UNIVERSITY OF CALIFORNIA,
IRVINE

Machine Learning for Large-Scale Genomics: Algorithms, Models and Applications

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Yifei Chen

Dissertation Committee:
Professor Xiaohui Xie, Chair
Professor Pierre Baldi
Professor Amelia Regan

2014

DEDICATION

To my parents.

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vii**

LIST OF ALGORITHMS **ix**

ACKNOWLEDGMENTS **x**

CURRICULUM VITAE **xii**

ABSTRACT OF THE DISSERTATION **xiv**

1 Introduction **1**
1.1 Background 1
1.2 Dissertation Contributions and Outline 3

2 Efficient Variable Selection in Elastic Net SVM for Cancer Diagnosis **8**
2.1 Introduction 8
2.2 Support Vector Machines with Elastic Net Penalty 10
2.2.1 SVM as Regularized Function Estimation 10
2.2.2 Related Work 12
2.3 Algorithm for Elastic Net SVM 13
2.3.1 Deriving ADMM for Elastic Net SVM 13
2.3.2 Convergence Analysis 20
2.3.3 Computational Cost 20
2.4 Numerical Results 21
2.4.1 Simulation 22
2.4.2 Gene Expression Data 26
2.5 Discussion 27

3 Efficient Latent Variable Gaussian Graphical Model Selection **29**
3.1 Introduction 29
3.2 Split Bregman Method for Latent Variable Gaussian Graphical Model Selection 33
3.2.1 Derivation of the Split Bregman Algorithm for Latent Variable Gaussian Graphical Model Selection 34
3.3 Numerical Experiments 39

3.3.1 Artificial Data . . . 40
3.3.2 Gene Expression Data . . . 42
3.4 Discussion . . . 43

4 Ensemble Learning of Concordance Index for Cancer Survival Analysis 46
4.1 Introduction . . . 46
4.2 Survival Analysis, Existing Models and the New Approach . . . 49
4.2.1 Survival Analysis . . . 49
4.2.2 Gradient Boosting Machine . . . 51
4.2.3 Concordance Index Learning via Gradient Boosting . . . 53
4.3 Results . . . 55
4.3.1 Dataset and Feature Extraction . . . 55
4.3.2 Experimental Settings . . . 55
4.3.3 Empirical Comparison . . . 57
4.4 Discussion . . . 62

5 Deep Learning for Gene Expression Inference 65
5.1 Introduction . . . 65
5.2 Methods . . . 68
5.2.1 Linear Regression . . . 69
5.2.2 k-Nearest Neighbors Regression . . . 69
5.2.3 GEIDN . . . 70
5.3 Results . . . 74
5.3.1 GEO Data and L1000 Data . . . 74
5.3.2 Main Results on the Complete GEO Dataset . . . 75
5.3.3 The Effect of Data Size . . . 77
5.3.4 Inference on the L1000 Data . . . 80
5.4 Discussion . . . 81

6 Deep Learning for Genetic Variants Annotation 84
6.1 Introduction . . . 84
6.2 Methods . . . 85
6.2.1 Model Training . . . 85
6.2.2 Features . . . 86
6.2.3 Training Data . . . 86
6.3 Results and Discussion . . . 87

7 Conclusion 90
7.1 Future Directions . . . 92

Bibliography 94

A Supplementary Material for Chapter 5 103
A.1 Supplementary Table . . . 103
A.2 Supplementary Figure . . . 104

LIST OF FIGURES

Page

2.1 CPU times of ADMM-ENSVM for the same problem as in Table 2.1, for different values of n and p. In each case the times are averaged over 10 runs. (a) n is fixed and equals to 300; (b) p is fixed and equals to 2000. 24

3.1 (a) Comparison of CPU time curves w.r.t. number of variables p for the artificial data; (b) Comparison of CPU time bar charts w.r.t. number of variables p for the gene expression data 45

4.1 Predictive performance I of GBM methods on the breast cancer dataset. The box plots show the predictive concordance indices of "gbmsci" and "gbmcox" in 50 random experiments without subsampling, using the five feature representations explained in Table 4.1. In each box plot, the central red line indicates the median C-index; the blue box is the $[25\%, 75\%]$ area; the black whiskers reach the upper and lower extremes not including outliers; the red "+" symbols represent the outliers. 58

4.2 Predictive performance II of GBM methods on the breast cancer dataset. The box plots show the predictive concordance indices of "gbmsci" and "gbmcox" in 50 random experiments with subsampling ($\frac{n_s}{n} = 0.5$), using the five feature representations explained in Table 4.1. In each box plot, the central red line indicates the median C-index; the blue box is the $[25\%, 75\%]$ area; the black whiskers reach the upper and lower extremes not including outliers; the red "+" symbols represent the outliers. 59

4.3 Predictive performance of the RSF method on the breast cancer dataset. The box plots show the predictive concordance indices of "rsf" in 50 random experiments, using the five feature representations explained in Table 4.1. In each box plot, the central red line indicates the median C-index; the blue box is the $[25\%, 75\%]$ area; the black whiskers reach the upper and lower extremes not including outliers; the red "+" symbols represent the outliers. 60

5.1 The prediction error box plot over all target genes for predictive models discussed in Section 5.3.2. GEO-tr, GEO-va, and GEO-te are used for model training, validation, and testing. (a) Using RMSE as the evaluation metric; (b) using RMRSE as the evaluation metric. 76

5.2 The predictive errors of GEIDN-3 compared to LR and KNN-GE. Out of the 21,290 target genes, GEIDN-3 performs better than LR in 21,283 genes (**99.97%**), better than KNN-GE in 21,288 genes (**99.99%**), in terms of RMSE; statistics are similar in terms of RMRSE. (a) RMSEs of GEIDN-3 versus those of LR; (b) RMSEs of GEIDN-3 versus those of KNN-GE; (c) RMRSEs of GEIDN-3 versus those of LR; (d) RMRSEs of GEIDN-3 versus those of KNN-GE. 78
5.3 Learning curves of GEIDN-3. The target genes are randomly permuted and then assigned with labels 1~21,290. After that, they are evenly distributed into four neural networks for training and prediction, whose learning curves are shown by (a), (b), (c), and (d) separately. See Section 5.2.3 for the motivation of this design. 79
5.4 Inference summary statistics of LR, KNN-GE and GEIDN-3 on L1k-sub. (a)(c)(e) Minimum and maximum predictions at each target gene; (b)(d)(f) mean and standard deviation of predictions at each target gene. 82

6.1 ROC curves comparing performances of the neural network (DANN), support vector machine (SVM), and logistic regression (LR) models in discriminating (a) "simulated" variants from "observed" variants in the testing set and (b) pathogenic ClinVar variants from likely benign ESP alleles (DAF $\geq$ 5%). . . 89

LIST OF TABLES

Page

2.1 Running times (CPU seconds) of ADMM-ENSVM, the path algorithm and the stochastic sub gradient method (SSG) for HHSVM. The algorithms are run with various (p, n) pairs and two correlation ρ between the features. The results for ADMM-ENSVM and SSG are averaged over 25 runs (using 25 different values of λ_1, λ_2) and the results for HHSVM are averaged over 5 runs (using 5 different values of λ_2). 23

2.2 Comparison of test errors. The number of training samples is 50. The total number of input variables is 300, with only 10 relevant for classification. The results are averaged testing errors over 100 random repetitions, and the numbers in parentheses are the corresponding standard errors. $\rho = 0$ corresponds to the case where the input variables are independent, while $\rho = 0.8$ corresponds to a pairwise correlation of 0.8 between relevant variables. . . . 25

2.3 Comparison of variable selection. The setup is the same as that described in Table 2.2. q_{signal} is the number of selected relevant variables, and q_{noise} is the number of selected noise variables. 26

2.4 Running times (CPU seconds) for different values of the regularization parameters λ_1 and λ_2. The methods are ADMM-ENSVM and the path algorithm for HHSVM. 27

2.5 Comparison of testing error and variable selection on the gene expression data. Results are averaged over 100 repetitions and the numbers in the parenthesis are the standard deviations. 27

3.1 Numerical comparison at $p_o = 3000, p_h = 10$ for the artificial data 41

3.2 Numerical comparison at 3000-dimensional subset of the gene expression data 42

3.3 Comparison of generalization ability on the gene expression data at dimension of 1000 using latent variable Gaussian graphical model (LVGG) and sparse Gaussian graphical model (SGG) 44

4.1 The five sets of features extracted from the Metabric breast cancer dataset. . 56

4.2 Numerical statistics of predictive concordance indices of GBM models and the Cox model on the breast cancer dataset. The five feature representations are explained in Table 4.1. "gbmsci"-I and "gbmcox"-I run without subsampling ($\frac{n_s}{n} = 1$), while "gbmsci"-II and "gbmcox"-II run with subsampling ($\frac{n_s}{n} = 0.5$). The numerical values in each entry show the average C-index and the standard deviation (following $\pm$) over 50 random runs. The bold font highlights the best performance in each column. 61

5.1 The average prediction error over all target genes for predictive models discussed in Section 5.3.2. GEO-tr, GEO-va, and GEO-te are used for model training, validation, and testing. Numerical values after "$\pm$" are the standard deviations. The bold font highlights the best performances. 75

5.2 The average prediction error over all target genes for LR, LR-L1, KNN-GE and GEIDN-1. GEO-tr-sub, GEO-va-sub, and GEO-te are used for model training, validation, and testing. Numerical values after "$\pm$" are the standard deviations. The bold font highlights the best performances. 80

LIST OF ALGORITHMS

		Page
1	Alternating Direction Method of Multipliers for Solving the Elastic Net SVM (ADMM-ENSVM)	19
2	Split Bregman Method for Solving the Latent Variable Gaussian Graphical Model (SBLVGG)	39
3	Gradient Boosting Machine for Concordance Index Learning (GBMCI) . . .	55

ACKNOWLEDGMENTS

I owe my greatest gratitude to my research advisor, Prof. Xiaohui Xie, for his consistent support and guidance throughout my entire doctorate program. Xiaohui leads me into the field of machine learning and computational biology, carefully shapes my scientific and critical thinking ability, and always reminds me to think about the big picture and direction. He teaches me how to formulate a complicate biological problem as a concretized mathematical model, how to further reformulate or decompose it until tractable and solvable, and how to conduct detailed experiments rigorously and efficiently. His optimism, passion and trust, in particular during the difficult period, have been an essential factor to encourage me to go through this tough journey. I should also express my gratitude to my parents for their persistent support and encouragement over the decades. Without any of them, I would not have survived the most difficult time, progressed this far along my PhD program, and finished this dissertation.

I would like to thank other members of my dissertation committee: Prof. Pierre Baldi has given several excellent seminars about deep learning and scientific writing, the ideas from which I acquired have been important motivations to my recent research projects; Prof. Amelia Regan has offered an algorithm course and has attended my advancement exam committee, where I learnt a lot about the principles of computer science research. I would like to thank Prof. Max Welling and Prof. Alex Ihler. They have offered a series of machine learning courses, which helps me to set up my basic training skills in this field. They also provide many insightful discussions and suggestions during my research projects. I also would like to thank Prof. Ruqian Wu, who has shared many interesting ideas in computational physics, and has always encouraged me to be tough and optimistic.

I owe great thankfulness to my lab mates and colleagues. Jacob Biesinger popularizes python and bash scripting throughout the whole lab. Guibo Ye specializes in applied mathematics and has published a series of excellent papers on large-scale optimization. Yuanfeng Wang deploys his unique physics background to tackle systems biology problems. Yi Li often comes up with great ideas independently, and can push them towards publication so fast. Mengfan Tang does preliminary explorations for the CMap project and helps me a lot to take it. Daniel Newkirk always generously shares his high-end computer hardware with his colleagues. Daniel Quang is a genius scientific writer, and often solves problem amazingly fast. Peter Sadowski is a deep learning expert and helps me a lot to grasp Theano/Pylearn2. Lingjie Weng and Elmira Forouzmand both are active computational biologists with solid research background. I feel so fortunate to have had close interactions with all of them.

I would like to acknowledge UCI's ICS fellowship, and a few NSF & NIH grants of CBCL@UCI, which alternately supported me throughout my doctoral program. I would like to acknowledge my coauthors who have attended the research projects presented in this dissertation: Guibo Ye, Yuanfeng Wang, Zhengyu Jia, Dan Mercola, Rajiv Narayan, Aravind Subramanian, Daniel Quang and Xiaohui Xie. I have learnt a lot from my collaborators.

The researches presented in Chapters 2 and 3 are supported by a grant from University of

California, Irvine. I greatly acknowledge Prof. Jian-Feng Cai for helpful discussions. The research presented in Chapter 4 is partially supported by grants NSF DBI-0846218, NCI UO1CA11480 (SPECS UO1), NCI UO1CA152738 (EDRN), UO11CA162147, UCI Chao Family Comprehensive Cancer Center, NCI P30CA62203 and by the U.S. Department of Defense Congressionally Mandated Research Program grant PC120465. I greatly acknowledge Dr. Wei-Yi Cheng for sharing the Attractor Metagene algorithm, and Daniel Newkirk and Jacob Biesinger for helpful discussions. The Metabric dataset was accessed through Synapse (`https://synapse.sagebase.org/`). The research presented in Chapter 5 is supported by a grant from University of California, Irvine. I greatly acknowledge Peter Sadowski, Daniel Quang, Mengfan Tang, Yi Li, Ian Goodfellow, Frédéric Bastien, Kyle Kastner and Olivier Delalleau for helpful discussions. The GEO dataset and the L1000 dataset are both originally accessed through the LINCS Cloud (`http://www.lincscloud.org`). The research presented in Chapter 6 is partially supported by National Institute of Biomedical Imaging and Bioengineering, and National Research Service Award (EB009418) from Center for Complex Biological Systems, University of California Irvine. I gratefully acknowledge Oxford University Press for giving me permission to incorporate our publication into my dissertation, and Martin Kircher for helping us to understand CADD and providing the datasets.

CURRICULUM VITAE

Yifei Chen

EDUCATION

Doctor of Philosophy in Computer Science **2014**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science **2013**
University of California, Irvine *Irvine, California*

Bachelor of Engineering in Automation **2007**
Tongji University *Shanghai*

RESEARCH EXPERIENCE

Graduate Research Assistant **2010 – 2014**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

ICS 33 & CSE 43: Intermediate Programming Fall 2014
ICS 33 & CSE 43: Intermediate Programming Winter 2014
ICS 32 & CSE 42: Programming with Software Libraries Fall 2013
CS 174: Bioinformatics Spring 2011
CS 178: Machine Learning and Data Mining Winter 2011
CS 164 & CS 266: Computational Geometry Spring 2010
CS 171: Introduction to Artificial Intelligence Winter 2010
CS 171: Introduction to Artificial Intelligence Fall 2009
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATION

GEIDN: a deep learning approach for gene expression inference **2014**
Y. Chen, A. Subramanian, R. Narayan, and X. Xie. Manuscript in preparation

DANN: a deep learning approach for annotating the pathogenicity of genetic variants **2014**
D. Quang[†], Y. Chen[†], and X. Xie. Bioinformatics ([†]: joint First Authors)

A gradient boosting algorithm for survival analysis via direct optimization of concordance index **2013**
Y. Chen, Z. Jia, D. Mercola, and X. Xie. Computational and Mathematical Methods in Medicine

REFEREED CONFERENCE PUBLICATION

Efficient variable selection in support vector machines via the alternating direction method of multipliers **2011**
G. Ye, Y. Chen, and X. Xie. International Conference on Artificial Intelligence and Statistics

TECHNICAL REPORT

Inference in Kingman's Coalescent with Particle Markov Chain Monte Carlo method **2013**
Y. Chen, and X. Xie. arXiv

Efficient latent variable graphical model selection via Split Bregman method **2011**
G. Ye, Y. Wang, Y. Chen, and X. Xie. arXiv

SOFTWARE

GBMCI `https://github.com/uci-cbcl/GBMCI`
A C++/R implementation of the gradient boosting algorithm that learns Harrell's concordance index.

ABSTRACT OF THE DISSERTATION

Machine Learning for Large-Scale Genomics: Algorithms, Models and Applications

By

Yifei Chen

Doctor of Philosophy in Computer Science

University of California, Irvine, 2014

Professor Xiaohui Xie, Chair

Genomic malformations are believed to be the driving factors of many diseases. Therefore, understanding the intrinsic mechanisms underlying the genome and informing clinical practices have become two important missions of large-scale genomic research. Recently, high-throughput molecular data have provided abundant information about the whole genome, and have popularized computational tools in genomics. However, traditional machine learning methodologies often suffer from strong limitations when dealing with high-throughput genomic data, because the latter are usually very high dimensional, highly heterogeneous, and can show complicated nonlinear effects. In this thesis, we present five new algorithms or models to address these challenges, each of which is applied to a specific genomic problem.

Project 1 focuses on model selection in cancer diagnosis. We develop an efficient algorithm (ADMM-ENSVM) for the Elastic Net Support Vector Machine, which achieves simultaneous variable selection and max-margin classification. On a colon cancer diagnosis dataset, ADMM-ENSVM shows advantages over other SVM algorithms in terms of diagnostic accuracy, feature selection ability, and computational efficiency.

Project 2 focuses on model selection in gene correlation analysis. We develop an efficient algorithm (SBLVGG) using the similar methodology as of ADMM-ENSVM for the Latent Variable Gaussian Graphical Model (LVGG). LVGG models the marginal concentration ma-

trix of observed variables as a combination of a sparse matrix and a low rank one. Evaluated on a microarray dataset containing 6,316 genes, SBLVGG is notably faster than the state-of-the-art LVGG solver, and shows that most of the correlation among genes can be effectively explained by only tens of latent factors.

Project 3 focuses on ensemble learning in cancer survival analysis. We develop a gradient boosting model (GBMCI), which does not explicitly assume particular forms of hazard functions, but trains an ensemble of regression trees to approximately optimize the concordance index. We benchmark the performance of GBMCI against several popular survival models on a large-scale breast cancer prognosis dataset. GBMCI consistently outperforms other methods based on a number of feature representations, which are heterogeneous and contain missing values.

Project 4 focuses on deep learning in gene expression inference (GEIDN). GEIDN is a large-scale neural network, which can infer ˜21k target genes jointly from ˜1k landmark genes and can naturally capture hierarchical nonlinear interactions among genes. We deploy deep learning techniques (drop out, momentum training, GPU computing, etc.) to train GEIDN. On a dataset of ˜129k complete human transcriptomes, GEIDN outperforms both k-nearest neighbor regression and linear regression in predicting $> \mathbf{99.96}\%$ of the target genes. Moreover, increased network scales help to improve GEIDN, while increased training data benefits GEIDN more than other methods.

Project 5 focuses on deep learning in annotating coding and noncoding genetic variants (DANN). DANN is a neural network to differentiate evolutionarily derived alleles from simulated ones with 949 highly heterogeneous features. It can capture nonlinear relationships among features. We train DANN with deep learning techniques like for GEIDN. DANN achieves a **18.90%** relative reduction in the error rate and a **14.52%** relative increase in the area under the curve over CADD, a state-of-the-art algorithm to annotate genetic variants based on the linear SVM.

Chapter 1

Introduction

1.1 Background

Genomics has been of interest to biologists and doctors for decades. In the macroscopic paradigm, genomic malformations are believed to be the driving factors of many diseases, including different cancer types. This raises the question of how to diagnose, or to evaluate the risk of a disease, by reviewing individuals' genomic and transcriptomic profiles. A vivid application of this idea is cancer diagnosis and prognosis [2, 90, 26, 72]. However, disease evaluation is traditionally considered a responsibility of physicians, who examine patients' clinical conditions (ages, lab results, family history, etc.) and make assessments accordingly. This often requires long-term domain training and experience, but can still be subjective. Moreover, it remains challenging to incorporate large amounts of genomic information into decision-making processes. In the microscopic paradigm, genomic and transcriptomic activities have complicated patterns. For example, genes' expressions may be up-regulated or down-regulated, depending on different cellular conditions; DNA sequence base pairs may mutate (e.g., Single Nucleotide Polymorphism), delete, or duplicate, which can present vari-

ous alleles to a certain gene. Moreover, genes have intrinsic interactions: they regulate (promotes or inhibits) the activities of one another, by forming structured regulatory networks or pathways. Genomic and transcriptomic activities manifest cascading nonlinear effects, and fundamentally control the living process of any organism. Any abnormal change in the genome or the transcriptome is a potential factor to cause diseases [26, 63, 41]. Therefore, it is fundamentally important to explore the genomic process itself.

Genomic studies have significantly progressed in the recent decade, thanks to the advances of high-throughput molecular data, such as microarray gene expression data [79, 66, 65], and next-generation sequencing data [71, 92, 77]. Such datasets usually contain hundreds to tens of millions of individual samples, each of which is characterized by thousands or even millions of molecular signatures. They have provided abundant information about the whole genome in the molecular level. Therefore, deploying computational tools to study genomics from these high-throughput data has become an important interdisciplinary effort between molecular biology and machine learning. Representative research directions include gene regulatory networks [60, 52], genome-wide association studies (GWAS) and pathway analysis [14, 94, 115, 55, 27, 104]. Results of these kinds of research are further applied to many important biomedical applications, such as disease diagnosis and prognosis, drug design, etc.

However, traditional computational learning methodologies often cannot be directly applied to high-throughput molecular datasets and large-scale genomic problems, because of the following practical challenges:

- High-throughput molecular data are often of very high dimensions. For example, a whole-genome microarray assay typically measures expression levels of thousands to tens of thousands of genes. Moreover, the data are usually embedded in a low-dimensional manifold. Consequently, feature extraction or dimensionality reduction

has become a critical condition for the success of any computational learning algorithms, in particular, in the scenario in which only limited data are available.

- Heterogeneity commonly exists among available data resources. Specifically, gene expression data are continuous; next-generation sequencing data are categorical (i.e., A, C, G, T); clinical features and genetic variant features are often a mixture of continuous, Boolean, categorical and other types. Moreover, different samples may have different missing features or even different feature sets. Therefore, combining these highly heterogeneous data to achieve improved predictive power is an unavoidable problem.

- Shallow learning machines, such as linear regression and support vector machine, are still the prevalent approaches. However, the size of genomic data is increasing rapidly. For example, the colon cancer dataset collected in 1999 only has 62 samples of 2k genes [2], while the Gene Expression Omnibus dataset compiled in 2013 has ˜129k samples of ˜23k genes. Traditional methods often cannot scale up well, or cannot capture the complicated nonlinear structure within the data. Predictive models that have strong representative power but remain computationally efficient are in high demand in the "big data" era.

1.2 Dissertation Contributions and Outline

In this thesis, we will develop a series of computational learning models and algorithms to tackle high-throughput molecular data. The proposed methods have five concrete applications in large-scale genomics, as highlighted below:

1. cancer diagnosis;

2. cancer prognosis;

3. gene regulatory network interpretation;

4. gene expression inference;

5. annotating pathogenicity of genetic variants.

Applications 1 and 2 belong to the macroscopic paradigm of genomics, as they both aim to solve specific clinical problems; Applications 3, 4 and 5 belong to the microscopic paradigm of genomics, as they aim to study the intrinsic activities of the genome. Along this line of thinking, we will address the three computational challenges mentioned in the previous section from different perspectives.

The themes of the following chapters are outlined as follows:

In Chapter 2, we will present an efficient algorithm, ADMM-ENSVM, to solve the Elastic Net Support Vector Machine. ENSVM incorporates both ℓ_1- and ℓ_2-norm regularization into the standard SVM, and can achieve simultaneous variable selection and margin-maximization. However, because the loss function and the regularization term are both nondifferentiable, there is no efficient method available to solve ENSVM for large-scale problems. Our proposed algorithm is based on the Alternating Direction Method of Multipliers. By introducing auxiliary variables, it decomposes the non-smooth optimization of ENSVM into smaller easier sub-problems. We apply ADMM-ENSVM, SVM, ℓ_1-norm SVM and Hybrid Huberized SVM (HHSVM, a smoothed approximation of ENSVM solved by a path algorithm) on a colon cancer dataset with 2,000 microarray probes of 62 samples, and illustrate the advantage of ADMM-ENSVM in terms of cancer diagnose (classification) accuracy, feature selection ability, and computational efficiency over other approaches. This chapter is a revision of the original publication [111].

In Chapter 3, we will present an efficient first-order algorithm, SBLVGG, based on the Split Bregman method to solve the Latent Variable Gaussian Graphical Model. We consider

the problem of covariance matrix estimation in the presence of latent variables. Under proper conditions, it is possible to learn the marginal covariance matrix of the observed variables via a tractable convex program, where the concentration matrix of the observed variables is decomposed into a sparse matrix (representing the graphical structure of the observed variables) and a low rank matrix (representing the marginalization effect of latent variables). Empirically, SBLVGG converges quickly under mild conditions. We show that SBLVGG is notably faster than the state-of-the-art algorithm on both artificial and real-world data. Applying the algorithm to a gene expression dataset involving thousands of genes, we show that most of the correlations between genes can be explained by only a few dozen latent factors. This finding provides a new interpretation to the underlying structure of gene regulatory networks. This chapter is a revision of the original publication [112].

In Chapter 4, we will present an ensemble algorithm, GBMCI, for survival analysis via directly optimizing the concordance index. Survival analysis focuses on modeling the time to an event of interest, such as death or disease recurrence, while the concordance index is a widely adopted evaluation metric for survival analysis. Current statistical models for survival analysis often impose strong assumptions on hazard functions, which describe how the risk of an event changes depending on each individual's covariates. In particular, the prevalent proportional hazards model assumes that covariates are multiplicatively related to the hazard. Our nonparametric model GBMCI does not explicitly assume particular forms of hazard functions, but utilizes an ensemble of regression trees to model it. We develop a gradient boosting algorithm to train the tree ensemble by optimizing a smoothed approximation of the concordance index. We benchmark the performance of GBMCI against several popular survival models with a large-scale breast cancer prognosis task (˜50k microarray probes and ˜30 clinical variables of ˜2k patients). Our experiments demonstrate that GBMCI consistently outperforms other methods based on a number of feature representations, which are heterogeneous and contain missing values. This chapter has been originally published as part of [19].

