	57.9 ± 0.4	77.2 ± 0.3
Baseline with edge-feature	60.8 ±0.4	82.6 ± 0.5
Our approach	65.1 ±0.5	82.7 ± 0.3

We can see that our approach has achieved significant improvement compared with the two baseline approaches. Note that while both our approach and the "Baseline with edge-feature" approach can exploit the anchor text information, our approach has important advantages. A key observation is that the anchor text associated with edges may not imply any specific category by itself. For example, in the co-citation graph of the Yahoo Directory data, the word "foods" has appeared in the anchor text of 132 edges. While 112 of these edges connect two web pages that both belong to the "Health" category, the other 20 edges connect two web pages that both belong to the "Science" category. Therefore, the anchor text word "foods" implies that the two connected pages have the same class labels. However, if we use the "Baseline with edge-feature" approach and put this word into the

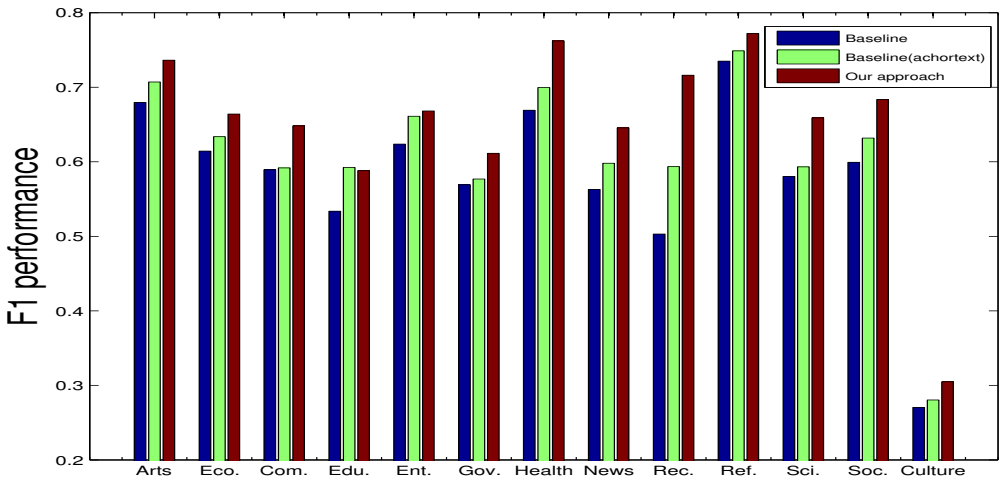

Figure 7.1: The F1 performance of three approaches for each of the 13 yahoo categories. The names of these 13 categories (from left to right) are "Arts", "Business and Economy", "Computers and Internet", "Education", "Entertainment", "Government","Health", "News and Media", "Recreation", "Reference", "Science", "Social Science" and "Society and Culture".

body text features, the appearance of this word tends to predict the "Health" category while ignoring the "Science" category. On the other hand, our approach only employs the word to determine the strength of the connected pages belonging to the same class, which is more effective.

In Table 7.3, we list for each of the 13 categories in the Yahoo! Directory data the top-five ranked words in the undirected co-citation edges that are assigned the largest weights in our approach. Notice that the meanings of the top ranked edge words in a category are not necessarily related to the meaning of that category. For example, the word "sw" is a top edge feature for the "Arts" category (and many other categories) while the meaning of this word is not related to "Arts". In fact, the word "sw" has appeared in the anchor text of 60 edges. While 58 of these edges are connecting two web pages that both belong to the "Entertainment" category, the other 2 edges are connecting two web pages that both belong to the "Computers and Internet" category. So the appearance of this word implies that the two connected pages have the same class labels, which is why "sw" is also given a high rank for "Arts" category (although it has no effect there).

We have also conducted experiments using directed graphs on the Webkb data. The micro-averaged F1 numbers of the "Baseline" approach, the "Baseline with edge-feature"

Table 7.3: Top-five ranked edge-feature words for the Yahoo! Directory with co-citation graph

Category name	Top-five ranked words in the edges
Arts	sw, halloween, b-school, artists, antonio
Business and Economy	hedge, sw, penny, fat, lib
Computers and Internet	hp, sharing, windows, programming, linux
Education	b-school, school's, gifted, exam, montessori
Entertainment	fanlisting, updated, wow, adventure, rumors
Government	base, military, b-school, representative, sw
Health	dieting, menopause, diets, vitamin, depression
News and Media	cbs, anchor, daniel, bio, cnn
Recreation	fat, b-school, sw, collections, foundation's
Reference	lib, catalog, collections, fat weight, sw
Science	horticulture, extension, soil, beef, floriculture
Social Science	sw, fat, univ, submarine, msu
Society and Culture	halloween, sw, b-school, fat, haunted

approach, and our approach are 0.850, 0.859, and 0.864 respectively. We can see that the classification performance of our approach on directed graphs is still competitive.

7.5 Summary

In this chapter, we presents a novel risk minimization formulation for classifying graph nodes that can effectively utilize the underlying graph structure with annotated edges. The main idea of our approach is that edge importance can be automatically determined from edge annotations. This is achieved by solving a joint optimization problem with an edge-annotation dependent graph-regularizer, which we introduced in this chapter. We tested the new method on web-page classification and gene-function prediction tasks, and compared its performance to two previously proposed strong base-line methods. The experiments showed that our approach outperformed both methods, confirming that the new approach can utilize edge-annotation information more effectively.

Chapter 8

Summary of Thesis Work and Contributions

8.1 Summary of Thesis Work

In this thesis, we propose a novel approach for learning large sparse undirected graph structures effectively and efficiently in a probabilistic framework. We use Graphical Gaussian model (GGM) as the underlying model and propose a novel ARD style Wishart prior for the precision matrix of the GGM, which encodes the graph structure we want to learn. With this prior, we can get the MAP estimation of the precision matrix by solving a set of (modified) Lasso regressions and achieve a sparse solution. By proposing a generalized version of Lasso regression, named Feature vector machine (FVM), we have further extended our structure learning model so that it can capture non-linear dependencies between node variables. The optimization problem in our model remains convex even in non-linear cases, which makes our solution global optimal. We have also developed a graph-based classification approach for predicting node labels given network structures, either observed or automatically induced. This approach can automatically determine edge weights based on the observed edge annotations. Compared with other graph-based semi-supervised learning approaches, which treat edge weights as pre-determined constants, our method is more flexible and achieves better performance.

We have applied our structure learning approach on microarray data for gene regulatory network re-construction, and on textual corpus for category network extraction. The extracted category network has been used to help multi-class text categorization tasks. Experimental results show that our approach is empirically effective and achieves some improvement over other state-of-art methods.

We also applied our extended graph-based classification approach on web-page classification tasks, and compared its performance to two previously proposed strong base-line methods. Experiments showed that our approach outperformed both methods, confirming that the new approach can utilize edge-annotation information more effectively.

8.2 Contributions

There are several contributions of this thesis work.

- **Theoretically and computationally**
  - We propose a novel approach for learning large sparse undirected graph structures in a probabilistic framework. Our approach transforms the problem of structure learning for Graphical Gaussian Model into the problem of solving a set of modified Lasso regressions. We can theoretically guarantee that the global optimal solution be found with a low polynomial (quadratic when the graph is sparse) computational cost.
  - We have explored the mapping between SVM regression and Lasso regression. Based on this exploration, we propose a generalization of Lasso regression named Feature vector machine, which extends our structure learning model to non-linear cases and enriches the current approaches for feature selection . Experimental results on the simulated data are very encouraging.
  - We have proposed a graph-based classification approach for predicting node labels given network structures, either observed or automatically induced. This approach can automatically determine edge weights based on the observed edge annotation. Compared with other graph-based semi-supervised learning approaches, which treat edge weights as pre-determined constants, our method is more flexible and can exploit the data in a better way.
- **Empirically**
  - We used our structure learning approach to re-construct gene regulatory networks from microarray data and other data sources and have achieved improvement over other state-of-art methods.
  - We used our structure learning approach to extract category networks from textual data and then used the learned network to help multi-class text categorization tasks. We applied our approach to the Reuter 21578 dataset and observed significant improvement compared to the baseline binary classifiers.

– We applied our extended graph-based classification approach on web-page classification tasks, and compared its performance to two previously proposed strong base-line methods. The experiments showed that our approach outperformed both methods, confirming that the new approach can utilize edge-annotation information more effectively.

Chapter 9

Future Work

The work presented in this thesis provides a basic framework for structure learning with large undirected graphs and its related applications. There are still many related research problems that I plan to pursue in future work. Some are listed as following.