In Chapter 5, we will present GEIDN, a large-scale neural network, to do gene expression inference. Although genome-wide microarrays can measure the expression of all ~22k human genes, they are prohibitively expensive. However, genes' expressions have strong correlation. The Library of Integrated Cellular Signatures (LINCS) project has identified ~1k landmark genes that capture most of the information of the other ~21k target genes, and has developed the L1000 microarray, which is optimized for measuring the expression of the landmark genes. LINCS currently uses linear regression to infer target genes' expressions separately, which potentially conflicts with the fact that genes have intrinsic nonlinear interactions, and that transcriptional programs are often grouped into modules. On the other hand, GEIDN can infer target genes jointly from landmark genes, and can naturally capture hierarchical nonlinear features shared among target genes. We deploy advanced deep learning techniques to train GEIDN, such as drop out, momentum training, and GPU computing. On a Gene Expression Omnibus dataset of ~129k samples, GEIDN improves **~41%** over k-nearest neighbor regression and **~17%** over linear regression in average prediction accuracy; GEIDN also outperforms the latter two methods for $> \mathbf{99.96}\%$ of the target genes. Moreover, increased network scales help to improve GEIDN, while increased training data benefits GEIDN more than other methods. Lastly, we infer target genes of the L1000 data with GEIDN, and have made the data publicly available for further study.

In Chapter 6[1], we will present DANN, a deep learning approach for annotating the pathogenicity of genetic variants. Annotating genetic variants, especially noncoding variants, for the purpose of identifying pathogenicity remains a challenging problem. CADD is an algorithm designed to annotate both coding and noncoding variants, and has been shown to outperform other annotation algorithms. CADD trains a linear SVM to differentiate evolutionarily derived (likely benign) alleles from simulated (likely deleterious) variants, with 949 highly

[1]This is a pre-copy-editing, author-produced PDF of an article accepted for publication in Bioinformatics following peer review. The definitive publisher-authenticated version "D. Quang, Y. Chen, and X. Xie. DANN: a deep learning approach for annotating the pathogenicity of genetic variants. Bioinformatics, 2014." is available online at: `http://bioinformatics.oxfordjournals.org/content/early/2014/11/19/bioinformatics.btu703.full.pdf+html`.

heterogeneous features. However, linear SVM cannot capture nonlinear relationships among the features, which can limit the performance. On the other hand, DANN uses the same feature set and training data as CADD, but trains a deep neural network. DANN can capture nonlinear relationships among features and are better suited than SVMs for problems with a large number of samples and features. As in Chapter 5, we exploit deep learning techniques to train DANN. DANN achieves a **18.90%** relative reduction in the error rate and a **14.52%** relative increase in the area under the curve (AUC) over CADD's SVM methodology.

In Chapter 7, we will make conclusions and discuss potential future directions.

Chapter 2

Efficient Variable Selection in Elastic Net SVM for Cancer Diagnosis

2.1 Introduction

Datasets with tens of thousands of variables have become increasingly common in many real-world applications. For example, in the biomedical domain a microarray dataset typically contains about 20,000 genes, while a genotype dataset commonly includes half of a million SNPs. Regularization terms that encourage sparsity in coefficients have become very popular for simultaneous variable selection and prediction [99, 117].

A widely adopted strategy for imposing sparsity on regression or classification coefficients is the ℓ_1-norm regularization. Perhaps the most well-known example is the least absolute shrinkage and selection operator (lasso) method for linear regression. The method minimizes the usual sum of squared errors while penalizing the ℓ_1-norm of the regression coefficients [99]. Due to the nondifferentiability of the ℓ_1-norm, the lasso is able to perform continuous shrinkage and automatic variable selection simultaneously. Although the lasso method has

shown successes in many situations and has been generalized for different settings [116, 69], it has several limitations. First, when the dimension of the data (p) is larger than the number of training samples (n), the lasso selects at most n variable before it saturates [32]. Second, if there is a group of variables whose pairwise correlations are very high, the lasso tends to select only one variable from the group and does not care which one is selected.

The elastic net penalty proposed by Zou et al. in [117] is a convex combination of the ℓ_1-norm and the ridge penalty, which has the characteristics of both the lasso and ridge regression in the regression setting. More specifically, the elastic net penalty simultaneously does automatic variable selection and continuous shrinkage, and can select groups of correlated variables. It is especially useful for "large p, small n" problems, where the "grouped variables" situation is a particularly important concern and has been addressed many times in the literature [51, 50].

The idea of using ℓ_1-norm constraints to automatically select variables has also been extended to classification problems. Zhu et al. [116] proposed an ℓ_1-norm support vector machine, whereas Wang et al. [105] proposed a SVM with the elastic net penalty term, which they named doubly regularized support vector machine (or Elastic Net SVM, abbr., ENSVM). Using a mixture of the ℓ_1-norm and the ridge penalties, ENSVM is able to perform automatic variable selection as the ℓ_1-norm SVM. Additionally, it also encourages highly correlated variables to be selected (or removed) together, and thus achieves the grouping effect.

Although ENSVM has a number of desirable features, solving ENSVM is non-trivial because of the nondifferentiability of both the loss function and the regularization term. This is especially problematic for large-scale problems. To circumvent this difficulty, Wang et al. [106] proposed the Hybrid Huberized Support Vector Machine (HHSVM), which uses a huberized hinge loss function to approximate the hinge loss in ENSVM. Because the huberized hinge loss function is differentiable, HHSVM is easier to solve than ENSVM. Wang et al. proposed a path algorithm to solve HHSVM [106]. However, because the path algorithm requires

tracking disappearance of variables along a regularization path, it is not easy to implement and still does not handle large-scale data well.

Our main contribution in this chapter is to introduce a new algorithm to directly solve ENSVM without resorting to the approximation as in HHSVM. Our method is based on the Alternating Direction Method of Multipliers (ADMM) [42, 44]. We demonstrate that the method is efficient even for large-scale problems with tens of thousands variables.

The rest of the chapter is organized as follows. In Section 2.2, we provide a description of the SVM model with the elastic net penalty. In Section 2.3, we derive an iterative algorithm based on ADMM to solve the optimization problem in ENSVM and prove its convergence property. In Section 2.4, we benchmark the performance of the algorithm on both simulated and real-world data.

2.2 Support Vector Machines with Elastic Net Penalty

2.2.1 SVM as Regularized Function Estimation

Consider the classification of the training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^T$ are the predictor variables and $y_i \in \{-1, 1\}$ is the corresponding class label. The support vector machine (SVM) was originally proposed to find the optimal hyperplane that separates the two classes of data points with the largest margin [101]. It can be equivalently reformulated as an ℓ_2-norm penalized optimization problem:

$$\min_{\beta_0,\beta} \frac{1}{n}\sum_{i=1}^{n}(1 - y_i(\beta_0 + \mathbf{x}_i^T\beta))_+ + \frac{\lambda}{2}\|\beta\|_2^2, \tag{2.1}$$

where the loss function $(1 - \cdot)_+ := \max(1 - \cdot, 0)$ is called the *hinge loss*, and $\lambda \geq 0$ is a regularization parameter, which controls the balance between the "loss" and the "penalty".

By shrinking the magnitude of the coefficients, the ridge penalty in (2.1) reduces the variance of the estimated coefficients, and thus can achieve better prediction accuracy. However, the ridge penalty cannot produce sparse coefficients and hence cannot automatically perform variable selection. This is a major limitation for applying SVM to do classification in high-dimensional datasets, such as gene expression data from microarrays [47, 78], where variable selection is essential for both achieving better prediction accuracy and providing reasonable interpretations.

To include variable selection, Zhu et al. [116] proposed an ℓ_1-norm support vector machine,

$$\min_{\beta_0,\beta} \frac{1}{n}\sum_{i=1}^{n}(1-y_i(\beta_0+\mathbf{x}_i^T\beta))_+ + \lambda\|\beta\|_1, \tag{2.2}$$

which does variable selection automatically via the ℓ_1-norm penalty. However, it shares similar disadvantages with the lasso method for "large p, small n" problems, such as selecting at most n relevant variables, and neglecting group effects. This is not satisfactory for some applications. For example, in traditional microarray analysis, we often have $p \gg n$. Furthermore, genes in the same biological pathway usually show highly correlated expressions. Therefore, it is desirable to identify all, instead a subset, of them for the purposes of both providing biological interpretations and building prediction models.

One natural thought to overcome the limitations outlined above is to apply the elastic net penalty to SVM:

$$\min_{\beta_0,\beta} \frac{1}{n}\sum_{i=1}^{n}(1-y_i(\beta_0+\mathbf{x}_i^T\beta))_+ + \lambda_1\|\beta\|_1 + \frac{\lambda_2}{2}\|\beta\|_2^2, \tag{2.3}$$

where $\lambda_1, \lambda_2 \geq 0$ are regularization parameters. The model was originally proposed by Wang et al.[105], and was named doubly regularized SVM. However, to emphasize the role of the elastic net penalty, we refer to (2.3) as the Elastic Net SVM or simply ENSVM in the remaining of the thesis. Due to the properties of the elastic net penalty, the optimal solution

of (2.3) will preserve both the sparsity and the grouping effects, which has been achieved in the regression setting.

2.2.2 Related Work

A similar model has been proposed by Wang et al. [106] who applied the huberized hinge loss function to ENSVM and proposed the Hybrid Huberized SVM (HHSVM):

$$\min_{\beta_0,\beta} \frac{1}{n}\sum_{i=1}^{n} \phi(y_i(\beta_0 + \mathbf{x}_i^T\beta)) + \lambda_1\|\beta\|_1 + \frac{\lambda_2}{2}\|\beta\|_2^2, \tag{2.4}$$

where ϕ is the huberized hinge loss function:

$$\phi(t) = \begin{cases} 0, & \text{for } t > 1, \\ (1-t)^2/2\delta, & \text{for } 1-\delta < t \leq 1, \\ 1 - t - \delta/2, & \text{for } t \leq 1-\delta, \end{cases} \tag{2.5}$$

with $\delta > 0$ being a pre-specified constant. The main motivation for [106] to use the huberized hinge loss function (2.5) is that it is an approximation of the hinge loss and differentiable everywhere, thereby making the optimization problem easier to solve while at the same time preserving the variable selection feature.

The minimizer of (2.4) is piecewise linear with respect to λ_1 for a fixed λ_2. Based on this observation, [106] proposed a path algorithm to solve HHSVM. The path algorithm keeps track of four sets as λ_1 decreases, and calls an "event" happens if any one of the four sets changes. Between any two consecutive "events", the solutions are linear in λ_1, and after an "event" occurs, the derivative of the solution with respect to λ_1 changes. When each "event" happens, the algorithm solves a linear system. If the dimension of the data p is large, it will be unavoidable to solve many large-scale linear systems so as to obtain the solution path. Furthermore, those linear equations are quite different from each other, and there are no

special structures involved. As a result, the path algorithm is computational very expensive for "large p" problems.

2.3 Algorithm for Elastic Net SVM

The Alternating Direction Method of Multipliers developed in the 1970s [42, 44] has recently become a method of choice for solving many large-scale problems [16, 15, 45]. It is equivalent or closely related to many other algorithms, such as the Douglas-Rachford splitting [107], the Split Bregman method [45] and the Method of Multipliers [87].

In this section, we propose an efficient algorithm based on ADMM to solve ENSVM in (2.3) by introducing auxiliary variables and reformulating the original problem.

2.3.1 Deriving ADMM for Elastic Net SVM

Because of the two nondifferentiable terms in (2.3), it is difficult to solve ENSVM directly. In order to derive an ADMM algorithm, we introduce some auxiliary variables to handle the nondifferentiability of the hinge loss term and the ℓ_1-norm term.

Let $X = (x_{ij})_{i=1,j=1}^{n,p}$, and Y be a diagonal matrix whose diagonal elements form the vector $y = (y_1, \ldots, y_n)^T$. The unconstrained problem of (2.3) can be reformulated as an equality-constraint problem

$$\begin{aligned} \arg\min_{\beta,\beta_0} \quad & \frac{1}{n}\sum_{i=1}^{n}(a_i)_+ + \lambda_1\|\mathbf{c}\|_1 + \frac{\lambda_2}{2}\|\beta\|_2^2 \\ s.t. \quad & \mathbf{a} = \mathbf{1} - Y\left(X\beta + \beta_0\mathbf{1}\right), \\ & \mathbf{c} = \beta, \end{aligned} \tag{2.6}$$

where $\mathbf{a} = (a_i, \ldots, a_n)^T$ and $\mathbf{1}$ is an n-column vector of 1s. Note that the Lagrangian function of (2.6) is

$$\begin{aligned} & L\left(\beta, \beta_0, \mathbf{a}, \mathbf{c}, \mathbf{u}, \mathbf{v}\right) \\ =& \frac{1}{n}\sum_{i=1}^{n}(a_i)_+ + \lambda_1\|\mathbf{c}\|_1 + \frac{\lambda_2}{2}\|\beta\|_2^2 + \langle \mathbf{u}, \mathbf{1} - Y(X\beta + \beta_0\mathbf{1}) - \mathbf{a}\rangle + \langle \mathbf{v}, \beta - \mathbf{c}\rangle, \end{aligned} \tag{2.7}$$

where $\mathbf{u} \in \mathbb{R}^n$ is a dual variable corresponding to the linear constraint $\mathbf{a} = \mathbf{1} - Y\left(X\beta + \beta_0\mathbf{1}\right)$, $\mathbf{v} \in \mathbb{R}^p$ is a dual variable corresponding to the linear constraint $\mathbf{c} = \beta$, and $\langle \cdot, \cdot \rangle$ denotes the standard inner product in Euclidean space. The augmented Lagrangian function (2.8) is similar to (2.7) except for adding two terms $\frac{u_1}{2}\|\mathbf{1} - Y\left(X\beta + \beta_0\mathbf{1}\right) - \mathbf{a}\|_2^2$ and $\frac{u_2}{2}\|\beta - \mathbf{c}\|_2^2$ to penalize the violation of linear constraints $\mathbf{a} = \mathbf{1} - Y\left(X\beta + \beta_0\mathbf{1}\right)$ and $\mathbf{c} = \beta$, thereby making the function strictly convex. That is,

$$\begin{aligned} & \mathcal{L}\left(\beta, \beta_0, \mathbf{a}, \mathbf{c}, \mathbf{u}, \mathbf{v}\right) \\ =& L\left(\beta, \beta_0, \mathbf{a}, \mathbf{c}, \mathbf{u}, \mathbf{v}\right) + \frac{\mu_1}{2}\|\mathbf{1} - Y(X\beta + \beta_0\mathbf{1}) - \mathbf{a}\|_2^2 + \frac{\mu_2}{2}\|\beta - \mathbf{c}\|_2^2, \end{aligned} \tag{2.8}$$

where $\mu_1 > 0$ and $\mu_2 > 0$ are two parameters. It is easy to see that solving (2.6) is equivalent to finding a saddle point $(\beta^*, \beta_0^*, \mathbf{a}^*, \mathbf{c}^*, \mathbf{u}^*, \mathbf{v}^*)$ of $\mathcal{L}\left(\beta, \beta_0, \mathbf{a}, \mathbf{c}, \mathbf{u}, \mathbf{v}\right)$ such that

$$\mathcal{L}\left(\beta^*, \beta_0^*, \mathbf{a}^*, \mathbf{c}^*, \mathbf{u}, \mathbf{v}\right) \leq \mathcal{L}\left(\beta^*, \beta_0^*, \mathbf{a}^*, \mathbf{c}^*, \mathbf{u}^*, \mathbf{v}^*\right) \leq \mathcal{L}\left(\beta, \beta_0, \mathbf{a}, \mathbf{c}, \mathbf{u}^*, \mathbf{v}^*\right),$$

for all $\beta, \beta_0, \mathbf{a}, \mathbf{c}, \mathbf{u}$ and $\mathbf{v}$.

We solve the saddle point problem via gradient ascent on the dual problem

$$\max_{\mathbf{u},\mathbf{v}} E(\mathbf{u}, \mathbf{v}), \tag{2.9}$$

where $E(\mathbf{u},\mathbf{v}) = \min_{\beta,\beta_0,\mathbf{a},\mathbf{c}} \mathcal{L}(\beta,\beta_0,\mathbf{a},\mathbf{c},\mathbf{u},\mathbf{v})$. Note that the gradient $\nabla E(\mathbf{u},\mathbf{v})$ can be calculated by the following [12]

$$\nabla E(\mathbf{u},\mathbf{v}) = \begin{pmatrix} \mathbf{1} - Y(X\beta(\mathbf{u},\mathbf{v}) + \beta_0(\mathbf{u},\mathbf{v})\mathbf{1}) - \mathbf{a}(\mathbf{u},\mathbf{v}) \\ \beta(\mathbf{u},\mathbf{v}) - \mathbf{c}(\mathbf{u},\mathbf{v}) \end{pmatrix}, \tag{2.10}$$

with

$$(\beta(\mathbf{u},\mathbf{v}),\beta_0(\mathbf{u},\mathbf{v}),\mathbf{a}(\mathbf{u},\mathbf{v}),\mathbf{c}(\mathbf{u},\mathbf{v})) = \arg\min_{\beta,\beta_0,\mathbf{a},\mathbf{c}} \mathcal{L}(\beta,\beta_0,\mathbf{a},\mathbf{c},\mathbf{u},\mathbf{v}). \tag{2.11}$$

Using gradient ascent on the dual problem (2.9), Eq. (2.10) and Eq. (2.11), we get the Method of Multipliers [87] to solve (2.6)

$$\begin{cases} (\beta^{k+1},\beta_0^{k+1},\mathbf{a}^{k+1},\mathbf{c}^{k+1}) = \arg\min_{\beta,\beta_0,\mathbf{a},\mathbf{c}} \mathcal{L}\left(\beta,\beta_0,\mathbf{a},\mathbf{c},\mathbf{u}^k,\mathbf{v}^k\right) \\ \mathbf{u}^{k+1} = \mathbf{u}^k + \mu_1(\mathbf{1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}) - \mathbf{a}^{k+1}), \\ \mathbf{v}^{k+1} = \mathbf{v}^k + \mu_2(\beta^{k+1} - \mathbf{c}^{k+1}). \end{cases} \tag{2.12}$$

The efficiency of the iterative algorithm (2.12) depends on whether the first equation of (2.12) can be solved quickly. The augmented Lagrangian function $\mathcal{L}$ still contains nondifferentiable terms. But different from the original objective function (2.3), the hinge loss induced nondifferentiability has now been transferred from terms involving $1 - y_i(\mathbf{x}_i^T\beta + \beta_0)$ to terms involving a_i; and the ℓ_1-norm induced nondifferentiability has now been transferred from terms involving β to terms involving $\mathbf{c}$. Moreover, the nondifferentiable terms involving $\mathbf{a}$ and $\mathbf{c}$ are now completely decoupled. Therefore, we can solve the first equation of (2.12)

by alternately minimizing (β, β_0), $\mathbf{a}$ and $\mathbf{c}$,

$$\begin{cases} (\beta^{k+1}, \beta_0^{k+1}) = \arg\min_{\beta,\beta_0} \mathcal{L}\left(\beta, \beta_0, \mathbf{a}^k, \mathbf{c}^k, \mathbf{u}^k, \mathbf{v}^k\right), \\ \mathbf{a}^{k+1} = \arg\min_{\mathbf{a}} \mathcal{L}\left(\beta^{k+1}, \beta_0^{k+1}, \mathbf{a}, \mathbf{c}^k, \mathbf{u}^k, \mathbf{v}^k\right), \\ \mathbf{c}^{k+1} = \arg\min_{\mathbf{c}} \mathcal{L}\left(\beta^{k+1}, \beta_0^{k+1}, \mathbf{a}^{k+1}, \mathbf{c}, \mathbf{u}^k, \mathbf{v}^k\right). \end{cases} \tag{2.13}$$

For the Method of Multipliers, the alternate minimization (2.13) should run multiple times until convergence. However, we do not have to completely solve the first equation of (2.12) since it is only one step of the overall iterative algorithm. Instead, we use only one alternation, which is called the Alternating Direction Method of Multipliers [42]. Specifically, we use the following iterations to solve (2.6)

$$\begin{cases} (\beta^{k+1}, \beta_0^{k+1}) = \arg\min_{\beta,\beta_0} \mathcal{L}\left(\beta, \beta_0, \mathbf{a}^k, \mathbf{c}^k, \mathbf{u}^k, \mathbf{v}^k\right), \\ \mathbf{a}^{k+1} = \arg\min_{\mathbf{a}} \mathcal{L}\left(\beta^{k+1}, \beta_0^{k+1}, \mathbf{a}, \mathbf{c}^k, \mathbf{u}^k, \mathbf{v}^k\right), \\ \mathbf{c}^{k+1} = \arg\min_{\mathbf{c}} \mathcal{L}\left(\beta^{k+1}, \beta_0^{k+1}, \mathbf{a}^{k+1}, \mathbf{c}, \mathbf{u}^k, \mathbf{v}^k\right), \\ \mathbf{u}^{k+1} = \mathbf{u}^k + \mu_1(\mathbf{1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}) - \mathbf{a}^{k+1}), \\ \mathbf{v}^{k+1} = \mathbf{v}^k + \mu_2(\beta^{k+1} - \mathbf{c}^{k+1}). \end{cases} \tag{2.14}$$

The first equation in (2.14) is equivalent to

$$\begin{aligned} (\beta, \beta_0) = \arg\min_{\beta,\beta_0} & \frac{\lambda_2}{2}\|\beta\|_2^2 + \left\langle \mathbf{v}^k, \beta - \mathbf{c}^k \right\rangle + \left\langle \mathbf{u}^k, \mathbf{1} - Y(X\beta + \beta_0\mathbf{1}) - \mathbf{a}^k \right\rangle \\ & + \frac{\mu_1}{2}\|\mathbf{1} - Y(X\beta + \beta_0\mathbf{1}) - \mathbf{a}^k\|_2^2 + \frac{\mu_2}{2}\|\beta - \mathbf{c}^k\|_2^2. \end{aligned}$$

The objective function in the above minimization problem is quadratic and differentiable.

Therefore, the optimal solution can be found by solving a set of linear equations:

$$\begin{aligned}&\begin{pmatrix}(\lambda_2+\mu_2)I+\mu_1 X^T X & \mu_1 X^T\mathbf{1}\\ \mu_1\mathbf{1}^T X & \mu_1 n\end{pmatrix}\begin{pmatrix}\beta^{k+1}\\ \beta_0^{k+1}\end{pmatrix}\\ =&\begin{pmatrix}X^T Y\mathbf{u}^k-\mu_1 X^T Y(\mathbf{a}^k-\mathbf{1})-\mathbf{v}^k+\mu_2\mathbf{c}^k\\ \mathbf{1}^T Y\mathbf{u}^k-\mu_1\mathbf{1}^T Y(\mathbf{a}^k-\mathbf{1})\end{pmatrix}.\end{aligned} \tag{2.15}$$

Note that the coefficient matrix in (2.15) is a $(p+1)\times(p+1)$ matrix. For small p, we can store its inverse in the memory, so the linear equations can be solved with minimal cost. For large p, we use the conjugate gradient algorithm (CG) to solve it at each iteration efficiently.

The linear system (2.15) has very special characteristics for "large p, small n" problems: as $X^T X$ is a positive-definite low-rank matrix with rank at most n, the coefficient matrix in (2.15) is a linear combination of an identity matrix and a positive-definite low-rank matrix with rank at most $n+1$. If we use CG to solve the linear system (2.15), it converges in less than $n+1$ steps [88]. In our numerical implementation, we found that CG converges in a few steps much less than $n+1$.

The second equation in (2.14) is equivalent to

$$\begin{aligned}\mathbf{a}^{k+1}=\arg\min_{\mathbf{a}}\frac{1}{n}\sum_{i=1}^{n}(a_i)_+&+\frac{\mu_1}{2}\|\mathbf{1}-Y(X\beta^{k+1}+\beta_0^{k+1}\mathbf{1})-\mathbf{a}\|_2^2\\ &+\langle\mathbf{u}^k,\mathbf{1}-Y(X\beta^{k+1}+\beta_0^{k+1}\mathbf{1})-\mathbf{a}\rangle.\end{aligned} \tag{2.16}$$

In order to solve (2.16), we need the following Proposition [110].

Proposition 2.1. *Let* $s_\lambda(\omega) = \arg\min_{x\in\mathbb{R}} \lambda x_+ + \frac{1}{2}\|x-\omega\|_2^2$. *Then*

$$s_\lambda(\omega) = \begin{cases} \omega - \lambda, & \omega > \lambda \\ 0, & 0 \le \omega \le \lambda, \\ \omega, & \omega < 0. \end{cases}$$

Note that each a_i is independent of one another in (2.16) and

$$\begin{aligned} &\frac{\|\mathbf{u}\|_2^2}{2\mu_1} + \frac{\mu_1}{2}\|\mathbf{1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}) - \mathbf{a}\|_2^2 + \langle \mathbf{u}^k, \mathbf{1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}) - \mathbf{a}\rangle \\ =&\frac{\mu_1}{2}\|\mathbf{a} - (\mathbf{1} + \frac{\mathbf{u}}{\mu_1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}))\|_2^2. \end{aligned}$$

Together with Proposition 2.1, we can then update $\mathbf{a}^{k+1}$ in (2.16) according to

Corollary 2.1. *The update of* $\mathbf{a}^{k+1}$ *in* (2.16) *is equivalent to*

$$\mathbf{a}^{k+1} = \mathcal{S}_{\frac{1}{n\mu_1}}(\mathbf{1} + \frac{\mathbf{u}^k}{\mu_1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1})), \tag{2.17}$$

where

$$\mathcal{S}_\lambda(\omega) = (s_\lambda(\omega_1), s_\lambda(\omega_2), \ldots, s_\lambda(\omega_n)), \forall \omega \in \mathbb{R}^n.$$

The third equation in (2.14) is equivalent to

$$\mathbf{c}^{k+1} = \arg\min_{\mathbf{c}} \lambda_1\|\mathbf{c}\|_1 + \langle \mathbf{v}^k, \beta^{k+1} - \mathbf{c}\rangle + \frac{\mu_2}{2}\|\beta^{k+1} - \mathbf{c}\|_2^2. \tag{2.18}$$

Minimization of $\mathbf{c}$ in (2.18) can be done efficiently using soft thresholding, because the objective function is quadratic and the nondifferentiable terms are completely separable.

Let $\mathcal{T}_\lambda$ be a soft thresholding operator defined in the vector space,

$$\mathcal{T}_\lambda(\omega) = (t_\lambda(\omega_1), \ldots, t_\lambda(\omega_p)), \forall \omega \in \mathbb{R}^p, \tag{2.19}$$

where

$$t_\lambda(\omega_i) = \text{sgn}(\omega_i) \max\{0, |\omega_i| - \lambda\}.$$

Using the soft thresholding operator (2.19), the optimal solution of $\mathbf{c}$ in (2.18) can be written as

$$\mathbf{c}^{k+1} = \mathcal{T}_{\frac{\lambda_1}{\mu_2}} \left(\frac{\mathbf{v}^k}{\mu_2} + \beta^{k+1} \right). \tag{2.20}$$

Finally, by combining (2.14), (2.15), (2.17) and (2.20) together, we obtain the ADMM algorithm for ENSVM (2.3) (Algorithm 1). It is a practical algorithm for "large p, small n" problems and is very easy to implement.

Algorithm 1 Alternating Direction Method of Multipliers for Solving the Elastic Net SVM (ADMM-ENSVM)

Initialize β^0, β_0^0, $\mathbf{a}^0, \mathbf{c}^0, \mathbf{u}^0$, and $\mathbf{v}^0$.
repeat
1) Update $\beta^{k+1}, \beta_0^{k+1}$ by solving the following linear equation system:

$$\begin{pmatrix} (\lambda_2 + \mu_2)I + \mu_1 X^T X & \mu_1 X^T \mathbf{1} \\ \mu_1 \mathbf{1}^T X & \mu_1 n \end{pmatrix} \begin{pmatrix} \beta^{k+1} \\ \beta_0^{k+1} \end{pmatrix} = \begin{pmatrix} X^T Y \mathbf{u}^k - \mu_1 X^T Y(\mathbf{a}^k - \mathbf{1}) - \mathbf{v}^k + \mu_2 \mathbf{c}^k \\ \mathbf{1}^T Y \mathbf{u}^k - \mu_1 \mathbf{1}^T Y(\mathbf{a}^k - \mathbf{1}) \end{pmatrix}.$$

2) $\mathbf{a}^{k+1} = \mathcal{S}_{\frac{1}{n\mu_1}} \left(\mathbf{1} + \frac{\mathbf{u}^k}{\mu_1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}) \right)$.
3) $\mathbf{c}^{k+1} = \mathcal{T}_{\frac{\lambda_1}{\mu_2}} \left(\frac{\mathbf{v}^k}{\mu_2} + \beta^{k+1} \right)$.
4) $\mathbf{u}^{k+1} = \mathbf{u}^k + \mu_1(\mathbf{1} - Y(X\beta^{k+1} + \beta_0^{k+1}\mathbf{1}) - \mathbf{a}^{k+1})$.
5) $\mathbf{v}^{k+1} = \mathbf{v}^k + \mu_2(\beta^{k+1} - \mathbf{c}^{k+1})$.
until
Convergence

2.3.2 Convergence Analysis

The convergence property of Algorithm 1 can be derived using the standard convergence theory of the Alternating Direction Method of Multipliers [42, 31].

Theorem 2.1. *Suppose there exists at least one solution* (β^*, β_0^*) *of* (2.3). *Assume* $\lambda_1 > 0, \lambda_2 > 0$. *Then the following property for Algorithm 1 holds:*

$$\begin{aligned}&\lim_{k\to\infty} \frac{1}{n}\sum_{i=1}^{n}(1 - y_i(\mathbf{x}_i^T\beta^k + \beta_0^k))_+ + \lambda_1\|\beta^k\|_1 + \frac{\lambda_2}{2}\|\beta^k\|_2^2\\ =&\frac{1}{n}\sum_{i=1}^{n}(1 - y_i(\mathbf{x}_i^T\beta^* + \beta_0^*))_+ + \lambda_1\|\beta^*\|_1 + \frac{\lambda_2}{2}\|\beta^*\|_2^2.\end{aligned}$$

Furthermore,

$$\lim_{k\to\infty} \|(\beta^k, \beta_0^k) - (\beta^*, \beta_0^*)\| = 0,$$

whenever (2.3) *has a unique solution.*

2.3.3 Computational Cost

The efficiency of Algorithm 1 mainly depends on whether we can quickly solve the linear equations (2.15). As we have described in Section 2.3.1, the coefficient of the linear equations (2.15) has a special structure and thus can be efficiently solved by the conjugate gradient method for "large p, small n" problems. More specifically, the computational cost for solving (2.15) is $O(n^2p)$. The number of iterations of Algorithm (1) is hard to predict and it depends on the choice of μ_1 and μ_2. According to our experience, we only need to iterate a few hundred iterations to get a reasonable result provided that μ_1 and μ_2 are chosen properly.

Similar to our algorithm for (2.3), the major computational cost in each iteration for HHSVM

also comes from solving a linear system. However, the linear system in HHSVM has no special structures. It takes at least $O(|\mathcal{A}|^2)$ with $|\mathcal{A}|$ being the number of unknowns variables. Moreover, $|\mathcal{A}|$ can increase at each iteration. Furthermore, for large scale problems, it usually takes a few thousand steps for the algorithm to converge. That's why our algorithm for (2.3) is much faster than the path algorithm for HHSVM for large scale problems.

2.4 Numerical Results

In this section, we use time trials on both simulation data and real microarray data to illustrate the efficiency of ADMM-ENSVM. To evaluate the performance of ADMM-ENSVM, we also compare it with the stochastic subgradient method and the path algorithm for HHSVM. Our algorithm and the stochastic subgradient method were implemented in Matlab, while HHSVM was implemented in R using the R code provided by the authors of [106]. All algorithms were compiled on a windows platform and time trials were generated on an Intel Core 2 Duo desktop PC (E7500, 2.93GHz).

The stopping criteria of Algorithm 1 is specified as follows. Let $\Phi(\beta^k, \beta_0^k) = \frac{1}{n}\sum_{i=1}^{n}(1 - y_i(\mathbf{x}_i^T\beta^k + \beta_0^k))_+ + \lambda_1\|\beta^k\|_1 + \frac{\lambda_2}{2}\|\beta^k\|_2^2$. According to Theorem 2.1, $\lim_{k\to\infty}\Phi(\beta^k, \beta_0^k) = \Phi(\beta^*, \beta_0^*)$. It is reasonable to terminate the algorithm when the relative change of the energy functional $\Phi(\beta, \beta_0)$ falls below certain threshold δ. Furthermore, the linear constraints in (2.6) should be satisfied when Algorithm 1 converges. Therefore, we would expect that $\frac{1}{\sqrt{n}}\|\mathbf{1} - Y(X\beta^k + \beta_0^k\mathbf{1}) - \mathbf{a}^k\|_2 \le \delta$ and $\frac{1}{\sqrt{p}}\|\beta^k - \mathbf{c}^k\|_2 \le \delta$ when the algorithm is terminated. We used $\delta = 10^{-5}$ in our experiments. To summarize, we stop Algorithm 1 whenever

$$RelE := \frac{|\Phi(\beta^k, \beta_0^k) - \Phi(\beta^*, \beta_0^*)|}{\max\{1, \Phi(\beta^k, \beta_0^k)\}} \le 10^{-5},$$

$$\frac{1}{\sqrt{n}}\|\mathbf{1} - Y(X\beta^k + \beta_0^k\mathbf{1}) - \mathbf{a}^k\|_2 \leq 10^{-5},$$

and

$$\frac{1}{\sqrt{p}}\|\beta^k - \mathbf{c}^k\|_2 \leq 10^{-5}.$$

Note that the convergence of Algorithm 1 is guaranteed as shown in Theorem 2.1, no matter what values μ_1 and μ_2 are set to be. However, the choices of μ_1 and μ_2 can influence the speed of the algorithm, as it will affect the number of iterations involved. In our implementation, we found that empirically setting $\mu_1 = \frac{100}{n}$ and $\mu_2 \in [50, 100]$ works well for all the experiments, though the parameter selecting procedure can certainly be further improved.

2.4.1 Simulation

We consider a binary classification problem, where the samples are lying in a p dimensional space. Only the first 10 dimensions are relevant for classification, and the remaining dimensions are all noises. More specifically, we generate n samples: half of them are labeled as "+1" and the other half are labeled as "−1". The samples from the "+1" class are i.i.d. drawn from a normal distribution with mean

$$\mu_+ = (\underbrace{1, \ldots, 1}_{10}, \underbrace{0, \ldots, 0}_{p-10})^T,$$

and covariance matrix

$$\Sigma = \begin{pmatrix} \Sigma^*_{10\times 10} & \mathbf{0}_{10\times(p-10)} \\ \mathbf{0}_{(p-10)\times 10} & I_{(p-10)\times(p-10)} \end{pmatrix},$$

where the diagonal elements of Σ^* are 1 and the off-diagonal elements are all equal to ρ. The "-1" class has a similar distribution except that

$$\mu_- = (\underbrace{-1, \ldots, -1}_{10}, \underbrace{0, \ldots, 0}_{(p-10)})^T.$$

So the Bayes optimal classification rule depends on $x_1, \ldots, x_{10}$, which are highly correlated if ρ is large. The Bayes error is independent of the dimension p. This kind of simulation data is also used in [106].

Table 2.1: Running times (CPU seconds) of ADMM-ENSVM, the path algorithm and the stochastic sub gradient method (SSG) for HHSVM. The algorithms are run with various (p, n) pairs and two correlation ρ between the features. The results for ADMM-ENSVM and SSG are averaged over 25 runs (using 25 different values of λ_1, λ_2) and the results for HHSVM are averaged over 5 runs (using 5 different values of λ_2).

n, p	Method	$\rho = 0$	$\rho = 0.8$
n=50	ENSVM	0.41	0.31
	HHSVM	3.30	3.19
p=300	SSG	2.35	4.06
n=100	ENSVM	1.19	0.71
	HHSVM	21.65	21.01
p=500	SSG	4.34	4.06
n=200	ENSVM	3.60	3.75
	HHSVM	405.9	390.1
p=1000	SSG	35.40	26.86
n=300	ENSVM	14.73	16.74
	HHSVM	2.07 hours	2.03 hours
p=2000	SSG	123.22	122.84
n=400	ENSVM	48.62	57.15
	HHSVM	$>$ 6 hours	$>$ 6 hours
p=5000	SSG	301.10	290.15
n=500	ENSVM	144.69	170.52
	HHSVM	-	-
p=10000	SSG	785.57	909.92

Table 2.1 shows the average CPU times (seconds) used by the ADMM algorithm, the path

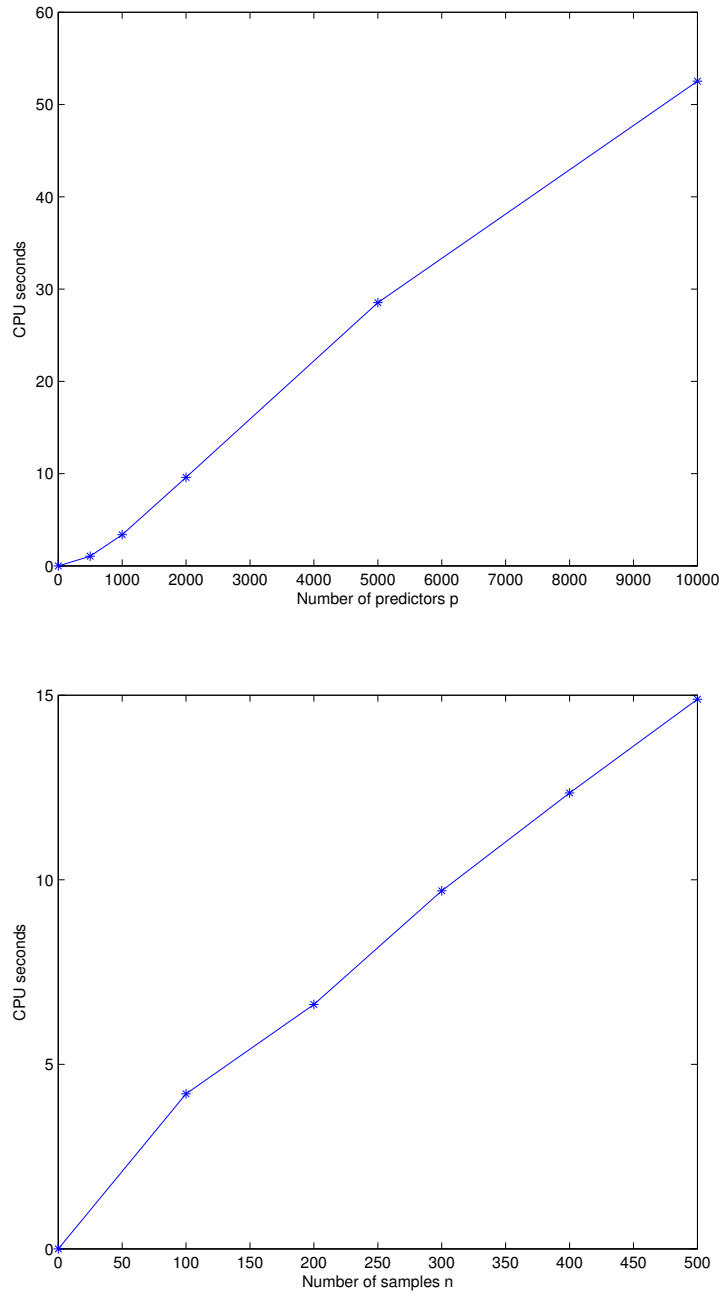


Figure 2.1: CPU times of ADMM-ENSVM for the same problem as in Table 2.1, for different values of n and p. In each case the times are averaged over 10 runs. (a) n is fixed and equals to 300; (b) p is fixed and equals to 2000.

algorithm for HHSVM, and the stochastic sub-gradient method. Our algorithm consistently outperforms both the stochastic sub-gradient method and the path algorithm in all cases we have tested. For the data with $n = 300, p = 2000$, the ADMM algorithm is able to achieve 120-fold speedup than the path algorithm. The ADMM algorithm is also significantly faster than the stochastic subgradient method, achieving about 5-10 fold speedup in all cases. We should also note that unlike the ADMM method, the objective function in the stochastic subgradient method can go up and down, which makes it difficult to design the stopping criteria.

To evaluate how the performance of our algorithm scales up with the problem size, we plot the CPU time that Algorithm 1 takes to solve (2.3) for the data described above as a function of p and n. Figure 2.1 shows such a curve, where the CPU times are averaged over 10 runs with different data. We note that the CPU times are roughly linear in both n and p.

We also compare the performance of prediction accuracy and variable selection of three different models: ℓ_1-norm SVM (L_1 SVM), HHSVM and ENSVM. The optimal (λ_1, λ_2) pair is chosen from a large grid using 10-fold cross-validation. As shown in Table 2.2 and Table 2.3, HHSVM and ENSVM are similar in prediction and variable selection accuracy, but both are significantly better than ℓ_1-norm SVM.

Table 2.2: Comparison of test errors. The number of training samples is 50. The total number of input variables is 300, with only 10 relevant for classification. The results are averaged testing errors over 100 random repetitions, and the numbers in parentheses are the corresponding standard errors. $\rho = 0$ corresponds to the case where the input variables are independent, while $\rho = 0.8$ corresponds to a pairwise correlation of 0.8 between relevant variables.

	$\rho = 0$	$\rho = 0.8$
SVM	0.214(0.004)	0.160(0.003)
L_1 SVM	0.143(0.007)	0.160(0.002)
HHSVM	0.133(0.005)	0.143(0.001)
ENSVM	0.111(0.002)	0.144(0.001)

Table 2.3: Comparison of variable selection. The setup is the same as that described in Table 2.2. q_{signal} is the number of selected relevant variables, and q_{noise} is the number of selected noise variables.

	$\rho = 0$		$\rho = 0.8$	
	q_{signal}	q_{noise}	q_{signal}	q_{noise}
L_1 SVM	7.2(0.3)	6.5(1.4)	2.5(0.2)	2.9(1.2)
HHSVM	7.6(0.3)	7.1(1.3)	7.9(0.4)	3.3(2.5)
ENSVM	8.6(0.1)	6.4(0.4)	6.6(0.2)	2.0(0.2)

2.4.2 Gene Expression Data

A microarray gene expression dataset typically contains the expression values of tens of thousands of mRNAs collected from a relatively small number of samples. Genes sharing the same biological pathways often have highly correlated expression levels [90]. Because of these two features, it is more desirable to apply the Elastic Net SVM to do variable selection and classification on the microarray data than the standard SVM or the ℓ_1-SVM [117, 106].

The dataset we investigate is taken from the paper published by Alon et al. [2]. It contains microarray gene expressions from 62 samples (40 colon tumor tissues and 22 normal tissues). Each sample consists of the expression values of $p = 2000$ genes. We apply the Elastic Net SVM to select variables (i.e. genes) that can be used to predict sample labels and compare its performance to the path algorithm developed for HHSVM. The results are summarized in Table 2.4, which shows the computational times spent by different solvers in a ten-fold cross-validation procedure with different parameters λ_1 and λ_2. The ADMM algorithm for ENSVM is consistently many times faster than the path algorithm for HHSVM, with an approximately ten-fold speedup in almost all cases.

We also evaluate the prediction accuracy and the variable selection functionality of ENSVM with Algorithm 1. Following the method in [106], we randomly split the samples into a

Table 2.4: Running times (CPU seconds) for different values of the regularization parameters λ_1 and λ_2. The methods are ADMM-ENSVM and the path algorithm for HHSVM.

λ_1	λ_2	10-CV error	ENSVM	HHSVM
0.1	0.2	8/62	8.56	108.2
0.1	0.5	8/62	6.30	107.9
0.05	2	8/62	8.76	109.3
0.05	5	7/62	7.12	109.5

Table 2.5: Comparison of testing error and variable selection on the gene expression data. Results are averaged over 100 repetitions and the numbers in the parenthesis are the standard deviations.

	Test error	Number of genes selected
SVM	17.9% (0.69%)	All
HHSVM	15.45%(0.59%)	138.37(8.67)
ENSVM	14.95% (0.53%)	87.7 (7.9)

training set (27 cancer samples and 15 normal tissues) and a testing set (13 cancer samples and 7 normal tissues). In the training phase, we adopt 10-fold cross-validation to tune the parameter λ_1, λ_2. This experiment is repeated 100 times. Table 2.5 shows the statistics of the testing error and the number of selected genes, in comparison to the statistics of SVM and HHSVM. We note that in terms of the testing error, ENSVM is slightly better than HHSVM, which in turn is better than the standard SVM. In terms of variable selection, ENSVM tends to select a smaller number of genes than HHSVM.

2.5 Discussion

In this chapter, we have developed an efficient algorithm based on the Alternating Direction Method of Multipliers to solve the optimization problem of the Elastic Net SVM. We show

that the proposed algorithm is substantially faster than both the subgradient method and the path algorithm used in HHSVM, an approximation of ENSVM [105]. We also illustrate the advantage of ENSVM in both variable selection and prediction accuracy using simulated and real-world data.

Chapter 3

Efficient Latent Variable Gaussian Graphical Model Selection

3.1 Introduction

Estimating covariance matrices in the high-dimensional setting arises in many applications and has drawn considerable attention recently. Because the sample covariance matrix is often ill-conditioned in the high-dimensional regime, regularizing the sample covariance based on proper assumptions of the underlying true covariance is often essential to gain robustness and stability of the estimation.

One form of regularization that has gained popularity recently is to require that the underlying inverse covariance matrix to be sparse [28, 81, 6, 37]. If the data follow a multivariate Gaussian distribution with covariance matrix Σ, the entries of the inverse covariance matrix $K = \Sigma^{-1}$ (also known as concentration matrix or precision matrix) encode the information of conditional dependencies between variables: $K_{ij} = 0$ if the variables i and j are conditionally independent given all others. Therefore, the sparsity regularization is equivalent to the

assumption that most of the variable pairs in the high-dimensional setting are conditionally independent.

To make the estimation problem computational tractable, one often adopts a convex relaxation of the sparsity constraint and uses the ℓ_1-norm to enforce the sparsity of the concentration matrix [6, 37, 76, 113]. Denote Σ^n the empirical covariance. Under the maximum likelihood framework, the covariance matrix estimation problem is then formulated as solving the following optimization problem:

$$\begin{aligned} \min \quad & -\log\det K + \mathrm{tr}(\Sigma^n K) + \lambda\|K\|_1 \\ s.t. \quad & K \succ 0, \end{aligned} \tag{3.1}$$

where tr denotes the trace, λ is a sparsity regularization parameter, and $K \succ 0$ denotes that K is positive definite. Due to the ℓ_1-norm penalty and the explicit positive definite constraint on K, the method leads to a sparse estimation of the concentration matrix that is guaranteed to be positive definite. The problem is convex and many algorithms have been proposed to solve it efficiently in high-dimensional settings [37, 70, 89, 114].

However, in many real applications only a subset of the variables is directly observed, and no additional information is provided on both the number of latent variables and their relationship with the observed ones. For instance, in the area of functional genomics it is often the case that only mRNAs of the genes can be directly measured, but not the proteins, which are correlated with but have no direct correspondence to the mRNAs because of the prominent role of the posttranscriptional regulation. Another example is the movie recommender system where the preference of a movie can be strongly influenced by latent factors such as advertisements, social environment, etc. In such cases, the observed variables can be densely correlated because of the marginalization over the unobserved hidden variables. Therefore, the sparsity regularization alone may fail to model the data properly.

We consider the setting in which the hidden (X_H) and the observed variables (X_O) are jointly Gaussian with covariance matrix $\Sigma_{(OH)}$. The marginal statistics of the observed variable X_O are given by the marginal covariance matrix Σ_O, which is simply a submatrix of the full covariance matrix $\Sigma_{(OH)}$. Let the concentration matrix $K_{(OH)} = \Sigma_{(OH)}^{-1}$. The marginal concentration matrix Σ_O^{-1} corresponding to the observed variables X_O is given by the Schur complement [18]:

$$\hat{K}_O = \Sigma_O^{-1} = K_O - K_{O,H} K_H^{-1} K_{H,O}, \tag{3.2}$$

where $K_O, K_{O,H}$, and K_H are the corresponding submatrices of the full concentration matrix. Based on the Schur complement, it is clear that the marginal concentration matrix of the observed variables can be decomposed into two components: one is K_O, which specifies the conditional dependencies of the observed variables given both the observed and latent variables; and the other is $K_{O,H} K_H^{-1} K_{H,O}$, which represents the effect of marginalization over the hidden variables. One can now impose assumptions to the two underlying components separately.

By assuming that the K_O matrix is sparse and the number of latent variables is small, the maximum likelihood estimation of the covariance matrix of the observed variables at the presence of latent variables can then be formulated as

$$\begin{aligned} \min_{S,L} \quad & -\log\det(S-L) + \operatorname{tr}(\Sigma_O^n (S-L)) + \lambda_1 \|S\|_1 + \lambda_2 \operatorname{tr}(L) \\ s.t. \quad & S - L \succ 0, \quad L \succeq 0, \end{aligned} \tag{3.3}$$

where we decompose $\Sigma_O^{-1} = S - L$ with S denoting K_O and L denoting $K_{O,H} K_H^{-1} K_{H,O}$. Because the number of the hidden variables is small, L is of low rank, whose convex relaxation is the trace norm. There are two regularization parameters in this model: λ_1 regularizes the sparsity of S, and λ_2 regularizes the rank of L. Under certain regularity conditions,

Chandrasekaran et al. showed that this model can consistently estimate the underlying model structure in the high-dimensional regime in which the number of observed/hidden variables grow with the number of samples of the observed variables [18].

The objective function in (3.3) is strictly convex, so a global optimal solution is guaranteed to exist and to be unique. However, finding the optimal solution in the high-dimension setting is computationally challenging, due to the log det term and the trace norm term in the likelihood function, the nondifferentiability of the ℓ_1-norm penalty, and the positive semidefinite constraints. For large-scale problems, the state-of-the-art algorithm for solving (3.3) is the special-purpose algorithm LogdetPPA [103] developed for log-determinant semidefinite programs. However, LogdetPPA is designed to solve smooth problems. In order to use LogdetPPA, one has to reformulate (3.3) as a smooth problem. As a result, no optimal sparse matrix S can be generated and additional heuristic steps involving thresholding have to be applied in order to enforce sparsity. In addition, LogdetPPA is not specially designed for (3.3). We believe a much more efficient algorithm can be developed by exploiting the unique structure of the model.