- First, I want to combine the problem of node label prediction and the problem of graph structure learning into a unified framework. In many cases, each node in a graph has one or more labels and nodes with the same labels tend to be connected in the graph. Thus it's interesting to explore a unified framework which could simultaneously learn graph structures using partially known label information and predict un-observed labels using learned structures. One example of such a scenario is gene regulatory network induction from microarray data. We know co-regulated genes (which are connected nodes in the graph) are likely to share similar biological functions. Thus we can treat gene functions as node labels and use them to help learning gene network structures. The estimated structures could in turn help us to predict unknown gene functions. As far as we know, such a unified framework does not exist yet and I am planning to work on it.

- Second, I want to explore structure learning at different granularities in graph-based classification models. In this thesis, I have proposed one approach that uses learned category networks to help multi-class text categorization tasks. In fact, a category network can be thought as a kind of higher granularity generalization of document networks. We can also learn networks at other granularities, like networks defined on words or latent aspects, and use them to influence classification decisions in different ways. Are these networks all helpful for the classification task? Can we combine them to further improve the performance? How should we consider the mappings

between these networks in our structure learning process? I hope to answer these questions in the future.

- Third, I would like to try the FVM approach on real-world data. In this thesis, I have conducted experiments on simulated data and FVM achieved encouraging results. But in real-world applications, things will be more complex. How should we choose the best non-linear kernels for feature vectors? Will the introduction of slack variables in FVM really help with noisy data? I expect to address these questions in my future research.

Chapter 10

References

Bibliography

[1] M. Ashburner, CA. Ball, JA. Blake, D. Botstein, and H. Butler. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[2] Z. Bar-Joseph, GK. Gerber, T. Lee, and NJ. Rinaldi. Computational discovery of gene module and regulatory networks. *Nature Biotechnology*, 21(11):1337–42, 2003.

[3] M. Belkin and P. Niyogi. Semi-supervised learning on riemannian manifolds. *Machine Learning, Special Issue on Clustering*, pages 209–239, 2004.

[4] A. Bernard and AJ. Hartemink. Informative structure priors: Joint learning of dynamic regulatory networks from multiple types of data. In *Pacific Symposium on Biocomputing*, Hawaii, 2005.

[5] D. Blei and J. Lafferty. Correlated topic models. In *In Advances in Neural Information Processing Systems 18*, 2006.

[6] S. Canu and Y. Grandvalet. Adaptive scaling for feature selection in svms. In *NIPS*, 2002.

[7] D. Chickering. Learning equivalence classes of bayesian network structures. *JMLR*, 2:445–498, 2002.

[8] TM. Cover and JA. Thomas. *Elements in Information Theory*. New York: John Wiley & Sons Inc, New York, 1991.

[9] A. Dobra, C. Hans, B. Jones, J.R. Nevins, G. Yao, and M. West. Sparse graphical models for exploring gene expression data. *J. Mult. Analysis*, 90:196–212, 2004.

[10] MB. Eisen, PT. Spellman, PO. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Nat. Acad. Sci. USA.*, 95:14863–14868, 1998.

[11] MB. Eisen, PT. Spellman, PO. Brown, and D. Botstein. Systematic determination of genetic network architecture. *Nat Genetics*, 22:281–285, 1998.

[12] M. Figueiredo and AK. Jain. Bayesian learning of sparse classifiers. In *CVPR*, pages 35–41, 2001.

[13] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian network to analyze expression data. *Jounnal of Computational Biology*, 7(2002):175–186, 1996.

[14] N. Friedman, I. Nachman, and D. Peer. Learning bayesian network structure from massive datasets: The sparse candidate algorithm. In *UAI*, 1999.

[15] D. Geiger and D.Heckerman. Parameter priors for directed acyclic graphical models and the characterization of several probability distributions. *Annals of Statistics*, 5:1412–1440, 2002.

[16] Y. Grandvalet. Least absolute shrinkage is equivalent to quadratic penalization. *ICANN'98*, 1998.

[17] M. Gustafsson, M. Hornquist, and A. Lombardi. Large-scale reverse engineering by the lasso. In *ICSB 2003 Poster*, 2003.

[18] AJ. Hartemink, DK. Gifford, TS. Jaakkola, and RA. Young. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. In *Pacific Symposium on Biocomputing*, Hawaii, 2001.