The main contribution of this chapter contains two aspects. First, we present a new algorithm for solving (3.3) and show that the algorithm is significantly faster than the state-of-the-art method, especially for large-scale problems. The algorithm is derived by reformulating the problem and adapting the Split Bregman method [89, 114]. We derive a closed-form solution for each subproblem involved in the Split Bregman iterations. Second, we apply the method to analyze a large-scale gene expression data, and find that the model with latent variables explains the data much better than the one without latent variables. In addition, we find that most of the correlations between genes can be explained by only a few latent factors, which provides a new perspective for analyzing this type of data.

The rest of the chapter is organized as follows. In Section 3.2, we derive a Split Bregman method, called SBLVGG, to solve the latent variable Gaussian graphical model (3.3). The

convergence property of the algorithm is also given. SBLVGG consists of four update steps and each update has explicit formulas to calculate. In Section 3.3, we illustrate the utility of our algorithm and compare its performance to LogdetPPA using both simulated data and gene expression data.

3.2 Split Bregman Method for Latent Variable Gaussian Graphical Model Selection

The Split Bregman method was originally proposed by Osher and his colleagues to solve total variation based image restoration problems [45]. It was later found to be either equivalent or closely related to a number of other existing optimization algorithms, including the Douglas-Rachford splitting [107], the Alternating Direction Method of Multipliers (ADMM) [42, 44, 45] and the Method of Multipliers [87]. Because of the fast convergence and the easiness of implementation, it has become a method of choice for solving large-scale sparsity recovery problems [15, 16]. Recently, it is also used to solve (3.1) and is found to be very successful [89, 114].

In this section, we first reformulate the problem by introducing an auxiliary variable and then proceed to derive a Split Bregman method to solve the reformulated problem. Here we would like to emphasize that, although the Split Bregman method has been introduced to solve graphical model problems [89, 114], our algorithm is different from theirs in three aspects. First, it is the first time to use the Split Bregman method to solve (3.3) and we introduce an auxiliary variable for the data fitting term instead of the penalty term which has been adopted in [89, 114]. Second, we provide an explicit formula for the third update, which did not appear in [89, 114]. Third, instead of using the eigenvalue (or Schur) decomposition as done in the previous work [89, 114], we use the LAPACK routine dsyevd.f (based on

a divide-and-conquer strategy) to compute the full eigenvalue decomposition of symmetric matrices, which is essential for solving the first and the third subproblems.

3.2.1 Derivation of the Split Bregman Algorithm for Latent Variable Gaussian Graphical Model Selection

The log-likelihood term and the regularization terms in (3.3) are coupled, which makes the optimization problem difficult to solve. However, the three terms can be decoupled if we introduce an auxiliary variable to transfer the coupling from the objective function to the constraints. More specially, the problem (3.3) is equivalent to the following problem:

$$
\begin{aligned}
(\hat{A}, \hat{S}, \hat{L}) = \arg\min_{A,S,L} & -\log\det A + tr(\Sigma_O^n A) + \lambda_1\|S\|_1 + \lambda_2 tr(L) \\
s.t. \quad & A = S - L \\
& A \succ 0, L \succeq 0.
\end{aligned} \tag{3.4}
$$

The introduction of the new variable A is a key step of our algorithm, which makes the problem amenable to the Split Bregman procedure to be detailed below. Although the Split Bregman method originates from the Bregman iteration, it has been demonstrated to be equivalent to the Alternating Direction Method of Multipliers (ADMM) [42, 44, 91]. For simplicity of presentation, we will derive the Split Bregman method using the augmented Lagrangian method [53, 87].

First, the augmented Lagrangian function of (3.4) is defined as,

$$
\begin{aligned}
\mathcal{L}(A, S, L, U) := & -\log\det A + tr(\Sigma_O^n A) + \lambda_1\|S\|_1 + \lambda_2 tr(L) \\
& + tr(U(A - S + L)) + \frac{\mu}{2}\|A - S + L\|_F^2,
\end{aligned} \tag{3.5}
$$

where U is a dual matrix variable corresponding to the equality constraint $A = S - L$, and $\mu > 0$ is a parameter. Compared with the standard Lagrangian function, the augmented Lagrangian function has an extra term $\frac{\mu}{2}\|A - S + L\|_F^2$, which penalizes the violation of the linear constraint $A = S - L$.

With the definition of the augmented Lagrangian function (3.5), the primal problem (3.4) is equivalent to

$$\min_{A\succ 0, L\succeq 0, S} \max_{U} \mathcal{L}(A, S, L, U). \tag{3.6}$$

Exchanging the order of min and max in (3.6) leads to the formulation of the dual problem,

$$\max_{U} E(U) \quad \text{with} \quad E(U) = \min_{A\succ 0, L\succeq 0, S} \mathcal{L}(A, S, L, U). \tag{3.7}$$

Note that the gradient $\nabla E(U)$ can be calculated by the following [12],

$$\nabla E(U) = A(U) - S(U) + L(U), \tag{3.8}$$

where $(A(U), S(U), L(U)) = \arg\min_{A\succ 0, L\succeq 0, S} \mathcal{L}(A, S, L, U)$.

Applying gradient ascent on the dual problem (3.7) and using equation (3.8), we obtain the Method of Multipliers [87] to solve (3.4),

$$\begin{cases} (A^{k+1}, S^{k+1}, L^{k+1}) = \arg\min_{A\succ 0, L\succeq 0, S} \mathcal{L}(A, S, L, U^k), \\ U^{k+1} = U^k + \mu(A^{k+1} - S^{k+1} + L^{k+1}). \end{cases} \tag{3.9}$$

Here we have used μ as the step size of the gradient ascent. It is easy to see that the efficiency of the iterative algorithm (3.9) largely depends on whether the first equation of (3.9) can be

solved efficiently. Note that the augmented Lagrangian function $\mathcal{L}(A, S, L, U^k)$ still contains A, S, L and can not easily be solved directly. But we can solve the first equation of (3.9) through an iterative algorithm that alternates between the minimization of A, S and L. The Method of Multipliers requires that the alternative minimization of A, S and L are run multiple times until convergence to get the solution $(A^{k+1}, S^{k+1}, L^{k+1})$. However, because the first equation of (3.9) represents only one step of the overall iteration, it is actually not necessary to solve it completely. In fact, the Split Bregman method (or the Alternating Direction Method of Multipliers [42]) uses only one alternative iteration to get a very rough solution of (3.9), which leads to the following iterative algorithm for solving (3.4) after some reformulations,

$$\begin{cases} A^{k+1} = \arg\min_{A\succ 0} -\log\det(A) + tr(A\Sigma_O^n) + \frac{\mu}{2}\left\|A - S^k + L^k + \frac{U^k}{\mu}\right\|_F^2, \\ S^{k+1} = \arg\min_S \lambda_1\|S\|_1 + \frac{\mu}{2}\left\|A^{k+1} - S + L^k + \frac{U^k}{\mu}\right\|_F^2, \\ L^{k+1} = \arg\min_{L\succeq 0} \lambda_2 tr(L) + \frac{\mu}{2}\left\|A^{k+1} - S^{k+1} + L + \frac{U^k}{\mu}\right\|_F^2, \\ U^{k+1} = U^k + \mu(A^{k+1} - S^{k+1} + L^{k+1}). \end{cases} \tag{3.10}$$

Convergence

The convergence of the iteration (3.10) can be derived from the convergence theory of the Alternating Direction Method of Multipliers or the convergence theory of the Split Bregman method [42, 31, 15].

Theorem 3.1. *Let* (S^k, L^k) *be generated by* (3.10)*, and* $(\hat{S}, \hat{L})$ *be the unique minimizer of* (3.4)*. Then,*

$$\lim_{k\to\infty} \|S^k - \hat{S}\| = 0 \quad \textit{and} \quad \lim_{k\to\infty} \|L^k - \hat{L}\| = 0.$$

From Theorem 3.1, the condition for the convergence of the iteration (3.10) is quite mild and even irrelevant to the choice of the parameter μ in the iteration (3.10).

Explicit formulas to update A, S and L

We first focus on the computation of the first equation of (3.10). Taking the derivative of the objective function and setting it to be zero, we get

$$-A^{-1} + \Sigma_O^n + U^k + \mu(A - S^k + L^k) = 0. \tag{3.11}$$

It is a quadratic equation where the unknown variable is a matrix. The complexity for solving this equation is at least $O(p^3)$ because of the matrix inversion involved in (3.11). Note that $S = S^T$ and $L = L^T$. Therefore, if U^k is symmetric, $\Sigma_O^n + U^k - \mu(S^k - L^k)$ is symmetric as well. It is easy to show that the explicit form for the solution of (3.11) under constraint $A \succ 0$ is

$$A^{k+1} = \frac{K^k + \sqrt{(K^k)^2 + 4\mu I}}{2\mu}, \tag{3.12}$$

where $K^k = \mu(S^k - L^k) - \Sigma_O^n - U^k$ and $\sqrt{C}$ denotes the square root of a symmetric positive definite matrix C. Recall that the square root of a symmetric positive definite matrix C is defined as a matrix whose eigenvectors are the same as those of C and whose eigenvalues are the square root of those of C. Therefore, to get the update of A^{k+1}, one can first compute the eigenvalues and eigenvectors of K^k, then get the eigenvalues of A^{k+1} according to (3.12) by replacing K with its eigenvalues and I with 1, and finally multiply the eigenvectors back. We adopt the LAPACK routine dsyevd.f (based on a divide-and-conquer strategy) to compute the full eigenvalue decomposition of $(K^k)^2 + 4\mu I$. It is about 10 times faster than eig (or schur) routine when n is larger than 500.

For the second equation of (3.10), we have made the data fitting term $\frac{\mu}{2}\left\|A^{k+1}-S+L^k+\frac{U^k}{\mu}\right\|_F^2$ separable with respect to the entries of S. The sparsity term $\|S\|_1$ is also separable. Thus, it is very easy to get the solution and the computational complexity would be $O(p^2)$. Let $\mathcal{T}_\lambda$ be a soft thresholding operator defined in the matrix space that satisfies

$$\mathcal{T}_\lambda(\Omega) = (t_\lambda(\omega_{ij}))_{i,j=1}^p,$$

where $t_\lambda(\omega_{ij}) = \text{sgn}(\omega_{ij})\max\{0, |\omega_{ij}| - \lambda\}$. Then the update of S is

$$S^{k+1} = \mathcal{T}_{\frac{\lambda_1}{\mu}}(A^{k+1} + L^k + \mu^{-1}U^k).$$

For the update of L, the following theorem is required.

Theorem 3.2. *Given a symmetric matrix X and $\eta > 0$. Denote*

$$\mathcal{S}_\eta(X) = \arg\min_{Y \succeq 0} \eta tr(Y) + \frac{1}{2}\|Y - X\|_F^2.$$

Then $\mathcal{S}_\eta(X) = V diag((\lambda_i - \eta)_+)V^T$, where $\lambda_i (i \in 1, ..., n)$ are the eigenvalues of X with V being the corresponding eigenvector matrix and $(\lambda_i - \eta)_+ = \max(0, \lambda_i - \eta)$.

Proof. Note that $tr(Y) = \langle I, Y\rangle$, where I is the identity matrix. Thus, $\arg\min_{Y\succeq 0} \eta tr(Y) + \frac{1}{2}\|Y - X\|_F^2 = \arg\min_{Y\succeq 0}\langle Y - X + \eta I, Y - X + \eta I\rangle$. Denote the eigenvalue decomposition of matrix X as $X = V\Lambda V^T$, where $VV^T = V^TV = I$ and Λ is a diagonal matrix. Then

$$\langle Y - X + \eta I, Y - X + \eta I\rangle = \langle V^TYV - (\Lambda - \eta I), V^TYV - (\Lambda - \eta I)\rangle.$$

Together with the fact that $\mathcal{S}_\eta(X) \succeq 0$, $\mathcal{S}_\eta(X)$ should satisfy $(V^T\mathcal{S}_\eta(X)V)_{ij} = \max(0, \lambda_i - \eta)$ for $i = j$ and 0 otherwise. Therefore, $\mathcal{S}_\eta(X) = V diag((\lambda_i - \eta)_+)V^T$. ∎

Using the operator $\mathcal{S}_\eta$ defined in Theorem 3.2, it is easy to see that

$$L^{k+1} = \mathcal{S}_{\frac{\lambda_2}{\mu}}(S^{k+1} - A^{k+1} - \mu^{-1}U^k). \tag{3.13}$$

Here we also use the LAPACK routine dsyevd.f (based on a divide-and-conquer strategy) to compute the full eigenvalue decomposition of $S^{k+1} - A^{k+1} - \mu^{-1}U^k$. Putting all components together, we get SBLVGG to solve the latent variable Gaussian graphical model (3.3) as shown in Algorithm 2.

Algorithm 2 Split Bregman Method for Solving the Latent Variable Gaussian Graphical Model (SBLVGG)

Initialize S^0, L^0, U^0.
repeat
1) $A^{k+1} = \frac{K^k + \sqrt{(K^k)^2 + 4\mu I}}{2\mu}$, where $K^k = \mu(S^k - L^k) - \Sigma - U^k$.
2) $S^{k+1} = \mathcal{T}_{\frac{\lambda_1}{\mu}}(A^{k+1} + L^k + \mu^{-1}U^k)$.
3) $L^{k+1} = \mathcal{S}_{\frac{\lambda_2}{\mu}}(S^{k+1} - A^{k+1} - \mu^{-1}U^k)$.
4) $U^{k+1} = U^k + \mu(A^{k+1} - S^{k+1} + L^{k+1})$.
until
Convergence

3.3 Numerical Experiments

Next we illustrate the efficiency of the Split Bregman method (SBLVGG) for solving (3.3) using time trials on an artificial dataset as well as a gene expression dataset. All the algorithms were implemented in Matlab and run on a 64-bit linux desktop with Intel i3 - 3.2GHz QuadCore CPU and 8GB memory. To evaluate the performance of SBLVGG, we compare it with logdetPPA [103], the state-of-the-art solver for (3.3) in the large-scale case. In order to solve (3.3) using LogdetPPA , we need to reformulate (3.3) as a smooth problem as done in [18], which makes the derived matrix $\hat{S}$ not strictly sparse with many entries close to but not exactly 0. We also demonstrate that the latent variable Gaussian graphical model (3.3) is

better than the sparse Gaussian graphical model (3.1) in terms of the generalization ability evaluated on the gene expression data.

Note that the convergence of Algorithm 2 is guaranteed as shown in Theorem 3.1, no matter what value μ is set to. The speed of the algorithm can, however, be influenced by the choices of μ as it would affect the number of iterations involved. In our implementation, we choose μ in $[0.005, 0.01]$ for the artificial data and $[0.001, 0.005]$ for the gene expression data.

3.3.1 Artificial Data

Let $p = p_o + p_h$, where p is the total number of variables in the graph, p_o is the number of observed variables and p_h is the number of hidden variables. The synthetic dataset is generated in a similar way as that in Section 6.1 of [103]. First, we generate a $p \times p$ random sparse matrix W with non-zero entries drawn from the normal distribution $\mathcal{N}(0, 1)$. Then, we set

$$
\begin{aligned}
&C = W' * W; \quad C(1 : p_o, p_o + 1 : p) = C(1 : p_o, p_o + 1 : p) + 0.5 * randn(p_o, p_h);\\
&C = (C + C')/2; \quad d = diag(C); \quad C = max(min(C - diag(d), 1), -1);\\
&K = B + max(-1.2 * min(eig(B)), 0.001) * eye(p); \quad K_O = K(1 : p_o, 1 : p_o)\\
&K_{OH} = K(1 : p_o, p_o + 1 : p); \quad K_{HO} = K(p_o + 1 : p, 1 : p_o);\\
&K_H = K(p_o + 1 : p, p_o + 1 : p); \tilde{K}_O = K_O - K_{OH} K_H^{-1} K_{HO}.
\end{aligned}
$$

Note that $\tilde{K}_O$ is the marginal precision matrix of observed variables. We generate n Gaussian random samples from $\tilde{K}_O$, and calculate the sample covariance matrix Σ_O^n. In our numerical experiments, we set the sparsity ratio of K_O around 5%, and $p_h = 10$. The stopping criteria is specified as follows. Let $\Phi(A, L) = -\log\det A + tr(A\Sigma) + \lambda_1 \|A + L\|_1 + \lambda_2 tr(L)$. We stop our algorithm if $|\Phi(A^{k+1}, L^{k+1}) - \Phi(A^k, L^k)| / \max\left(1, |\Phi(A^{k+1}, L^{k+1})|\right) < \epsilon$ and $\|A - S + L\|_F < \epsilon$

Table 3.1: Numerical comparison at $p_o = 3000, p_h = 10$ for the artificial data

(λ_1, λ_2)	Method	Obj. Value	Rank	Sparse Ratio
(0.0025, 0.21)	SBLVGG	-5642.6678	8	5.56%
	LodgetPPA	-5642.6680	8	99.97%
(0.0025, 0.22)	SBLVGG	-5642.4894	3	5.58%
	LodgetPPA	-5642.4895	3	99.97%
(0.0027, 0.21)	SBLVGG	-5619.2744	16	4.14%
	LodgetPPA	-5619.2746	16	99.97%
(0.0027, 0.22)	SBLVGG	-5619.0194	6	4.17%
	LodgetPPA	-5619.0196	6	99.97%

with $\epsilon = 1e - 4$.

Figure 3.1a shows CPU time curves of SBLVGG and LogdetPPA with respect to the number of variable (p) for the artificial data. For each fixed p, the CPU time is averaged over four runs with four different (λ_1, λ_2) pairs. We can see that SBLVGG consistently outperforms LogdetPPA. When $p_o \leq 2500$, it is 3.5 times faster on average; when $p_o = 3000$, it is 4.5 times faster. This illustrates that SBLVGG scales better to problem size than LogdetPPA. In terms of accuracy, Table 3.1 summarizes performance of two algorithms at $p_o = 3000$, $p_h = 10$ in three aspects: objective value, rank of L, sparsity of S (ratio of non-zero off-diagonal elements). We find that in terms of objective value and rank, the two algorithms generate almost identical results. However, SBLVGG outperforms LogdetPPA in terms of sparsity of S, thanks to its soft-thresholding operator in Algorithm 2. LogdetPPA doesn't have such kind of operations. It generates many nonzero entries that are very close to zero due to numerical error. We would like to emphasize that the results in lower dimensions are very similar to $p_o = 3000$, $p_h = 10$. We omit the details here.

3.3.2 Gene Expression Data

The gene expression dataset [57] contains mRNA expression levels of the 6316 genes of *S. cerevisiae* (yeast) under 300 different experimental conditions. First we centralize the data and choose three subset of the data, i.e., 1000, 2000 and 3000 genes with the highest variances. Figure 3.1b shows CPU time of SBLVGG and LogdetPPA with different p. We can see that SBLVGG consistently performs better than LogdetPPA: in 1000-dimensional case, SBLVGG is 2.5 times faster, while in 2000- and 3000-dimensional cases, it is almost 3 times faster. Table 3.2 summarizes the accuracy for the 3000-dimensional case in three aspects: the objective value, the rank of L, the sparsity of S (Number of non-zero off-diagonal elements) for four fixed pairs of (λ_1, λ_2). Similar to the case of the artificial data, SBLVGG and LogdetPPA generate identical results in terms of objective value and number of hidden units. However, LogdetPPA suffers from the floating point problem, and is not able to generate exact sparse matrices; on the other hand, SBLVGG is doing much better.

Table 3.2: Numerical comparison at 3000-dimensional subset of the gene expression data

(λ_1, λ_2)	Algorithm	Obj. Value	Rank	# Non-0 Entries
(0.01,0.05)	SBLVGG	-9793.3451	88	34
	LodgetPPA	-9793.3452	88	8,997,000
(0.01,0.1)	SBLVGG	-9607.8482	60	134
	LodgetPPA	-9607.8483	60	8,997,000
(0.02,0.05)	SBLVGG	-8096.2115	79	0
	LodgetPPA	-8096.2115	79	8,996,998
(0.02,0.1)	SBLVGG	-8000.9047	56	0
	LodgetPPA	-8000.9045	56	8,997,000

We also investigate the generalization ability of the latent variable Gaussian graphical model (3.3) versus that of the sparse Gaussian graphical model (3.1). A subset of the data, i.e., 1000 genes with the highest variances, are used for this experiment. The 300 samples are randomly divided into 200 for training and 100 for testing. We use the following quantity to

evaluate the generalization ability,

$$NLoglike = -\log\det A + tr(A\Sigma^n),$$

where Σ^n is the empirical covariance matrix of the observed samples and A is the estimated covariance matrix based on model (3.3) or model (3.1). It is easy to see that $NLoglike$ is equivalent to the negative log likelihood function up to some scaling constant. Therefore, it is reasonable to use $NLoglike$ as a criteria for cross-validation and prediction. Regularization parameters λ_1, λ_2 for model (3.3) and λ for model (3.1) are selected by 10-fold cross-validation on the training set. Table 3.3 shows that the latent variable Gaussian graphical model (3.3) consistently outperforms the sparse Gaussian graphical model (3.1) in terms of $NLoglike$. We also note that the latent variable Gaussian graphical model (3.3) tends to use a moderate number of hidden units, and a very sparse conditional correlation matrix to explain the data. For $p = 1000$, it tends to predict about 50 hidden units, and about tens of direct interconnections between observed variables (sometimes even 0). This suggests that most of the correlations between observed genes in the mRNA expression measurements can be explained by only a small number of latent factors. Currently we only evaluate the generalization ability using $NLoglike$. The initial result on the gene expression data is encouraging. Further evaluation will be performed by incorporating other prior information or by comparing with known gene interactions.

3.4 Discussion

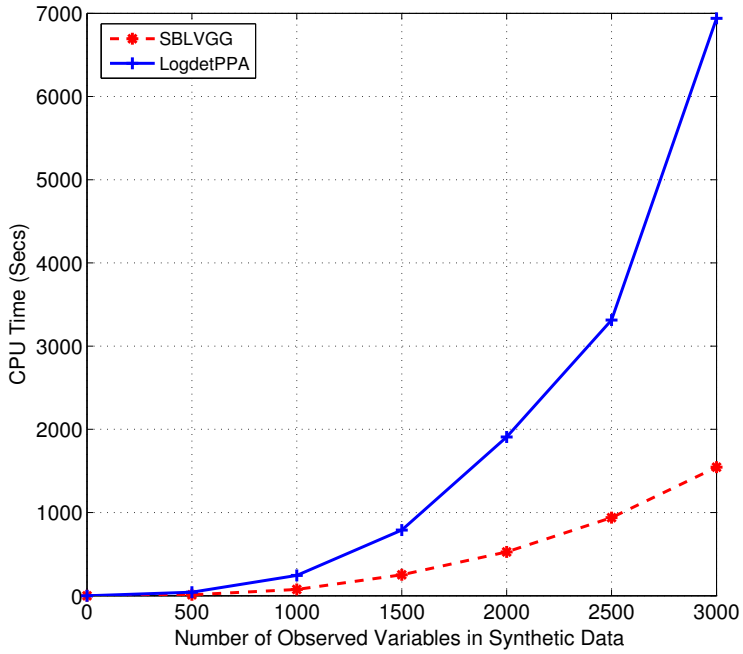
Graphical model selection in high-dimensions arises in a wide range of applications, where it is common that only a subset of the variables are directly observable. In this scenario, the marginal concentration matrix of the observed variables is generally not sparse due to the marginalization effect of latent variables. A computationally attractive approach is to

Table 3.3: Comparison of generalization ability on the gene expression data at dimension of 1000 using latent variable Gaussian graphical model (LVGG) and sparse Gaussian graphical model (SGG)

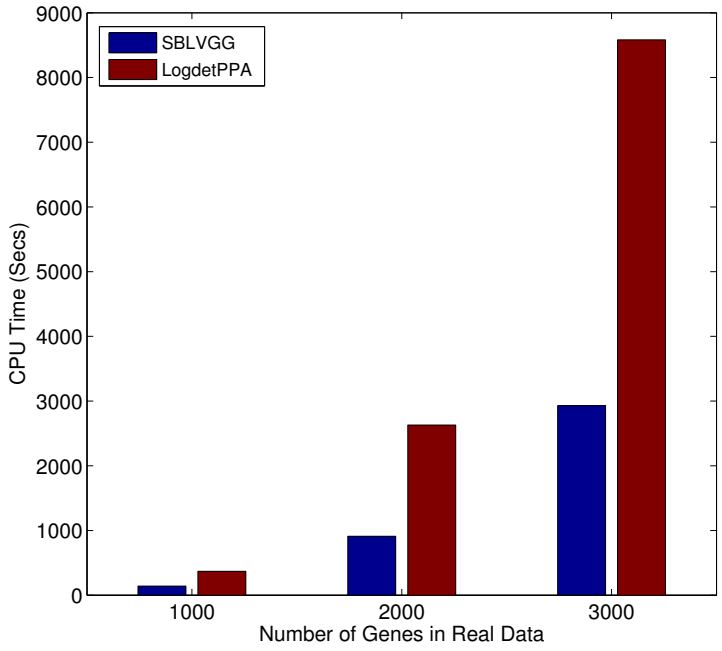
Exp. Number	LVGG			SGG	
	Rank of L	Sparsity of S	$NLoglike$	Sparsity of K	$NLoglike$
1	48	30	-2191.3	24734	-1728.8
2	47	64	-2322.7	28438	-1994.1
3	50	58	-2669.9	35198	-2526.3
4	52	64	-2534.6	30768	-2282.5
5	48	0	-2924.0	29880	-2841.4
6	51	52	-2707.1	28754	-2642.6
7	45	0	-2873.3	30374	-2801.4
8	49	0	-2765.5	31884	-2536.7
9	48	54	-2352.0	29752	-2087.2
10	47	0	-2922.9	29760	-2843.5

decompose the marginal concentration matrix into a sparse matrix and a low-rank one, which reveals both the conditional graphical model structure of the observed variables and the effect of the hidden variables. However, solving the regularized maximum likelihood problem is nontrivial at large-scale, because of the complexity of the log-likelihood term, the trace norm penalty and the ℓ_1-norm penalty. In this chapter, we propose a new approach based on the Split Bregman method (SBLVGG) to solve this problem. We show that our algorithm is at least three times faster than the state-of-the-art solver for large-scale problems.

We have applied the method to analyze the genes' expressions of yeast in a dataset of thousands of genes measured at 300 different experimental conditions. It is interesting to note that the model considering the latent variables consistently outperforms the one without latent variables in term of the testing likelihood. We also note that most of the correlations observed between mRNAs can be explained by only tens of latent variables. The observation is consistent with the module network idea proposed in the genomics community. It might also suggest that the posttranscriptional regulation plays a more prominent role than it is previously appreciated.



(a)



(b)

Figure 3.1: (a) Comparison of CPU time curves w.r.t. number of variables p for the artificial data; (b) Comparison of CPU time bar charts w.r.t. number of variables p for the gene expression data

Chapter 4

Ensemble Learning of Concordance Index for Cancer Survival Analysis

4.1 Introduction

Survival analysis focuses on developing diagnostic and prognostic models to analyze the effect of covariates on the outcome of an event of interest, such as death or disease recurrence in disease studies. The analysis is often carried out using regression methods to estimate the relationship between the covariates and the *time to event* variable. In clinical trials, time to events are usually represented by *survival times*, which measure how long a patient with a localized disease is alive or disease-free after treatment, such as surgery or surgery plus adjuvant therapy. The covariates used in predicting survival times often include clinical features, such as age, disease status, treatment type, etc. More recently, molecular features, such as expression of genes, and genetic features, such as mutations in genes, have been increasingly included in the set of covariates. Survival analysis also has applications in many other fields. For instance, it is often used to model machine failure in mechanical systems.

Depending on specific circumstances, survival times may also be referred to as *failure times*.

A major complication for survival analysis is that the survival data are often incomplete due to censoring, because of which standard statistical and machine learning tools on regression cannot be readily applied. The most common type of censoring that occurrs in clinical trials is the right censoring, where the survival time is known to be longer than a certain value but its precise value is unknown. This can be due to multiple reasons. For instance, a patient might withdraw from a clinical trial, or a clinical trail might end early such that some patients are not followed up with afterwards.

Many statistical methods have been developed for survival analysis. One major category of these methods adopts a likelihood-based approach. An essential component of the models in this category is the estimation of the hazard function $\lambda(t)$, defined as the event rate at time t conditional on survival up to time t. Different models often impose different assumptions on the forms of the hazard function. In particular, the proportional hazards (PH) model (also called the Cox model), one of most prevalent models in survival analysis, assumes that different covariates contribute multiplicatively to the hazard function [23, 24, 3, 74]. To relax the proportional hazards assumption and allow for more complicated relationships between covariates, parametric models based on artificial neural networks (ANN) [34, 73, 85, 86] and ensembles of tree models based on boosting [39, 40, 83, 84] have also been proposed. In order to handle the censored data, all these models use an approximation of the likelihood function, called the Cox partial likelihood, to train the predictive model. The partial likelihood function is computationally convenient to use; however, it is unclear how well the full likelihood can be approximated by the partial likelihood.

Many other methods aiming at optimizing a different class of objective functions rather than the partial likelihood have also been proposed. Some of these methods adapt existing regression models to estimate the relationship between survival times and covariates, by taking the censored data into account in training the models [93, 100], while others adopt

a classification-based framework and train their models using only the rank information associated with the observed survival times [17, 33, 86]. Recently, random survival forests [56, 59], a new ensemble-of-trees model based upon bagging, became popular in survival analysis. They resort to predicting either the cumulative hazard function, or the log-transformed survival time.

In clinical decision-making, physicians and researchers are often more interested in evaluating the *relative risk* of a disease between patients with different covariates than the absolute survival times of these patients. For this purpose, Harrell et al. introduced the important concept of *concordance index* (C-index, concordance C, or simply CI) as a measure of the separation between two survival distributions [48, 49]. Given two survival distributions, the C-index computes the fraction of pairs of patients with consistent risk orders over the total number of validly comparable pairs. Because of its focus on assessing the accuracy of relative risk, the C-index is widely adopted in survival model performance evaluation, where the order of predicted survival times is compared to the order of the observed ones [108, 82, 62].

Our goal in this chapter is to develop a new survival model to capture the relationship between survival times and covariates by directly optimizing the C-index between the predicted and observed survival times. Although both the Cox model based on the partial likelihood and the ranking based methods mentioned above also utilize only the order information between survival times, the C-index based method provides a more principled way of combining all pair-wise order information into a single metric. There have been prior attempts in directly learning the C-index for survival analysis, including a neural network based model [108] and an extension of the Cox model trained using a lower bound of C-index [82]. However, both methods impose parametric assumptions on the effect of covariates on survival times. Our contribution here is to adopt a nonparametric approach to model the relationship between survival times and covariates by using an ensemble of trees, and to

train the ensemble model by learning the C-index.

In the following, we will provide a detailed description of our ensemble survival model based on learning the C-index. We will derive an algorithm to train the model using the gradient boosting method originally proposed by Friedman [39]. The algorithm is implemented in an R software packaged called GBMCI (gradient boosting machine for concordance index). We benchmark the performance of GBMCI using a large-scale breast cancer prognosis dataset and show that GBMCI outperforms several popular survival models, including the Cox PH model, the gradient boosting PH model, and the random survival forest, in a number of covariate settings.

4.2 Survival Analysis, Existing Models and the New Approach

4.2.1 Survival Analysis

We review the basic concepts of survival analysis here. For a systematic treatment, see [25, 1]. In survival analysis, the time to event (death, failure, etc) t is typically modeled as a random variable, which follows some probability density distribution $p(t)$. The density can be characterized by the *survival function* $S(t) = Pr(T > t) = \int_t^\infty p(T)\mathrm{d}T$ for $t > 0$. The survival function captures the probability that the event does not happen until time t. A closely-related concept is the *hazard function* $\lambda(t) = \lim_{\Delta t \to 0} \frac{Pr(t<T<t+\Delta t|T>t)}{\Delta t} = \frac{p(t)}{S(t)}$, which measures the event rate at time t conditioned on survival until t. One can further show that $S(t) = e^{-\int_0^t \lambda(\tau)\mathrm{d}\tau}$.

The likelihood function for right-censored survival data is expressed as:

$$L\left(\theta; \{x_i, t_i, \delta_i\}_{i=1}^n\right) = \prod_{i \in E} p(t_i|x_i, \theta) \prod_{j \in C} S(t_j|x_j, \theta) = \prod_{i=1}^{n} \lambda(t_i|x_i, \theta)^{\delta_i} S(t_i|x_i, \theta). \tag{4.1}$$

Note the augmentation of our notation (We will follow this convention in the following context unless otherwise state): θ is the set of regression parameters of the survival/hazard model; $\delta_i, i = 1, ..., n$ indicates whether the event happens ($\delta = 1$), or not ($\delta = 0$, i.e., the data is censored); $x_i, i = 1, ..., n$ are the explanatory covariates that affect the survival time; E is the set of data whose events are observed, and C is the set of censored data. The full maximum-likelihood approach would optimize L over the functional space of S (or λ) and parameter space of θ. Unfortunately, this is often intractable.

Proportional hazard model

In his seminal work [23, 24], Cox introduced the *proportional hazard* (PH) model $\lambda(t|x, \theta) = \lambda_0(t) \exp\{x^T\theta\}$. $\lambda_0(t)$ is the *baseline* hazard function; $\exp\{x^T\theta\}$ is the relative hazard, which summarizes the effect of covariates. Cox observed that under the PH assumption, it suffices to estimate θ without the necessity of specifying $\lambda_0(t)$ and optimizing the likelihood (4.1). Instead, he proposed to optimize the so-called *Cox partial likelihood*,

$$L_p\left(\theta; \{x_i, t_i, \delta_i\}_{i=1}^n\right) = \prod_{i \in E} \frac{\exp\{\theta^T x_i\}}{\sum_{j:t_j \geq t_i} \exp\{\theta^T x_j\}}. \tag{4.2}$$

The Cox model has become very popular in evaluating the covariates' effect on survival data, and has been generalized to handle time-varying covariates and time-varying coefficients [3, 74]. However, the proportional hazards assumption and the maximization of the partial likelihood remain two main limitations. Nonlinear models, e.g., multi-layer neural networks [34, 73, 85], have been proposed to replace $\theta^T x$. However, they still assume parametric forms of the hazard function and attempt to optimize the partial likelihood.

Concordance index

The *C-index* is a commonly used performance measure of survival models. Intuitively, it is the fraction of all pairs of patients whose predictions have correct orders over the pairs that can be ordered. Formally, the C-index is,

$$CI = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} I(F(x_i) < F(x_j)) = \frac{1}{|\mathcal{P}|} \sum_{i\in E} \sum_{j:t_j>t_i} I(F(x_i) < F(x_j)). \tag{4.3}$$

$\mathcal{P}$ is the set of validly orderable pairs where $t_i < t_j$; $|\mathcal{P}|$ is the number of pairs in $\mathcal{P}$; $F(x)$ is the prediction of survival time; I is the indicator function of whether the condition in the parentheses is satisfied or not. In the PH setting, the predicted survival time can be equivalently represented by the negative log relative hazard. The C-index estimates the probability that the order of the predictions of a pair of comparable patients is consistent with their observed survival information.

4.2.2 Gradient Boosting Machine

The *gradient boosting machine* (GBM) is an ensemble learning method, which constructs a predictive model by additive expansion of sequentially fitted weak learners [39, 40]. The general problem is to learn a functional mapping $y = F(x;\beta)$ from data $\{x_i, y_i\}_{i=1}^n$, where β is the set of parameters of F, such that some cost function $\sum_{i=1}^n \Phi(y_i, F(x_i;\beta))$ is minimized. Boosting assumes $F(x)$ follows an "additive" expansion form $F(x) = \sum_{m=0}^{M} \rho_m f(x;\tau_m)$, where f is called the *weak* or *base learner* with a weight ρ and a parameter set τ. Accordingly, $\{\rho_m, \tau_m\}_{m=1}^M$ compose the whole parameter set β. They are learnt in a greedy "stage-wise" process: (1) set an initial estimator $f_0(x)$; (2) for each iteration $m \in \{1, 2, .., M\}$, solve $(\rho_m, \tau_m) = \arg\min_{\rho,\tau} \sum_{i=1}^{n} \Phi(y_i, F_{m-1}(x_i) + \rho f(x_i;\tau))$. GBM approximates (2) with two steps.

First, it fits $f(x;\tau_m)$ by

$$\tau_m = \arg\min_{\tau} \sum_{i=1}^{n} \left(g_{im} - f(x_i;\tau)\right)^2, \tag{4.4}$$

where

$$g_{im} = -\left[\frac{\partial \Phi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}. \tag{4.5}$$

Second, it learns ρ by

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{n} \Phi\left(y_i, F_{m-1}(x_i) + \rho f(x_i;\tau_m)\right). \tag{4.6}$$

Then, it updates $F_m(x) = F_{m-1}(x) + \rho_m f(x;\tau_m)$. In practice however, *shrinkage* is often introduced to control overfitting, and the update becomes $F_m(x) = F_{m-1}(x) + \nu\rho_m f(x;\tau_m)$, where $0 < \nu \leq 1$. If the weak learner is the regression tree, the complexity of $f(x)$ is determined by tree parameters, e.g., the tree size (or depth), and the minimum number of samples in terminal nodes. Besides using proper shrinkage and tree parameters, one could improve the GBM performance by *subsampling*, i.e., fitting each base learner on a random subset of the training data. This method is called *Stochastic Gradient Boosting* [40].

Compared to parametric models such as *generalized linear models* (GLM) [75] and neural networks, GBM does not assume any functional form of F but uses additive expansion to build up the model. This non-parametric approach gives more freedom to researchers. GBM combines predictions from the ensemble of weak learners, and so tends to yield more robust results than the single learner. Empirically, it also works better than the bagging-based random forests [38], probably due to its functional optimization motivation. However, it requires the cost function Φ to be differentiable with respect to F. GBM has been implemented in the popular open-source R package "gbm" [84] which supports several regression models.

Boosting the Proportional Hazard model

Ridgeway [83] adapted GBM for the Cox model. The cost function is the negative log partial likelihood

$$\Phi(y, F) = -\sum_{i=1}^{n} \delta_i \left\{ F(x_i) - \log \left(\sum_{j:t_j \geq t_i} e^{F(x_j)} \right) \right\}. \tag{4.7}$$

One can then apply (4.4), (4.5) and (4.6) to learn each additive model. In the "gbm" package, this cost function corresponds to the "coxph" distribution and is further optimized to re-fit terminal nodes with Newton's method. We denote this particular GBM algorithm as GBMCOX, and its implementation in the "gbm" package as "gbmcox".

4.2.3 Concordance Index Learning via Gradient Boosting

We now propose a gradient boosting algorithm to learn the C-index. As the C-index is a widely used metric to evaluate survival models, previous works [108, 82] have investigated the possibility to optimize it, instead of Cox's partial likelihood. However, these works are limited to parametric models, such as linear models or neural networks. Our key contribution is to tackle the problem from a non-parametric ensemble perspective based on gradient boosting.

Optimizing the C-index directly is difficult because of its discrete nature, i.e., the summation over indicator functions in (4.3). We resort to the differentiable approximation proposed in [108], which adopts the logistic sigmoid function in each term. We call it the *smoothed concordance index* (SCI). Specifically,

$$SCI = \frac{1}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} \frac{1}{1 + e^{\alpha(F(x_i) - F(x_j))}}, \tag{4.8}$$

where α is a hyper-parameter that controls the steepness of the sigmoid function (accordingly,

the approximability of SCI to CI), and $F(x)$ is the prediction of survival time. Let $\Phi(y, F) = -SCI$. Then, at each iteration $m > 0$ of gradient boosting,

$$\begin{aligned} g_{im} &= \left[\frac{\partial SCI}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \\ &= \frac{\alpha}{|\mathcal{P}|}\left\{\sum_{(k,i)\in\mathcal{P}} \frac{e^{\alpha(F_{m-1}(x_k)-F_{m-1}(x_i))}}{\left[1+e^{\alpha(F_{m-1}(x_k)-F_{m-1}(x_i))}\right]^2}\right. \\ &\quad \left. - \sum_{(i,j)\in\mathcal{P}} \frac{e^{\alpha(F_{m-1}(x_i)-F_{m-1}(x_j))}}{\left[1+e^{\alpha(F_{m-1}(x_i)-F_{m-1}(x_j))}\right]^2}\right\}. \end{aligned} \tag{4.9}$$

So the base learner $f(x; \tau_m)$ can be fitted using $\{g_{im}\}_{i=1}^n$ and (4.4). Next,

$$\rho_m = \arg\max_{\rho} \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \frac{1}{1+e^{\alpha(F_{m-1}(x_i)+\rho f(x_i;\tau_m)-F_{m-1}(x_j)-\rho f(x_j;\tau_m))}}. \tag{4.10}$$

Although differentiable, SCI has a complicated error surface and is neither convex nor concave. This brings two problems. First, the algorithm's performance depends on its initialization which may lead to different local optima; Second, it is difficult to find the global solution of ρ_m in (4.10). In our implementation, we set the initial estimaton $\{f_0(x_i)\}_{i=1}^n$ as the prediction from a fitted PH model, and use line-search to detect ρ_m locally. Empirically, we have found that these heuristics work well for the algorithm.

Algorithm 3, *Gradient Boosting Machine for Concordance Index Learning* (GBMCI), summarizes our whole algorithm, which also incorporates the stochastic boosting mechanism [40]. Note that ensemble size M is an important parameter that requires tuning, as small M may not capture the true model, while large M makes the algorithm apt to overfitting. In practice, it is often selected by cross-validation. We implement GBMCI in the "gbm" package, under a new distribution called "sci", which shares the same regression tree engine and complete software architecture as "gbmcox" does. We name our implementation of GBMCI as "gbmsci".

Algorithm 3 Gradient Boosting Machine for Concordance Index Learning (GBMCI)

Initialize $\{f_0(x_i)\}_{i=1}^n$ with the prediction of Cox's PH model.
Set shrinkage ν, and subsampling size $n_s \leq n$.
for m=1:M **do**
1) Compute $\{g_{im}\}_{i=1}^n$ by (4.9).
2) Randomly select a subset $\{x_i, t_i, \delta_i\}_{i=1}^{n_s}$ from the whole dataset.
3) Fit the weak learner $f(x; \tau_m)$, e.g., a regression tree, upon $\{x_i, g_{im}\}_{i=1}^{n_s}$.
4) Compute ρ_m by (4.10) using line-search.
5) Update $\{F_m(x_i)\}_{i=1}^n$ by $F_m(x) = F_{m-1}(x) + \nu\rho_m f(x; \tau_m)$.
end for

4.3 Results

4.3.1 Dataset and Feature Extraction

We illustrate the utility of GBMCI on a large breast cancer dataset, which was originally released by Curtis et al [26]. The dataset was adopted by the Sage Dream Breast Cancer Challenge (BCC) [72], where it was named *Metabric*. It contains gene expressions, copy number variations, clinical information, and survival data of 1,981 breast cancer patients. The gene expression data consist of 49,576 microarray probes; the copy number data consist of 18,538 SNP probes; the clinical data contain 25 clinical covariates; the survival data contain the survival time and status (dead or censored). Following the convention of BCC, we reserve 1001 patients for training, and the other 980 for testing. We applied several successful feature selection schemes from the top competitors in BCC. See Table 4.1 for details on how these features were generated.

4.3.2 Experimental Settings

As a boosting model, GBMCI's main competitor is the boosted proportional hazard model GBMCOX. As they share identical software environment with a common regression tree engine, the comparison should be reliable and reasonable. For baseline evaluation, we in-

Table 4.1: The five sets of features extracted from the Metabric breast cancer dataset.

Category	Abbreviation	Explanation
Clinical feature	*cl*	A subset of clinical covariates are selected by fitting the Cox model with AIC in a stepwise algorithm. The frequently selected features include age at diagnosis, lymph node status, treatment type, tumor size, tumor group, tumor grade, etc.
Gene feature	*ge*	A subset of gene expression microarray probes using Illumina HT 12v3 platform are selected whose concordance indices to the survival data are ranked highest (positive concordant) or lowest (negative concordant). A few examples are, "ILMN_1683450", "ILMN_2392472", "ILMN_1700337".
Clinical and gene feature	*clge*	A combination of previously selected clinical features and gene expression features are used to fit the Cox model with AIC in a stepwise algorithm, yielding a refined subset of features.
Metagene feature	*mt*	The high-dimensional gene expression data is fed into an iterative *attractor finding* algorithm, yielding a few Attractor Metagenes which are found commonly present in multiple cancer types [20]. Some multi-cancer attractors are strongly associated with the tumor stage, grade, or the lymphocyte status.
Clinical and Metagene feature	*mi*	A minimum subset of Metagenes that have strong prognosis power for breast cancer [20], combined with several important clinical covariates, such as age at diagnosis and treatment type.

vestigate the performance of the PH model with a step-wise Akaike Information Criterion (AIC) model selection scheme (denoted as "cox"). In addition, we also consider the popular random survival forest (RSF) approach by Ishwaran [59], which is implemented in the R package *randomSurvivalForest* [58] (denoted as "rsf"). We use the concordance index as the evaluation criteria. All experiments are performed in R 2.15.1 software environment.

For "gbmsci", the hyper-parameter α controls how well SCI approximates CI. Large α values make the approximation good, but the gradient can be very large or even ill-defined, and

vice versa. In practice, we find $\alpha = 1$ strikes a good balance between approximability and numerical stability. The line-search range is $[0, 100]$ along the gradient direction. The shrinkage ν in "gbm" is 0.001 by default. In our experiments, we find $\nu = 0.002$ works well for "gbmcox"; and $\nu = 1$ does for "gbmsci". We do not essentially apply shrinkage for "gbmsci", because the small line-search range $[0, 100]$ does not necessary detect the global optimal ρ, thus it implicitly contributes to shrinkage. This is mainly for computational efficiency purpose. "gbmsci" and "gbmcox" share other important parameter configurations: maximum number of trees is 1500 (actual number is automatically tuned by 5-fold cross-validation); tree depth is 6; $\frac{n_s}{n}$ (see Algorithm 3) is 1 or 0.5. For "rsf", the number of trees is 1500; other parameters use default configurations.

4.3.3 Empirical Comparison

Each method are tested using the five feature representations in Table 4.1. For "gbmsci" and "gbmcox", as cross-validation introduces randomness by partitioning the training data, we repeat the experiment 50 times. Their predictive concordance indices are shown in Figure 4.1 and 4.2. For "cox", the predictive concordance indices are shown in Table 4.2, which also summarizes the performances of "gbmsci" and "gbmcox". For "rsf", we also do 50 random tests, because of bootstrapping when growing trees. The predictive concordance indices are shown in Figure 4.3.

Figures 4.1 and 4.2 show that "gbmsci" fairly consistently outperforms "gbmcox". The advantage is notable when using the features of *cl*, *clge*, *ge*, *mt* (without subsampling), and substantial when using *mi*. "gbmsci" performs slightly worse only when using *mt* (with subsampling), but is still comparable. Furthermore, all differences except *mt* (with subsampling) are statistically significant (Student's t-test, all p-values $< 10^{-13}$). We also note that subsampling generally improves the predictive power of both "gbmsci" and "gbmcox",

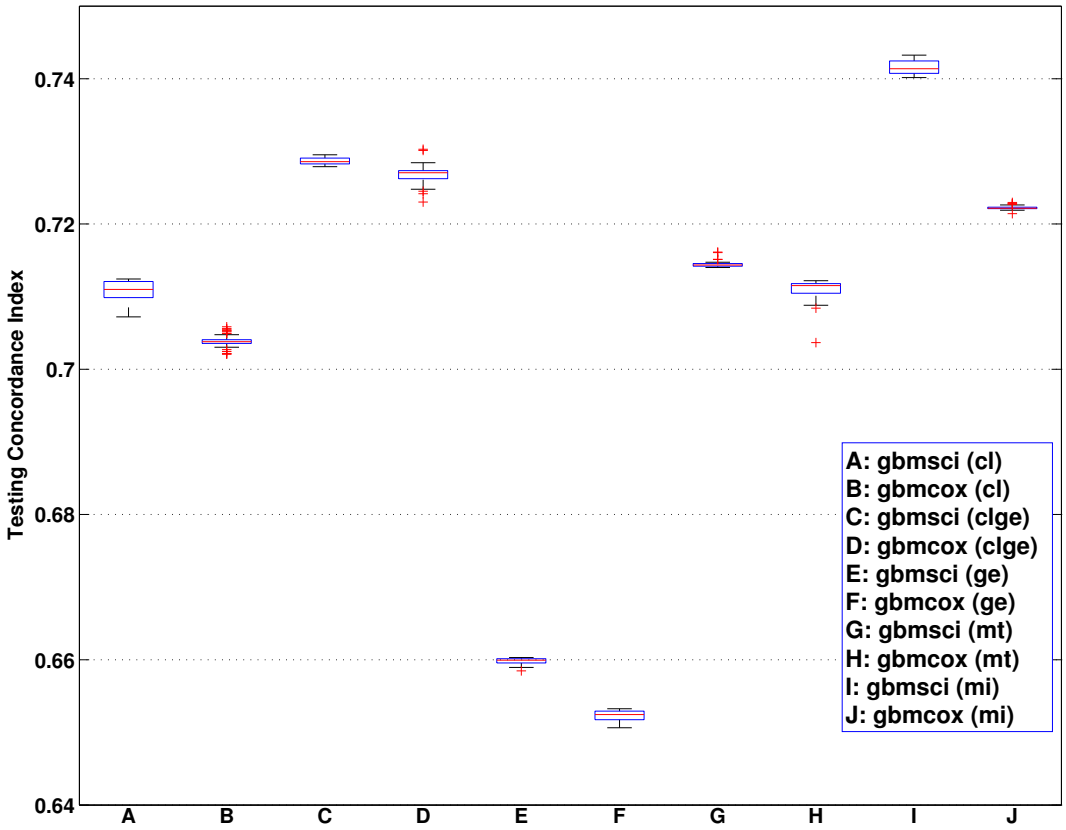


Figure 4.1: Predictive performance I of GBM methods on the breast cancer dataset. The box plots show the predictive concordance indices of "gbmsci" and "gbmcox" in 50 random experiments without subsampling, using the five feature representations explained in Table 4.1. In each box plot, the central red line indicates the median C-index; the blue box is the $[25\%, 75\%]$ area; the black whiskers reach the upper and lower extremes not including outliers; the red "+" symbols represent the outliers.