[19] T. Hastie and R. Tibshirani. *Generalized Additive Models*. New York: Chapman and Hall, 1990.

[20] D. Heckerman. A tutorial on learning with bayesian networks. In *In Learning in Graphical Models*, Cambridge, MA, USA, 1999. MIT Press.

[21] S. Hochreiter and K. Obermayer. Gene selection for microarray data. *Kernel Methods in Computational Biology*, pages 319–355, 2004.

[22] C.F. Aliferis I. Tsamardinos, L.E. Brown. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 2004.

[23] T. Joachims. Making large-scale svm learning practical. *Advances in Kernel Methods - Support Vector Learning*, 1999.

[24] B. Krishnapuram. Joint classifier and feature optimization for cancer diagnosis using gene expression data. In *RECOMB*. ACM press, 2003.

[25] P. Larraaga, M. Poza, Y. Yurramendi, R. Murga, and C. Kuijpers. Learning of bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(9):912–926, 1996.

[26] S. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996.

[27] TI. Lee, NJ. Rinaldi, FD Robert, and DT Odom. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, 298:799–804, 2002.

[28] F. Li and Y. Yang. A loss function analysis for classification methods in text categorization. In *ICML*, 2003.

[29] F. Li and Y. Yang. An analysis of recursive feature selection elimination methods for statistical classification. In *SIGIR*, 2005.

[30] F. Li and Y. Yang. Analysis of recursive gene selection approaches from microarray data. *Bioinformatics*, 21(19):3741–3747, 2005.

[31] F. Li and Y. Yang. Using modified lasso regression to learn large undirected graphs in a probabilistic framework. In *AAAI*, 2005.

[32] F. Li, Y. Yang, and E. Xing. From lasso regression to feature vector machine. In *NIPS*, 2005.

[33] D. Margaritis. Learning bayesian network model structure from data. *Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University*, 2003.

[34] D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1999.

[35] N. Meinshausen and P. Buhlmann. Consistent neighbourhood selection for sparse high-dimensional graphs with lasso. Technical Report Seminar for statistic, ETH 123, Institute for the Study of Learning and Expertise, 2004.

[36] A. Ng. Feature selection, l1 vs l2 regularization, and rotational invariance. In *ICML*, 2003.

[37] S. James Press. *Applied Multivariate Analysis*. Holt, Rinehart and Winston, INC, New York, 1972.

[38] V. Roth. The generalized lasso. *IEEE Transactions on Neural Networks*, 15, 2004.

[39] S. Russell and P. Norvig. *Artificial Intelligence: A Mordern Approach*. Prentice-Hall, 1995.

[40] J. Schafer and K. Strimmer. An empirical bayes approach to inferring large-scale gene association networks. *Bioinformatics in press*, 2004.

[41] E. Segal, R. Yelensky, and D. Koller. Genome-wide discovery of transcriptional modules from dna sequence and gene expression. *Bioinformatics*, 19:273–82, 2003.

[42] P. Simon, L. Kevin, and T. James. Grafting: Fast,incremental feature selection by gradient descent in function space. *JMLR*, pages 1333–1356, 2003.

[43] PT. Spellman, G. Sherlock, MQ. Zhang, and VR. Iyer. Compehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. molecular. *Biology of the Cell*, 9:3273–3297, 1998.

[44] P. Spirtes, C. Glymour, and R. Scheines. *Causation,Prediction,and Search*. The MIT Press, Cambridge, Massachusetts, London, England, 2000.

[45] C.J. Stone. Additive regression and other nonparametric models. *Annals of Statistics*, 13:689–705, 1985.

[46] M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. In *NIPS 2001*, 2002.

[47] R. Tibshirani. Optimal reinsertion:regression shrinkage and selection via the lasso. *J.R.Statist*, Soc. B(1996), 58,No.1:267–288, 1996.

[48] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[49] J. Weston, S. Mukherjee, O. Chapell, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for svms. In *NIPS*, 2000.

[50] A. Wille, P. Zimmermann, E. Vranov, and A. Frholz. Sparse graphical gaussian modeling of the isoprenoid gene network in arabidopsis thaliana. *Genome Biology*, 5:92, 2004.

[51] T. Zhang, A. Popescul, and B. Dom. Linear prediction models with graph regularization for web-page categorization. In *SIGKDD 2006*, 2006.

[52] X. Zhu. *Semi-Supervised Learning with Graphs*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2005.