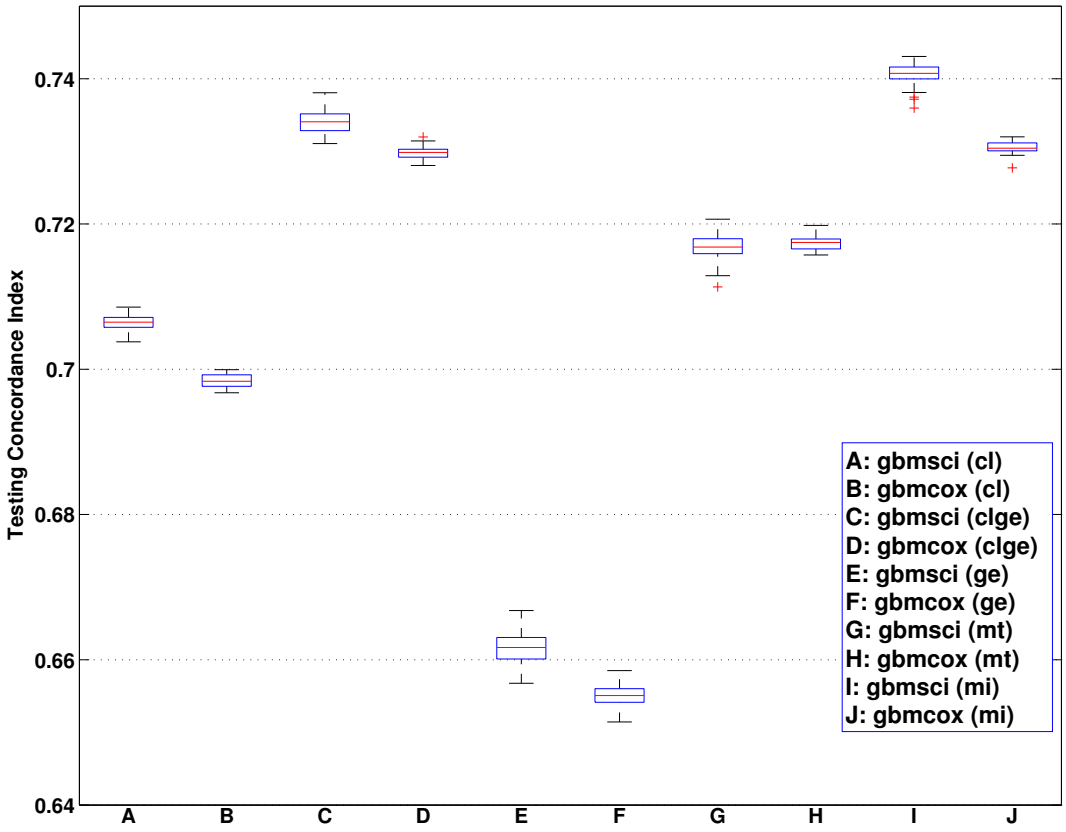


Figure 4.2: Predictive performance II of GBM methods on the breast cancer dataset. The box plots show the predictive concordance indices of "gbmsci" and "gbmcox" in 50 random experiments with subsampling ($\frac{n_s}{n} = 0.5$), using the five feature representations explained in Table 4.1. In each box plot, the central red line indicates the median C-index; the blue box is the $[25\%, 75\%]$ area; the black whiskers reach the upper and lower extremes not including outliers; the red "+" symbols represent the outliers.

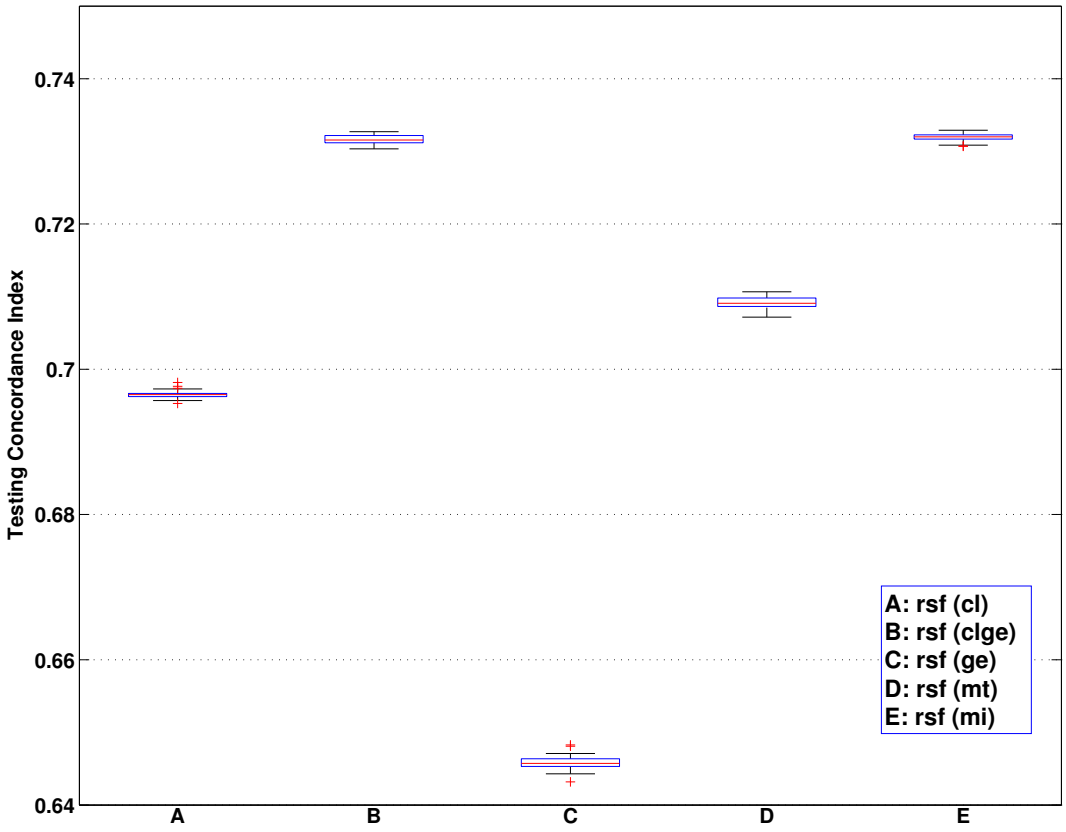


Figure 4.3: Predictive performance of the RSF method on the breast cancer dataset. The box plots show the predictive concordance indices of "rsf" in 50 random experiments, using the five feature representations explained in Table 4.1. In each box plot, the central red line indicates the median C-index; the blue box is the $[25\%, 75\%]$ area; the black whiskers reach the upper and lower extremes not including outliers; the red "+" symbols represent the outliers.

Table 4.2: Numerical statistics of predictive concordance indices of GBM models and the Cox model on the breast cancer dataset. The five feature representations are explained in Table 4.1. "gbmsci"-I and "gbmcox"-I run without subsampling ($\frac{n_s}{n} = 1$), while "gbmsci"-II and "gbmcox"-II run with subsampling ($\frac{n_s}{n} = 0.5$). The numerical values in each entry show the average C-index and the standard deviation (following $\pm$) over 50 random runs. The bold font highlights the best performance in each column.

Model	Feature Representation				
	cl	*clge*	*ge*	*mt*	*mi*
"gbmsci"-I	**0.7107** ±0.0015	0.7287 ±0.0005	0.6599 ±0.0004	0.7145 ± 0.0004	**0.7416** ±0.0010
"gbmcox"-I	0.7039 ±0.0008	0.7268 ±0.0013	0.6523 ±0.0007	0.7110 ±0.0014	0.7222 ±0.0003
"gbmsci"-II	0.7063 ±0.0011	**0.7341** ±0.0014	**0.6617** ±0.0020	0.7169 ±0.0017	0.7405 ±0.0015
"gbmcox"-II	0.6983 ±0.0009	0.7298 ±0.0008	0.6549 ±0.0014	**0.7173** ±0.0010	0.7306 ±0.0008
"cox"	0.7042	0.7140	0.6590	0.6659	0.7299

except when using *cl*. This is consistent with the theoretical argument of [40, 83].

From Table 4.2, one can see "gbmsci" performs better than "cox" overall. The advantage is notable when using *cl* (without subsampling), and substantial when using *clge*, *mt* and *mi*. For other cases, "gbmsci" and "cox" are comparable. On the other hand, "gbmcox" performs better than or comparable to "cox" for *cl*, *clge* and *mt*, but does slightly worse for *ge* and *mi*. Comparing Figures 4.1 and 4.2 with 4.3, one can see "gbmsci" outperforms "rsf" in most cases, while "gbmcox" also performs better than "rsf" overall.

To summarize the comparative study, GBMSCI outperforms GBMCOX, Cox PH and RSF in most of the feature-subsampling settings. The results also shed light on the importance of feature representation. First, gene expression data may have potential prognosis power given well designed feature extraction schemes - for example, the Attractor Metagene (*mt*). Second, combining clinical and gene features together seems to provide enhanced prognosis power over using them separately. This is the case in both the original gene space (*clge*),

and the transformed space (mi).

4.4 Discussion

Many machine learning techniques have been adapted and developed for survival analysis [118, 61, 62]. In particular, several important parametric models, such as neural networks and support vector regression, have been generalized to handle censored data. They provide survival studies with more comprehensive and flexible methodologies. However, ensemble methods are mostly limited to either direct adaptation of boosting to the classical PH model [83, 84], or bagging approaches such as random survival forests [56, 59]. Our proposed algorithm GBMCI generalizes the gradient boosting machine to learn the concordance index directly, which does not impose parametric assumptions on hazard functions and provides a new ensemble learning methodology for survival analysis. As the C-index is a ranking function in essence [82], our model also serves as an ensemble treatment to the *ranking problem* for survival data. This is novel and has not been addressed previously [36, 13, 100].

We implemented GBMCI in an open-source R package, and tested it using a comprehensive cancer prognosis study on the large-scale *Metabric* breast cancer dataset. We found that GBMCI ("gbmsci") performs notably and consistently better than three state-of-the-art survival models (the Cox PH model, "cox", its boosting expansion, "gbmcox", and the Random Survival Forest, "rsf") in terms of predictive C-indices when various feature representations were applied. This study also demonstrates the enhanced prognosis power when gene expression profiles and clinical variables are combined and when the gene space is re-mapped in the predictive model, and the importance of feature engineering of clinical and molecular data in cancer prognosis studies. Interestingly, "gbmsci" typically outperforms "gbmcox" and "cox" when using these informative features. This may provide useful cues for clinical decision-making. Moreover, we also confirm the utility of the subsampling scheme of gradient

boosting.

Although GBMCI has free parameters that require tuning, e.g., α and the line-search range, they empirically work well among different experiments once they have been well tuned. In addition, the algorithm still renders similar performance, when α is within a reasonable neighborhood of 1 (e.g., $\alpha = 2$). One possible reason for the robustness is that both the objective function (4.8) and the gradient (4.9) are upper- and lower-bounded (as can be shown through basic algebraic manipulations). Such bounds are not typically available when optimizing other objective functions for different regression problems, such as the partial likelihood for the Cox model, the mean absolute error for the Lasso regression, and the polynomial alternative of SCI as proposed by [108].

The proposed algorithm has room for improvement. First, current initialization and line-search steps, although working well in practice, are not necessarily the globally optimal strategy. For initialization, one potential alternative is to fit PH models by subsampling or bootstrapping of the training data. To better address the problems, one may have to design other initialization heuristics, or adopt a global optimization technique such as Monte Carlo methods. Second, GBMCI is computationally more intensive than other methods, because of the pairwise sigmoid computation in (4.9) and (4.10). Fortunately, GBMCI is easily parallelizable, which should help in dealing with large datasets. Third, biomedical research often deals with high-throughput data, e.g., microarray gene expression profiles and next generation sequencing data, which require feature selection and dimension reduction. GBMCI does not tackle this task yet. However, as node-splitting of regression trees implicitly perform feature extraction, one could either run GBMCI several iterations and pre-select informative variables as a "warm-up" step before the main learning routine, or start GBMCI with all variables, iteratively rank their node-split frequency and refine the variable pool. These would allow GBMCI to perform feature selection and concordance index learning in a unified framework.

Last but not least, we note that ensemble methods are in general more expensive than the Cox model, because of the necessity of tuning parameters, training ensemble weak learners, and cross-validation. The trade-off between predictive power and computational cost remains a question that depends on specific case requirements. For example, given a particular prognosis analysis task, the Cox model may provide a quick baseline evaluation; ensemble methods could be applied, if higher predictive accuracy and more thorough investigation of covariates' effect are required.

Chapter 5

Deep Learning for Gene Expression Inference

5.1 Introduction

Human mRNA expression (i.e., *gene expression*) profiles are historically adopted as the genomic signature to characterize cellular responses to diseases, genetic perturbations and drug actions. The *Connectivity Map* (CMap) project has built large public databases of such signatures to discover functional connections among diseases, genes and drugs [66, 65]. Conventionally, gene expression profiles are acquired via high-density microarrays such as the Affymetrix platform, which measure the expression of genes across the whole genome. This is prohibitively expensive (e.g., ~$300 per profile) to explore large chemical libraries, which usually contain many genotypes, cell lineages and perturbations, etc. [79, 109].

Despite the large number of genes (~22k), their expressions are strongly correlated, because they tend to regulate the activity of one another by forming a complicated structured regulatory network. The regulatory interactions can be highly non-linear, dynamical and

cascading, depending on different cell lineages or cell chemical environments. Furthermore, most of human genes are believed to be regulated by a small subset of genes that are closely related to transcription factors, grows factors and so on [79, 109]. In particular, the LINCS project of the National Institutes of Health analyzes gene expression profiles of various normal and disease tissue cells in the CMap databases. By doing this, researchers identified ˜1k genes that capture ˜80% of the whole-genome expression information. They call these ˜1k genes the *landmark genes*, and the remaining ˜21k genes the *target genes*.

The LINCS' discovery implies that gene expression profiles can be recovered by measuring landmark genes, a greatly reduced subset of human genes. This further implies a potential reduction of both time and financial costs of human genome profiling. Indeed, with this motivation, LINCS has developed a gene expression profiling assay called L1000, which is based on the Luminex bead platform and directly measures landmark genes at a much lower cost (˜$5 per profile). The unmeasured part of the transcriptome is inferred computationally. With L1000, the LINCS program has generated massive (˜1.3 million) gene expression profiles under various cell types and perturbations.

However, LINCS currently adopts linear regression as the inference tool, which trains regression models separately for each target gene. It potentially conflicts with the fact that genes have intrinsic nonlinear interactions, and that transcriptional programs are often grouped into modules. Kernel machines can express dextrous nonlinear property and have been applied to similar problems [109]. Unfortunately, they suffer from the poor scalability to growing data size, which is a decisive trend of large-scale machine learning.

On the other hand, artificial neural networks can naturally abstract hierarchical nonlinear features that are often shared between multiple tasks (outputs). Recently, they are significantly promoted by the *deep learning* methodologies [7, 29], due to the advancement of modern computer hardwares, in particular, General-Purpose Computing on Graphics Processing Units (GPGPU) [21, 64], the explosive availability of large datasets, and new training

methodologies, such as dropout training [97, 5], momentum learning[98], layer-wise pre-training [54, 9]. Deep learning has acquired wide success in many challenging applications, such as object recognition, natural language processing and protein structure prediction [68, 95, 30].

Given the increased availability of high-throughput microarray data, and the frequent observation of nonlinear interactions among human genes, we conceive that a deep neural network is a promising candidate to tackle the gene expression inference problem. Motivated by this, we propose GEIDN (*Gene Expression Inference via Deep Neural Networks*), to computationally infer target genes from landmark genes. GEIDN is a multi-output multi-layer neural network equipped with several advanced deep learning techniques, specifically, drop out, momentum training and GPU computing. For evaluation purposes, we also investigate a few other representative machine learning models, including linear regression (with and without the ℓ_1-norm regularization) and k-nearest neighbor regression. We do a comparative study on a Gene Expression Omnibus dataset (GEO, http://www.ncbi.nlm.nih.gov/geo/) accessed from the LINCS Cloud (http://www.lincscloud.org/), which consists of ˜129k samples. Experimental results demonstrate that GEIDN systematically outperforms other methods in terms of prediction accuracy. Moreover, increased abundance of data benefits GEIDN more than other methods; moderate expansion of the architecture also helps to improve GEIDN's performance. These results suggest potential advantages of applying large-scale deep neural networks to gene expression inference. Lastly, we infer target genes of the L1000 dataset with GEIDN, which can be used by biologists for further study.

The outline of this chapter is as follows. Section 5.2 formulates the gene expression inference problem mathematically, introduces major predictive models, and proposes the GEIDN framework. Section 5.3 evaluates the inference results of GEIDN and other models on both the GEO dataset and the L1000 dataset. Section 5.4 summarizes the finished work, discusses the limitations of GEIDN and future directions.

5.2 Methods

In this section, we formulate gene expression inference as a machine learning problem. We will first introduce the basic notations, then introduce several important predictive models that can be applied to this problem, and lastly propose GEIDN, explaining a few key deep learning techniques we deployed to train GEIDN and implementation issues. First, assume that there are L landmark genes, M target genes, and N training samples; the dataset is expressed as $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathcal{R}^L$ denotes the expression of landmark genes and $\mathbf{y}_i \in \mathcal{R}^M$ denotes the expression of target genes in the i-th sample. Our goal is to infer the functional mapping $\mathcal{F} : \mathcal{R}^L \to \mathcal{R}^M$ that fits $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, which is essentially a multi-task regression problem.

$\forall$ testing dataset $\{\mathbf{x}_j, \mathbf{y}_j\}_{j=1}^{N'}$, suppose the prediction from $\mathcal{F}$ is $\{\hat{\mathbf{y}}_j\}_{j=1}^{N'}$. We use the Rooted Mean Squared Error (RMSE) to evaluate the predictive performance at each target gene t,

$$RMSE^{(t)} = \sqrt{\frac{1}{N'}\sum_{j=1}^{N'}\left(y_j^{(t)} - \hat{y}_j^{(t)}\right)^2}. \tag{5.1}$$

We also use the Rooted Mean Relative Squared Error (RMRSE) as an alternative evaluation metric,

$$RMRSE^{(t)} = \sqrt{\frac{1}{N'}\sum_{j=1}^{N'}\left(\frac{y_j^{(t)} - \hat{y}_j^{(t)}}{y_j^{(t)}}\right)^2}. \tag{5.2}$$

This is motivated by the multi-task nature of the problem. As tasks (target genes) often have different expected values (expression levels), it is reasonable to gauge the predictive performance on each task using a common scale.

5.2.1 Linear Regression

In linear regression (LR), $\mathcal{F}(\mathbf{x}) = \left\{\mathbf{w}^{(m)T}\mathbf{x} + b^{(m)}\right\}_{m=1}^{M}$ is a set of independently trained models, where $\mathbf{w}^{(m)} \in \mathcal{R}^L, b^{(m)} \in \mathcal{R}$, and

$$\left(\mathbf{w}^{(m)}, b^{(m)}\right) = \arg\min_{\mathbf{w},b} \frac{1}{N} \sum_{i=1}^{N} \left(y_i^{(m)} - \mathbf{w}^T\mathbf{x}_i - b\right)^2. \tag{5.3}$$

The ℓ_1-norm regularization can be further introduced to enforce sparse solution of $\mathbf{w}$ for the variable selection purpose. In this case,

$$\left(\mathbf{w}^{(m)}, b^{(m)}\right) = \arg\min_{\mathbf{w},b} \frac{1}{N} \sum_{i=1}^{N} \left(y_i^{(m)} - \mathbf{w}^T\mathbf{x}_i - b\right)^2 + \lambda^{(m)}\|\mathbf{w}\|_1. \tag{5.4}$$

LR (5.3) is currently adopted by the LINCS project. In our study, we explore both (5.3) and (5.4) (denoted as LR-L1). They are conveniently deployed from *scikit-learn*, a widely adopted machine learning software package [80] (`http://scikit-learn.org/stable/modules/linear_model.html`).

5.2.2 k-Nearest Neighbors Regression

The classical k-nearest neighbor (KNN) regression has the following procedure: First, a spatial data structure $\mathcal{T}$ such as the KD tree [10] is built for training data in their feature space; then for any new data, the k nearest training samples are queried from $\mathcal{T}$, and the average (or weighted sum) of their target values is computed as the prediction. However, this strategy can be biased when duplicate features frequently exist in the data, which is often the case for microarray data. Therefore, in gene expression study, a commonly adopted alternative is to exchange the role of genes and samples. Specifically to our problem, $\mathcal{T}$ is built for landmark genes in the space spanned by the training data; then for each target gene,

the k landmark genes with the closest training sample values are queried, and the average (or weighted sum) of their testing sample values is computed as the prediction. We call it the *gene-based* KNN (KNN-GE). Note that each gene is scaled by the mean and standard deviation of its training portion before both training and querying, which helps to make genes at different expression levels numerically comparable; the KNN predictions are scaled back as the final predictions.

Due to the non-parametric and distance-based nature, KNN doesn't impose any prior assumption on the learning machine. Therefore, it is very flexible to model nonlinear functions. However, as performing prediction involves building and querying spatial data structures, which has to keep all the training data, KNN suffers from the poor scalability to growing data size and dimension. We evaluated KNN-GE in our study. KNN is also conveniently deployed from scikit-learn (`http://scikit-learn.org/stable/modules/neighbors.html`).

5.2.3 GEIDN

GEIDN is a large-scale multi-output multi-layer neural network (NN). There are 978 input neurons, which correspond to the 978 landmark genes, 2 or 3 hidden layers, each of which has 3000 ~8000 neurons, and 5322 (or 5323) output neurons, which correspond to 1/4 of the 21,290 target genes. To be able to predict all target genes, we build 4 separate GEIDNs, each of which predicts a different 1/4 random subset of target genes. Training tasks are assigned to UCI HPC's GPU server, which has 4 Nvidia Tesla M2090 graphical cards (http://hpc.oit.uci.edu/gpu). We did not train a network that jointly predicts all 21,290 genes, because larger networks require more CUDA cores and GPU memory to train in a reasonable time. However, we note that this problem could be largely resolved, given more powerful GPUs (e.g., Tesla K40, GTX TITAN Z), and network-distributed training techniques on multiple GPUs such as those described in [21, 64, 22].

We adopt the hyperbolic tangent (TANH) activation function for hidden units. During learning, the hidden layers naturally represent the input data as a series of hierarchical nonlinear features, which encode the highly coupled interactions between landmark genes and target genes. We apply the linear activation function to output units for the regression purpose. The loss function for training is the sum of mean squared error at each output unit, namely,

$$\mathcal{L} = \frac{1}{N} \sum_{m=1}^{M} \sum_{i=1}^{N} \left(y_i^{(m)} - \hat{y}_i^{(m)} \right)^2 . \tag{5.5}$$

Training GEIDN fundamentally follows the framework of *error gradient back-propagation* and *mini-batch gradient descent*, and is supplemented with advanced deep learning techniques. See [67, 8] for systematic descriptions. Supplementary Table S1 introduces three GEIDN architectures that we have explored, along with detailed parameter configurations, which are tuned according to a validation dataset. We discuss a few key points as follows:

1. *Dropout* is a technique to perform model averaging and to prevent overfitting. For each training sample, the neurons of each layer are randomly dropped out with some probability $1 - p$, so the forward- and back-propagation are performed on a particularly "thinned" network. For an architecture with n non-input neurons, there are $O\left(\frac{1}{p^n}\right)$ such thinned networks. Therefore, dropout can achieve model averaging of exponentially many different neural networks in an approximate but efficient framework. Randomly dropping out neurons also helps to suppress *co-adaptation* among neurons [97]. In GEIDN, p is set to $[0.85, 0.9]$ for all neurons except the input and output neurons. Dropping out the input neurons is essentially introducing "missing-value" noise to the data, which empirically is not necessary for our problem.

2. *Momentum training* is a technique to acceleration gradient-based learning. Denote the

weights of the NN as $\mathbf{W}$, associated with velocity $\mathbf{V}$. Then at each training iteration $t+1$,

$$\begin{aligned}\mathbf{V}^{(t+1)} &= \mu^{(t+1)}\mathbf{V}^{(t)} - (1-\mu^{(t+1)})\eta^{(t+1)}\nabla\mathcal{L}\left(\mathbf{W}^{(t)}\right)\\ \mathbf{W}^{(t+1)} &= \mathbf{W}^{(t)} + \mathbf{V}^{(t+1)},\end{aligned} \tag{5.6}$$

where $\mu \in [0,1]$ is the momentum coefficient, η is the learning rate, and $\nabla\mathcal{L}(\mathbf{W})$ is the error gradient of the loss function $\mathcal{L}$ with respect to $\mathbf{W}$. μ usually starts small but increases to the maximum over epochs. The motivation is to follow the gradient direction initially, but later on emphasize more and more on the velocity direction so as to accelerate learning.

3. *Learning rate* is initialized to $1e-3$ (or $5e-4$), and annealed over iterations with the following schedule,

$$\eta^{(t)} = \frac{\eta_0}{(1+\delta)^t}, \tag{5.7}$$

where δ is a small positive value, e.g., $1e-6$.

4. *Weights* are initialized by sampling from a uniform distribution defined by,

$$\mathbf{W}^{(0)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_i+n_o}}, \frac{\sqrt{6}}{\sqrt{n_i+n_o}}\right], \tag{5.8}$$

where n_i, n_o denote the number of fan-ins and fan-outs of the neuron which has the incoming nerve of that particular weight. This scheme is designed to stabilize the activation variance and the gradient variance as information propagates up and down the network [43]. For very large architectures, however, the uniform distribution of the output layer is set to be within a even smaller range (e.g., $\pm 1e-4$, see GEIDN-3 in Table S1). Empirically, (5.8) tends to generate large initial gradients that quickly saturate the hidden neurons in larger networks.

5. *Data processing* One standard way to pre-process neural network training data is to do standardization, i.e., for each dimension (input or output), first subtracting the mean and

then dividing by the standard deviation. The goal is to prevent neurons from saturating, by first centering data around zero and then rescaling them in a reasonable range. However, (1) as we are doing regression, standardizing will break the original scale relationship of output channels during training; (2) as many genes' expressions are highly skewed with little variance, which is often due to random noise, standardization will amplify the noise of these genes to be comparable to the variance of other genes. Since both effects are not desired, we propose an adjusted scheme to process the training data: first, subtract each dimension by its mean value; second, divide the whole training data by its global standard deviation. During prediction, each output channel is first multiplied by the global scale, and then added back with the training mean value of that channel.

6. *Model selection.* A separate validation dataset (detailed in next section) is used to tune model and algorithm parameters. Training is run for a preset number of epochs. However, the model is evaluated after each epoch on the validation dataset. Each time a new best performance (a lowest loss computed by (5.5)) is reached, the model is saved as the optimal model. After training finishes, the saved model is used for prediction.

GEIDN is implemented based on two Python libraries, *Theano* [11] and *Pylearn2* [46], both of which are developed by the LISA Lab at the University of Montreal. Theano is a symbolic computation library, which comprehensively support definition, evaluation and optimization of mathematical expressions. In particular, the back-propagation algorithm is achieved via symbolic differentiation upon the *Theano graph*. By direction manipulation of GPU, Theano can perform multi-dimensional tensor computation much faster than using CPU. Theano is publicly available at https://github.com/Theano/Theano. Pylearn2 is a neural network & deep learning library built on top of Theano. It supports a wide range of network architectures and learning algorithms. We adapt some of its modules to support the training of GEIDN. It is freely available at https://github.com/uci-cbcl/pylearn2.

5.3 Results

5.3.1 GEO Data and L1000 Data

The GEO dataset is generated from the Affymetrix platform. It contains 129,158 samples, each of which has 22,268 fully measured microarray probes, corresponding to the 978 landmark genes and the 21,290 target genes. The L1000 dataset is generated from the Luminex bead-based platform. Since the L1000 platform only measures the landmark genes, it can generate very high-throughput expression profiles. Currently, the dataset contains 1,328,098 samples from cultured cells treated with different chemical and genetic perturbations. Its "Level 3" representation has both directly measured landmark genes and linear-regression-imputed target genes. The GEO dataset and the L1000 Level 3 dataset are roughly on the same numerical scale (between 4.0 and 15.0, although the later has a small portion ranging between 0.0 and 4.0), and are the focus of the gene expression inference problem. Both datasets are originally accessed through the LINCS Cloud in a binary format called "gctx", whose size are ˜11G and ˜111G separately. They can be queried and manipulated on disk by the *L1000 Analysis Tools* (https://github.com/cmap/l1ktools), which works on top of Numpy, PyTables and HDF5. For more information about the data, see the LINCS Cloud.

To facilitate experiments and analysis, we transform the whole GEO dataset and a random subset of 10,000 samples of the L1000 dataset (denoted as L1k-sub) into Numpy array data files. Supplementary Figure S1 shows the marginal distribution of expression levels of landmark genes and target genes of GEO and L1k-sub. GEO is further randomly partitioned into ˜80% for training (104,000), ˜10% for validation (12,579) and ˜10% for testing (12,579). They are denoted as GEO-tr, GEO-va, GEO-te. Specifically, the validation data is used to do model selection and parameter tuning, such as λ (of LR-L1), k (of KNN-GE), and p, u, η, etc. (of GEIDN).

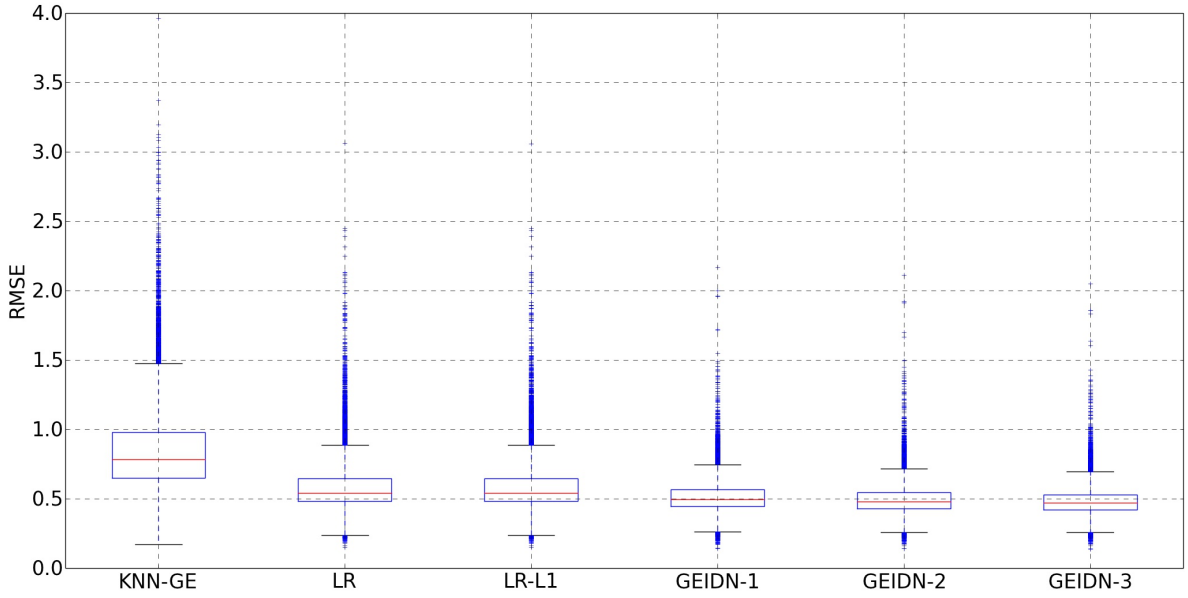
5.3.2 Main Results on the Complete GEO Dataset

LR, LR-L1, KNN-GE and three GEIDNs of different configurations (detailed in Supplementary Table S1) are trained using the GEO-tr data; free parameters (if any) are chosen by the GEO-va data; prediction accuracy is evaluated using the GEO-te data. The complete predictive performances are described in Table 5.1 and visualized in Figure 5.1, from which we can get a few key observations. First, KNN-GE seems to provide the baseline performance. Second, LR has notable improvement over KNN-SP, but the sparse regularization does not provide further improvement. Third, GEIDNs systematically outperform other approaches. Fourth, the two evaluation metrics, RMSE and RMRSE, provide fairly consistent conclusions. Furthermore, an internal comparison of GEIDN-1, 2 and 3 shows that a larger-scale architecture does help to improve GEIDN. In general, GEIDN-3 has **˜40.90%** relative improvement over KNN-GE (43.30% in RMSE and 38.49% in RMRSE), and **˜16.62%** relative improvement over LR (17.31% in RMSE and 15.94% in RMRSE).

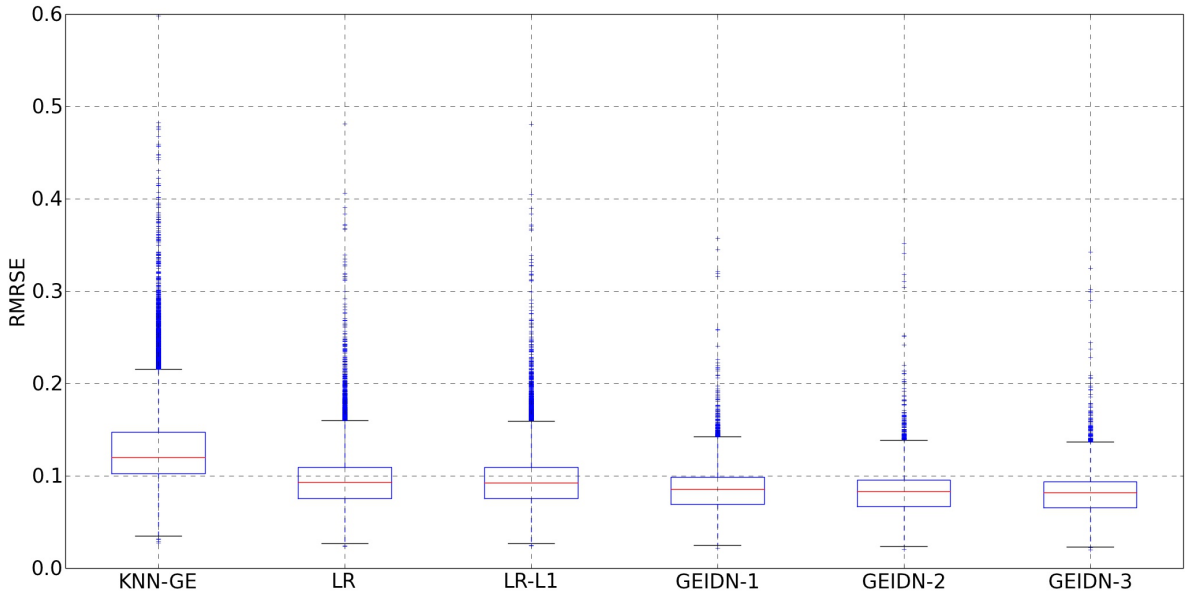
Table 5.1: The average prediction error over all target genes for predictive models discussed in Section 5.3.2. GEO-tr, GEO-va, and GEO-te are used for model training, validation, and testing. Numerical values after "±" are the standard deviations. The bold font highlights the best performances.

	RMSE	RMRSE
KNN-GE	0.8602±0.3221	0.1312±0.0451
LR	0.5898±0.1820	0.0960±0.0297
LR-L1	0.5897±	0.0959±0.0297
GEIDN-1	0.5195±0.1216	0.0851±0.0218
GEIDN-2	0.4997±0.1140	0.0822±0.0211
GEIDN-3	**0.4877±0.1044**	**0.0807±0.0204**

Figure 5.2 provides a more detailed comparative analysis of three representative models, LR, KNN-GE, and GEIDN-3. It demonstrates that GEIDN-3 consistently performs better than LR and KNN-GE at $> \mathbf{99.96}\%$ of the target genes. Also see Supplementary Figures S2 and S3 for the comparative analysis of GEIDN-1, -2 to LR and KNN-GE, which show



(a)



(b)

Figure 5.1: The prediction error box plot over all target genes for predictive models discussed in Section 5.3.2. GEO-tr, GEO-va, and GEO-te are used for model training, validation, and testing. (a) Using RMSE as the evaluation metric; (b) using RMRSE as the evaluation metric.

similar patterns to that of Figure 5.2. As transcriptional programs have grouping effects, joint prediction of target genes using shared nonlinearity might be preferred in the biological sense. Our results seem to substantiate this conjecture, because only the GEIDN series do joint nonlinear prediction and have the best prediction accuracy. All of these demonstrate the potential utility of deep neural networks in accurately and effectively modeling massive nonlinear interaction among human genes.

Figure 5.3 visualizes the learning curves of GEIDN-3, which traces the loss function's value (5.5) over training epochs. Also see Supplementary Figures S4 and S5 for those of GEIDN-1 and -2. Note again that the 21,290 target genes are evenly distributed into four networks for training and prediction. Accordingly, each GEIDN architecture has four sets of learning curves. A key observation is that overfitting is effectively controlled, even when training such a large-scale architecture as GEIDN-3. We think this is attributed to the deep learning techniques described in Section 5.2.3 and Supplementary Table S1, in particular, the drop out mechanism.

5.3.3 The Effect of Data Size

Section 5.3.2 has demonstrated that GEIDN provides more accurate prediction than other methods with a moderate large training dataset as GEO-tr. To explore the effects of data size to different learning machines, we further subsample 8,000 from GEO-tr (denoted as GEO-tr-sub) and 1,000 from GEO-va (denoted as GEO-va-sub), and use them to train LR, LR-L1, KNN-GE, and GEIDN-1 again. The learned models are still evaluated on GEO-te. The predictive performances are summarized in Table 5.2 and visualized in Supplementary Figure S6. Comparing them to Table 5.1 and Figure 5.1, we can see that increased training data size systematically improve predictive accuracy of all the four methods, while the improvement to GEIDN is more notable than others; when data is scarce, GEIDN still per-

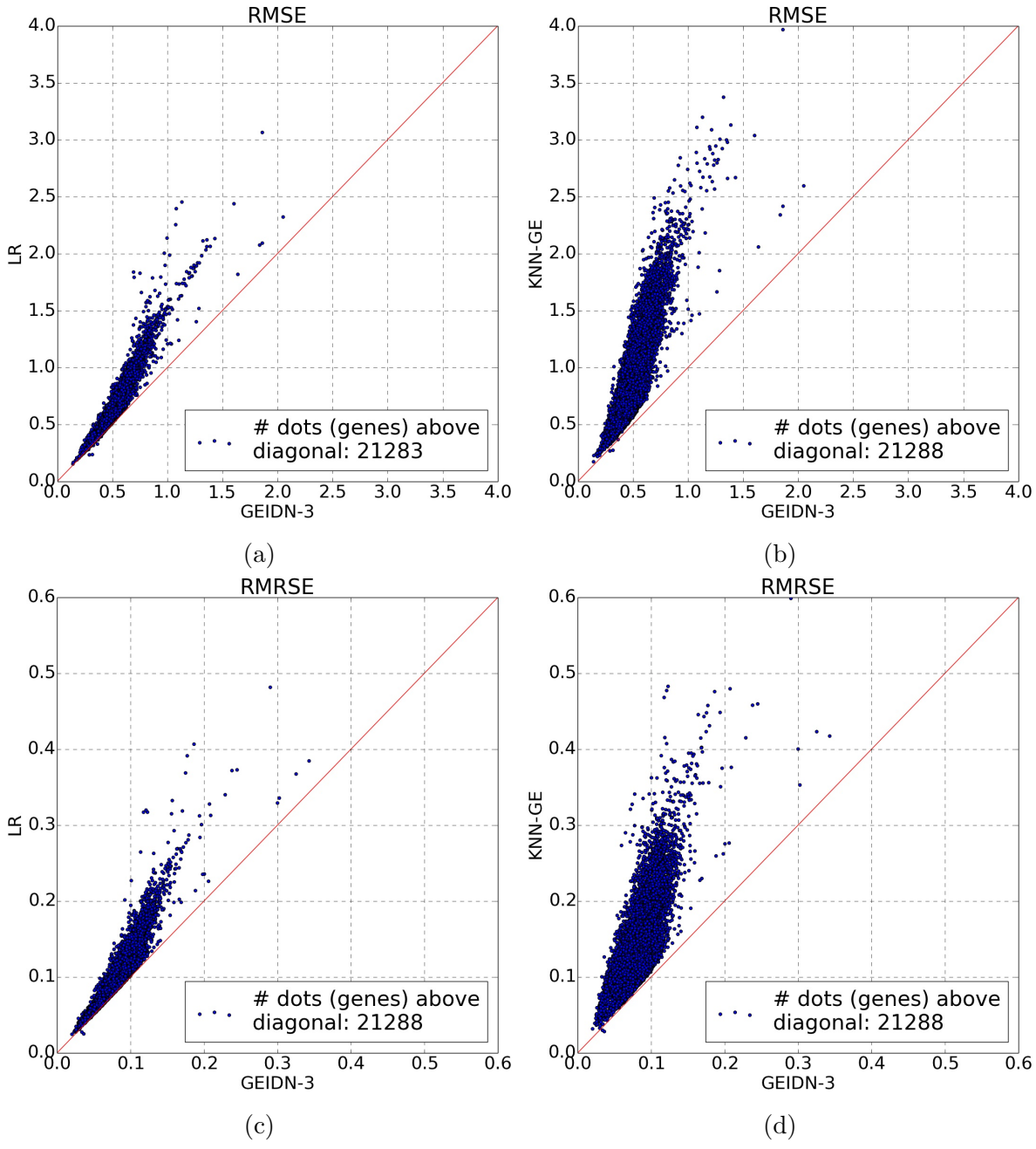


Figure 5.2: The predictive errors of GEIDN-3 compared to LR and KNN-GE. Out of the 21,290 target genes, GEIDN-3 performs better than LR in 21,283 genes (**99.97%**), better than KNN-GE in 21,288 genes (**99.99%**), in terms of RMSE; statistics are similar in terms of RMRSE. (a) RMSEs of GEIDN-3 versus those of LR; (b) RMSEs of GEIDN-3 versus those of KNN-GE; (c) RMRSEs of GEIDN-3 versus those of LR; (d) RMRSEs of GEIDN-3 versus those of KNN-GE.

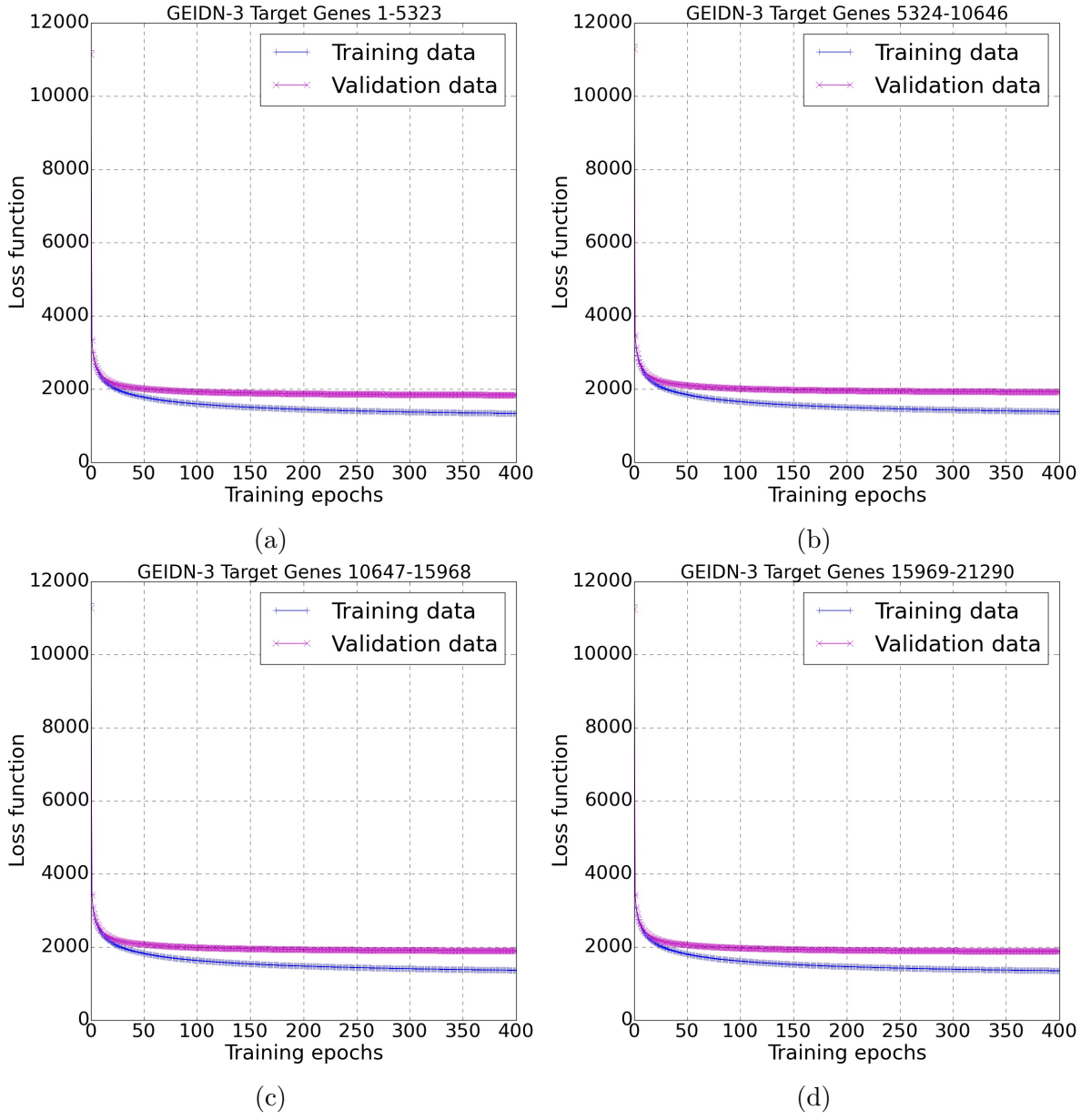


Figure 5.3: Learning curves of GEIDN-3. The target genes are randomly permuted and then assigned with labels 1~21,290. After that, they are evenly distributed into four neural networks for training and prediction, whose learning curves are shown by (a), (b), (c), and (d) separately. See Section 5.2.3 for the motivation of this design.

forms the best, but the advantage margins are less significant. These observations suggest that abundant data resource is a key element for deep learning to success, in particular in capturing the complicate nonlinear effects within the data, which is not easily achievable by traditional shallow models. Interestingly, we notice that LR-L1 performs notably better than LR in this scenario. A further investigation of the validation results shows that the sparse regularizations λ are typically stronger than that in Section 5.3.2. This result corroborates the utility of model regularization when the training data size is small.

Table 5.2: The average prediction error over all target genes for LR, LR-L1, KNN-GE and GEIDN-1. GEO-tr-sub, GEO-va-sub, and GEO-te are used for model training, validation, and testing. Numerical values after "±" are the standard deviations. The bold font highlights the best performances.

	RMSE	RMRSE
KNN-GE	0.9051±0.3482	0.1382±0.0476
LR	0.6433±0.1962	0.1054±0.0328
LR-L1	0.6256±0.1919	0.1013±0.0309
GEIDN-1	**0.5928±0.1651**	**0.0968±0.0276**

5.3.4 Inference on the L1000 Data

The LINCS project currently trains linear regression models on a smaller GEO dataset, and uses them to infer the unmeasured target genes of the L1000 data. Following this, we select three representative models trained in Section 5.3.2, LR, KNN-GE and GEIDN-3, to do inference on the L1k-sub data. The prediction summary statistics are shown in Figure 5.4. From 5.4a, 5.4c and 5.4e, it seems that LR's inference has quite many outliers, KNN's inference range is more controlled, and GEIDN-3's inference fairly consistently falls within [0,15], which happens to be the numerical range of the landmark genes. From 5.4b, 5.4d, and 5.4f, it seems the prediction variances of GEIDN-3 and KNN-GE are smaller than those of LR. As both GEIDN-3 and KNN-GE are nonlinear approaches and involve some linear pre-processing of the data, we think that designing different pre-processing schemes

may potentially alter the inference statistics. Above all, since the ground truth of target genes is unobserved, how to verify the inference quality of different methods remains an open question. Nevertheless, given the predictive performance on the fully-observed GEO dataset, we believe GEIDN's inference results on the L1000 data is a good candidate for further study.

5.4 Discussion

Inference of human gene expressions has become an important problem in computational biology, as it implies a notable reduction in both time and financial cost of human genome profiling. The current linear-regression-based approach cannot model massive nonlinear interactions among genes. We have developed GEIDN, a large-scale neural network, to computationally infer the ˜21k target genes from the ˜1k landmark genes in a join framework. With the help of modern deep learning techniques, such as dropout, momentum, and GPU computing, GEIDN is able to model massive nonlinear interactions between landmark genes and target genes. Using a GEO dataset with complete measurement of gene expression profiles of ˜129k samples, we demonstrate that GEIDN systematically outperforms two linear regression modes and one k-nearest neighbor model. We also find that expanded network scales help to improve GEIDN's performance. Moreover, increased abundance of data benefits GEIDN more than other approaches. As genomic data are increasingly available, GEIDN is likely to play a more important role in gene expression inference.

Currently, target genes are distributed to four GEIDN networks for prediction due to hardware limitations. Accordingly, a direct improvement is to enlarge the scale of GEIDN such that it can jointly predict all target genes, either by using better GPUs, or by exploiting multi-GPU technics such as those discussed in Section 5.2.3. This seems promising based upon the performance of the current architectures.

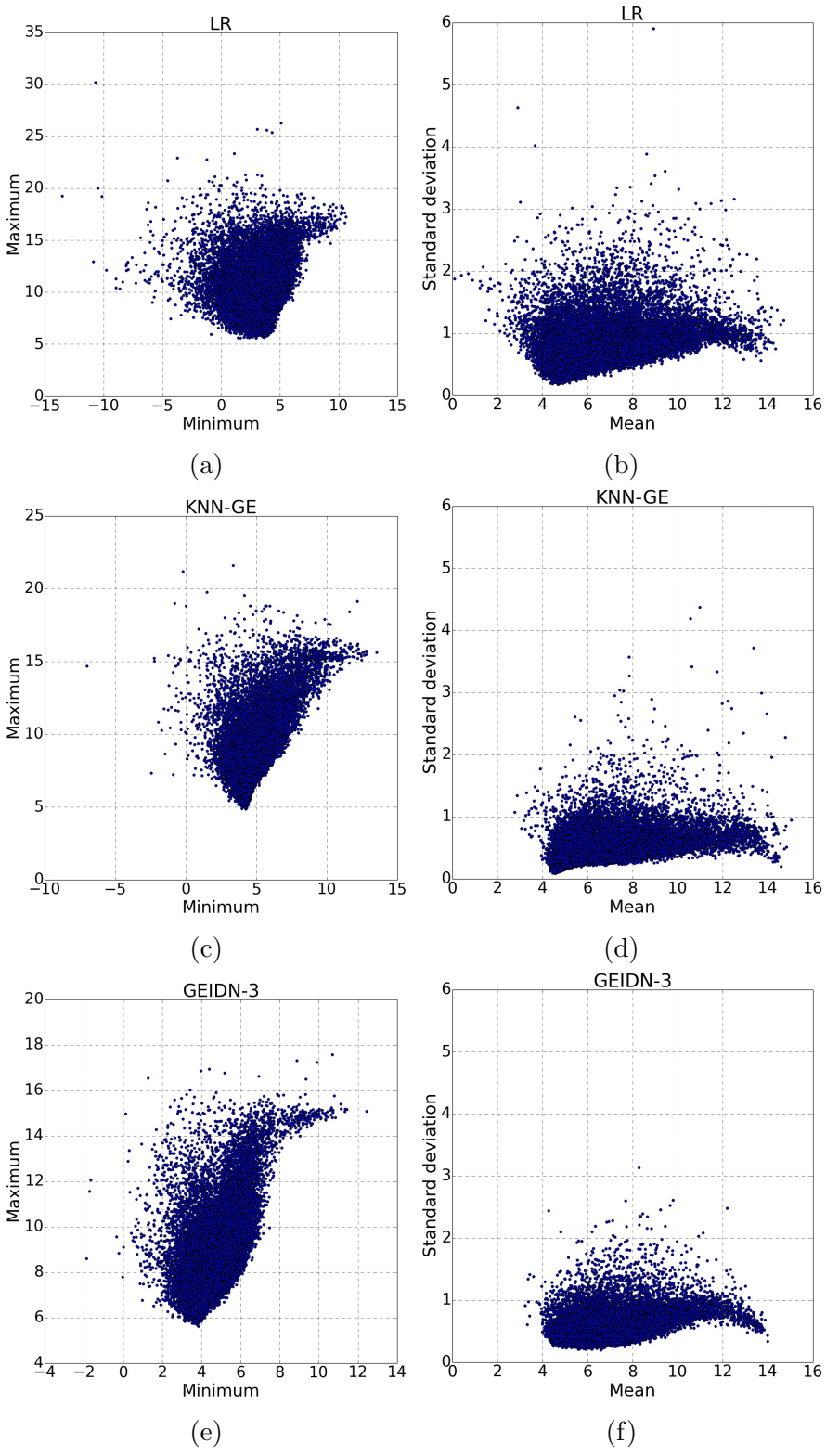


Figure 5.4: Inference summary statistics of LR, KNN-GE and GEIDN-3 on L1k-sub. (a)(c)(e) Minimum and maximum predictions at each target gene; (b)(d)(f) mean and standard deviation of predictions at each target gene.

There are several open questions as to how to interpret the learned network: What features does each hidden layer represent? Do the learned weights (upon convergence) have any biological meaning? How is the network related to the gene regulatory network as known by biologists? To address these questions, one needs to investigate the genes associated with each neuron, check their weight patterns (scale, sparsity, etc.), and analyze them using the domain knowledge. Unsupervised feature learning techniques such as the *auto encoder* can be deployed to assist this process [102]. In one word, to interpret learned network in the context of domain knowledge is a reverse engineering study. Otherwise, it is also possible to design network architectures, which encapsulate the prior knowledge of gene regulatory networks & pathways. Training such models might provide more informative features to biologists, just like convolutional neural networks to computer vision researchers [68].

Finally, one important goal of the LINCS L1000 project is to infer the unobserved targets genes from the measured landmark genes. But how to evaluate the inference quality on the L1000 data remains not clearly defined. A proper evaluation of GEIDN on L1000 data may be possible with a matched, representative gold standard dataset generated by a genome-wide assay such as the Affymetrix microarray platform.

Chapter 6

Deep Learning for Genetic Variants Annotation

6.1 Introduction

Identifying the genetic variants responsible for diseases can be very challenging. The majority of candidate variants lie in noncoding sections of the genome, whose role in maintaining normal genome function is not well understood. Most annotation methods can only annotate protein coding variants, excluding >98% of the human genome. Another annotation method, Combined Annotation–Dependent Depletion (CADD) [63], can annotate both coding and noncoding variants. CADD trains a linear kernel SVM to separate observed genetic variants from simulated genetic variants. Observed genetic variants are derived from differences between human genomes and the inferred human-chimpanzee ancestral genome. Because of natural selection effects, observed variants are depleted of deleterious variants. Simulated genetic variants are enriched for deleterious variants.

CADD's SVM can only learn linear representations of the data, which limits its performance.

To overcome this, we implemented a DNN algorithm that we have named DANN (**D**eleterious **A**nnotation of genetic variants using **N**eural **N**etworks). A DNN is an artificial neural network with several hidden layers of units between the input and output layers. The extra layers give a DNN added levels of abstraction, but can greatly increase the computational time needed for training. Deep learning techniques and GPU hardware can significantly reduce the computational time needed to train DNNs. DNNs outperform simpler linear approaches such as logistic regression (LR) and SVMs for classification problems involving many features and samples.

6.2 Methods

6.2.1 Model Training

DANN trains a DNN consisting of an input layer, a sigmoid function output layer, and three 1000-node hidden layers with hyperbolic tangent activation function. We use deepnet (https://github.com/nitishsrivastava/deepnet) to exploit fast CUDA parallelized GPU programming on an NVIDIA Tesla M2090 card and apply dropout and momentum training to minimize the cross entropy loss function. Dropout reduces overfitting by randomly dropping nodes from the DNN [96]. Momentum training adjusts the parameter increment as a function of the gradient and learning rate [98]. DANN uses a hidden node dropout rate of 0.1, a momentum rate that increases from 0.01 to 0.99 linearly for the first 10 epochs and then remains at 0.99, and stochastic gradient descent (SGD) with a minibatch size of 100. As a baseline comparison, we trained a LR model. For LR training, we applied SGD using the scikit-learn library [80] with parameter $\alpha = 0.01$, which we found to maximize the accuracy of the LR model. LR and DNN are sensitive to feature scaling, so we preprocess the features to have unit variance before training either model. We also train an SVM using

the LIBOCAS v0.97 library [35] with parameter $C = 0.0025$, closely replicating CADD's training.

6.2.2 Features

There are a total of 949 features defined for each variant. The feature set is sparse, and includes a mix of real valued numbers, integers, and binary values. For example, amino acid identities are only defined for coding variants. To account for this, we include Boolean features that indicate whether a given feature is undefined, and missing values are imputed. Moreover, all n-level categorical values, such as reference allele identity, are converted to n individual Boolean flags. See the Supplementary of [63] for more details about the features and imputation.

6.2.3 Training Data

CADD's training data consist of 16,627,775 "observed" variants and 49,407,057 "simulated" variants. We trained all three models on this dataset to differentiate the simulated variants from the observed variants. To account for the imbalance between the two datasets, we randomly sampled 16,627,775 simulated variants for training. These 33,255,550 variants are split into a "training set", a "validation set", and a "testing set" in an approximately 8:1:1 ratio. The three models are trained on the training set. For SGD, each gradient step is not guaranteed to minimize the loss function; at 1/10 epoch intervals throughout the 20 epochs of training the validation set is evaluated in order to select the "best" model that maximizes classification accuracy on the validation set. The validation set is also used to fine tune hyperparameters such as dropout rate, minibatch size, etc. Finally, the models are regularly evaluated on the testing set to monitor for overfitting. In contrast, [63] trained CADD using an "ensemble" strategy that involves training SVMs on ten different subsets of the training

data. We found little performance improvement when we applied this strategy.

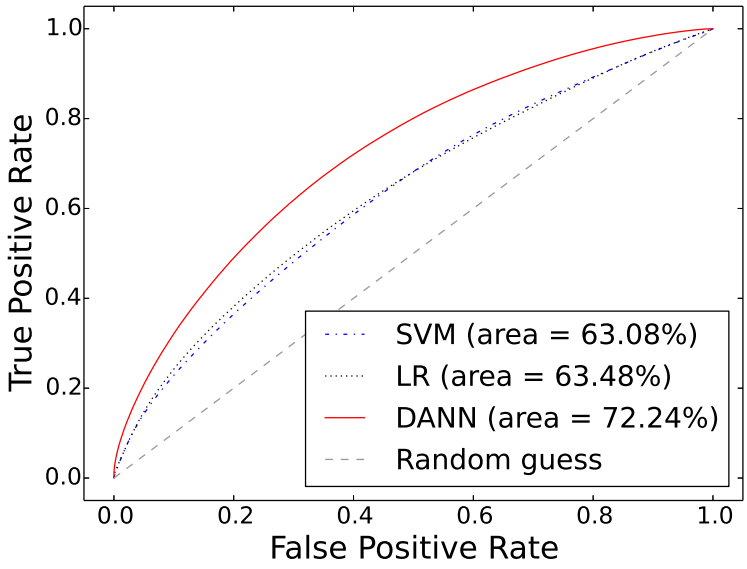
6.3 Results and Discussion

To compare the performance of the three models, we generated receiver operating characteristic (ROC) curves discriminating the 3,326,573 simulated and observed variants in the testing set and calculated AUC scores (Fig. 6.1a). We used the discriminant values of the SVM and the sigmoidal function output of the DNN and LR models as classifiers for the ROC curves. We do not directly compare to CADD because it can only evaluate 100,000 variants at a time and CADD was already trained on testing set variants; however, the SVM we trained performs very similarly to CADD despite being trained on a smaller dataset (data not shown). The classification accuracies of the SVM, LR, and DNN models are 58.2%, 59.8%, and 66.1%, respectively. A few observations emerge from our analysis. First, LR performs better than SVM, suggesting that the max margin regularization used by SVM plays little role in this particular dataset. Second, DNN performs significantly better than both LR and SVM, leading to a 18.90% reduction in the error rate and a 14.52% improvement in the AUC relative to SVM. This suggests the importance of accounting for nonlinear relationships among features, likely due to the heterogeneity of features generated in genome annotations. Third, although DNN improves on the linear methods, its accuracy is still unsatisfactory. We suspect a few factors might contribute to this: 1) The training data are inflated with mislabeled samples. Observed variants can be under positive or weak purifying selection, and therefore be functional. Conversely, many simulated variants can be nonfunctional since they are randomly sampled from the genome. 2) The features currently used in genome annotation are insufficient for functional prediction. 3) The model training needs further improvement.

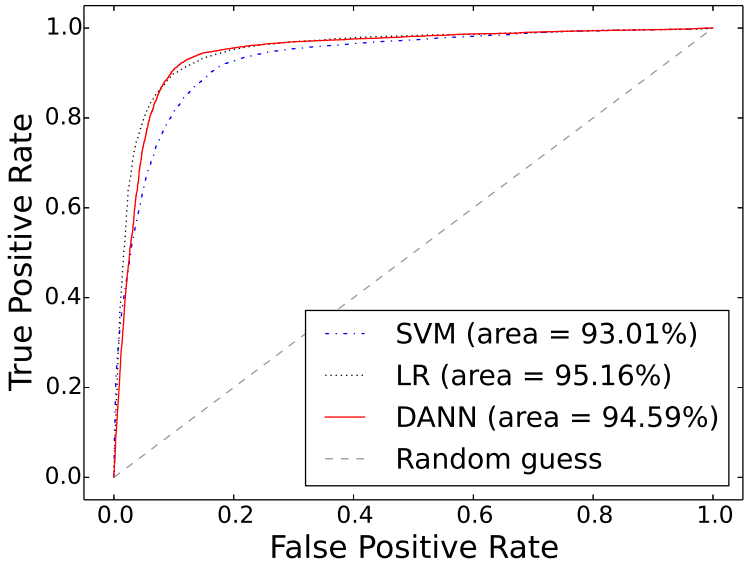
We also generated ROC curves showing the models discriminating pathogenic mutations de-

fined by the ClinVar database [4] from likely benign Exome Sequencing Project (ESP) [41] alleles with a derived allele frequency (DAF) $\geq 5\%$ (Fig. 6.1b, n = 10,000 pathogenic/10,000 likely benign). Coding variants constitute 85.6% and 43.0% of the ClinVar and ESP databases, respectively, reducing the difficulty of annotation since many more informative features are available in coding regions. For variants with multiple gene annotations, we only selected the gene annotation that yielded the highest score from each model. All three models greatly improve on the AUC metric, with the LR and DANN models outperforming SVM; however, the performance gap between the models is much smaller than the gap in the testing set.

In conclusion, we have improved considerably upon CADD's SVM methodology. We can even achieve better performance with LR, suggesting that LR is the preferred linear classifier in this case. DANN achieves the best overall performance, substantially improving upon the linear methods in terms of accuracy and separation on the testing set, which contains mostly noncoding variants. This makes DANN the most useful annotation algorithm since the vast majority of human variation is noncoding. When limited to a coding biased dataset, all three models perform well, but the performance gap is small. Given DANN's superior performance for annotating noncoding variants, which comprise the overwhelming majority of genetic variation, we expect DANN to play an important role in prioritizing putative causal variants, such as those derived from GWAS, for further downstream analysis.



(a) Testing set



(b) ClinVar and ESP

Figure 6.1: ROC curves comparing performances of the neural network (DANN), support vector machine (SVM), and logistic regression (LR) models in discriminating (a) "simulated" variants from "observed" variants in the testing set and (b) pathogenic ClinVar variants from likely benign ESP alleles (DAF $\geq 5\%$).

Chapter 7

Conclusion

Genomic and transcriptomic activities are the fundamental controllers of the living process of any organism. Their abnormal behaviors are believed to be the driving factors of many diseases. Therefore, understanding the intrinsic mechanisms of the genome and the transcriptome and informing clinical practices have become two important missions of large-scale genomic researchers. In the recent decade, high-throughput molecular data have provided abundant information about the whole genome; and it has become an important research direction to study genomics from these high-throughput data by deploying computational tools. However, traditional computational learning methodologies often have strong limitations when dealing with high-throughput genomic datasets, because the latter are usually very high dimensional, highly heterogeneous, and can show complicated nonlinear effects given a large amount of data. In this thesis, we have presented five new algorithms or models, each of which address previous challenges from a certain perspective and is applicable to a specific genomic problem:

1. ADMM-ENSVM. ENSVM can achieve simultaneous variable selection and max-margin classification, but is difficult to solve at large scale because of multiple nondifferentiable

terms in the objective function. ADMM-ENSVM decomposes ENSVM into smaller sub-problems, and solves it efficiently. On a colon cancer diagnosis study, ADMM-ENSVM shows the advantage over SVM, L1-norm SVM and HHSVM in terms of diagnose accuracy, feature selection ability, and computational efficiency.

2. SBLVGG. LVGG interprets the marginal concentration matrix of observed variables as a combination of a sparse matrix and a low rank matrix. SBLVGG decomposes LVGG into smaller sub-problems, solves it efficiently, and is notably faster than the state-of-the-art algorithms. Evaluated on a microarray dataset containing thousands of genes, SBLVGG shows that most of the correlation among genes can be explained by tens of latent factors. This provides a new interpretation of the underlying structure of gene regulatory networks.

3. GBMCI. Survival analysis models often impose strong assumptions on hazard functions. GBMCI does not explicitly assume particular forms of hazard functions; instead, it trains an ensemble of regression trees to approximately optimize the concordance index via gradient boosting. We benchmark the performance of GBMCI against several popular survival models with a large-scale breast cancer prognosis task. GBMCI consistently outperforms other methods based on a number of feature representations, which are heterogeneous and contain missing values.

4. GEIDN. Genome-wide microarrays are prohibitively expensive. However, genes' expressions are strongly correlated. The LINCS project has developed the L1000 microarray to measure the expression of ˜1k landmark genes, and uses linear regression to infer the expression of ˜21k target genes separately. However, genes have intrinsic nonlinear interactions, and transcriptional programs are often grouped into modules. GEIDN is a large-scale neural network, which can infer target genes jointly and naturally capture hierarchical nonlinear features shared among target genes. We deploy advanced deep learning techniques to train GEIDN, such as drop out, momentum training, and

GPU computing. On a GEO dataset of ˜129k complete transcriptomes, GEIDN outperforms both k-nearest neighbor regression and linear regression for $> \mathbf{99.96}\%$ of the target genes. Moreover, increased network scales help to improve GEIDN, while increased training data benefits GEIDN more than other methods.

5. DANN. Annotating noncoding genetic variants to identify pathogenicity remains challenging. CADD is a state-of-the-art algorithm to annotate both coding and noncoding variants. CADD trains a linear SVM to differentiate evolutionarily derived alleles from simulated variants, with 949 highly heterogeneous features. However, linear SVM cannot capture nonlinear relationships among the features. DANN uses the same training features and data as CADD, but trains a neural network. DANN can capture nonlinear relationships among features and deals with large data size better than SVM. Like in GEIDN, we exploit deep learning techniques to train DANN. DANN achieves a **18.90%** relative reduction in the error rate and a **14.52%** relative increase in AUC over CADD.

7.1 Future Directions

Chapters 2–6 have discussed open questions and future directions that are specific for each of the previous projects. Now we propose some higher-level directions:

- Feature representation has room for improvement. In the breast cancer prognosis study, genes' expression features are either selected by ranking the concordance index or linearly transformed via the Metagene algorithm. This can be potentially improved by more sophisticated feature extraction strategies. For example, one might introduce the elastic net penalty (the grouping effect) to the Cox model to select genes, and design an ADMM-alike algorithm to solve it; one can also use the node-splitting mechanism

of GBMCI to automatically select genes; moreover, unsupervised feature learning such as the auto encoder might extract nonlinear features that are more informative than the current feature sets.

- Data fusion remains an open challenge. In the breast cancer prognosis study, clinical covariates and gene expression features are combined by simple "concatenation" and provide improved predictive power; in the genetic variant annotation study, highly heterogeneous features are expanded into a "pseudo" feature vector. However, these are by no means the optimal data fusion strategy, in particular when one considers about the intrinsic connections among the raw feature sets. For example, gene expression features might be the cause of many clinical features, while certain genetic variant features might share redundant information. Therefore, more informative and more principled data fusion strategies are needed for large-scale genomic problems.

- Interpretation of learned computational models and features brings a series of scientific questions. For example, what biological meaning do the sparse part and low-rank part of LVGG indicate? How to interpret the hidden layer features and the weights of GEIDN? How well can LVGG and GEIDN characterize the underlying gene regulatory networks & pathways as known by biologists? To address these questions, one needs to analyze the learned model (the weights, structures, etc.) using the domain knowledge. One can even design models that encapsulate the domain knowledge. Training such models might provide more informative features to biologists.

- Developed models need more thorough applications and verifications to consolidate their utilities. For example, how to evaluate the inference quality on the L1000 data remains not clearly defined. A proper evaluation of GEIDN on L1000 data may be possible with a matched, representative gold standard dataset generated by a genome-wide assay such as the Affymetrix microarray platform.

Bibliography

[1] P. D. Allison. *Survival Analysis Using SAS: A Practical Guide*. SAS Institute, 2010.

[2] U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, and A. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Natl Acad. Sci. USA*, 96(12):6745, 1999.

[3] P. K. Andersen and R. D. Gill. Cox's regression model for counting processes: a large sample study. *The Annals of Statistics*, 10(4):1100–1120, 1982.

[4] M. Baker. One-stop shop for disease genes. *Nature*, 491(7423):171, Nov 2012.

[5] P. Baldi and P. J. Sadowski. Understanding dropout. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. Curran Associates, Inc., 2013.

[6] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *The Journal of Machine Learning Research*, 9:485–516, 2008.

[7] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[8] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[9] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[10] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[11] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010.

[12] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Computer Science and Applied Mathematics. Academic Press, Inc., New York, 1982.

[13] C. J. Burges, K. M. Svore, P. N. Bennett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. *Journal of Machine Learning Research*, 14:25–35, 2011.

[14] P. R. Burton, D. G. Clayton, L. R. Cardon, N. Craddock, P. Deloukas, A. Duncanson, D. P. Kwiatkowski, M. I. McCarthy, W. H. Ouwehand, N. J. Samani, et al. Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. *Nature*, 447(7145):661–678, 2007.

[15] J.-F. Cai, S. Osher, and Z. Shen. Split bregman methods and frame based image restoration. *Multiscale Model. Simul.*, 8(2):337–369, 2009.

[16] E. Candes, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Arxiv preprint arXiv:0912.3599*, 2009.

[17] G. C. Cawley, N. L. Talbot, G. J. Janacek, and M. W. Peck. Sparse bayesian kernel survival analysis for modeling the growth domain of microbial pathogens. *Neural Networks, IEEE Transactions on*, 17(2):471–481, 2006.

[18] V. Chandrasekaran, P. Parrilo, and A. Willsky. Latent variable graphical model selection via convex optimization. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1610–1613. IEEE, 2010.

[19] Y. Chen, Z. Jia, D. Mercola, and X. Xie. A gradient boosting algorithm for survival analysis via direct optimization of concordance index. *Computational and mathematical methods in medicine*, 2013, 2013.

[20] W.-Y. Cheng, T.-H. O. Yang, and D. Anastassiou. Biomolecular events in cancer revealed by attractor metagenes. *PLoS Computational Biology*, 9(2):e1002920, 2013.

[21] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[22] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew. Deep learning with COTS HPC systems. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, pages 1337–1345, Atlanta, 2013.

[23] D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972.

[24] D. R. Cox. Partial likelihood. *Biometrika*, 62(2):269–276, 1975.

[25] D. R. Cox and D. Oakes. *Analysis of Survival Data*, volume 21. Chapman & Hall/CRC, 1984.

[26] C. Curtis, S. P. Shah, S.-F. Chin, G. Turashvili, O. M. Rueda, M. J. Dunning, D. Speed, A. G. Lynch, S. Samarajiwa, Y. Yuan, et al. The genomic and transcriptomic architecture of 2,000 breast tumours reveals novel subgroups. *Nature*, 486(7403):346–352, 2012.

[27] R. K. Curtis, M. Orešič, and A. Vidal-Puig. Pathways to the analysis of microarray data. *Trends in biotechnology*, 23(8):429–435, 2005.

[28] A. Dempster. Covariance selection. *Biometrics*, 28(1):157–175, 1972.

[29] L. Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3:e2, 2014.

[30] P. Di Lena, K. Nagata, and P. Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28(19):2449–2457, 2012.

[31] J. Eckstein and D. Bertsekas. On the douglas rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318, 1992.

[32] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 2004. With discussion, and a rejoinder by the authors.

[33] L. Evers and C.-M. Messow. Sparse kernel methods for high-dimensional survival data. *Bioinformatics*, 24(14):1632–1638, 2008.

[34] D. Faraggi and R. Simon. A neural network model for survival data. *Statistics in Medicine*, 14(1):73–82, 1995.

[35] V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for large-scale risk minimization. *The Journal of Machine Learning Research*, 10:2157–2192, 2009.

[36] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *The Journal of Machine Learning Research*, 4:933–969, 2003.

[37] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[38] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag New York, 2009.

[39] J. H. Friedman. Greedy function approximation: a gradient boosting machine.(english summary). *Ann. Statist*, 29(5):1189–1232, 2001.

[40] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[41] W. Fu et al. Analysis of 6,515 exomes reveals the recent origin of most human protein-coding variants. *Nature*, 493(7431):216–20, Jan 2013.

[42] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers and Mathematics with Applications*, 2(1):17–40, 1976.

[43] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256, Chia Laguna Resort, 2010.

[44] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par penalisation-dualité, d'une classe de problèmes de dirichlet non linéaires. *Rev. Franc. Automat. Inform. Rech. Operat.*, pages 41–76, 1975.

[45] T. Goldstein and S. Osher. The split Bregman method for $L1$-regularized problems. *SIAM J. Imaging Sci.*, 2(2):323–343, 2009.

[46] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[47] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1):389–422, 2002.

[48] F. E. Harrell, R. M. Califf, D. B. Pryor, K. L. Lee, and R. A. Rosati. Evaluating the yield of medical tests. *JAMA: The Journal of the American Medical Association*, 247(18):2543–2546, 1982.

[49] F. E. Harrell, K. L. Lee, and D. B. Mark. Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*, 15(4):361–387, 1996.

[50] T. Hastie, R. Tibshirani, D. Botstein, and P. Brown. Supervised harvesting of expression trees. *Genome Biology*, 2(1):0003.1–0003.12, 2003.

[51] T. Hastie, R. Tibshirani, M. Eisen, A. Alizadeh, R. Levy, L. Staudt, W. Chan, D. Botstein, and P. Brown. Gene shaving as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology*, 1(2):1–0003, 2000.

[52] M. Hecker, S. Lambeck, S. Toepfer, E. Van Someren, and R. Guthke. Gene regulatory network inference: data integration in dynamic models?a review. *Biosystems*, 96(1):86–103, 2009.

[53] M. R. Hestenes. Multiplier and gradient methods. *J. Optimization Theory Appl.*, 4:303–320, 1969.

[54] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[55] J. N. Hirschhorn. Genomewide association studies–illuminating biologic pathways. *New England Journal of Medicine*, 360(17):1699, 2009.

[56] T. Hothorn, P. Bühlmann, S. Dudoit, A. Molinaro, and M. J. Van Der Laan. Survival ensembles. *Biostatistics*, 7(3):355–373, 2006.

[57] T. Hughes, M. Marton, A. Jones, C. Roberts, R. Stoughton, C. Armour, H. Bennett, E. Coffey, H. Dai, Y. He, et al. Functional discovery via a compendium of expression profiles. *Cell*, 102(1):109–126, 2000.

[58] H. Ishwaran and U. B. Kogalur. Random survival forests for r. *New Functions for Multivariate Analysis*, page 25, 2007.

[59] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841–860, 2008.

[60] G. Karlebach and R. Shamir. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology*, 9(10):770–780, 2008.

[61] M. W. Kattan. Comparison of cox regression with other methods for determining prediction models and nomograms. *The Journal of Urology*, 170(6):S6–S10, 2003.

[62] A. Khosla, Y. Cao, C. C.-Y. Lin, H.-K. Chiu, J. Hu, and H. Lee. An integrated machine learning approach to stroke prediction. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 183–192. ACM, 2010.

[63] M. Kircher et al. A general framework for estimating the relative pathogenicity of human genetic variants. *Nature genetics*, 46(3):310–315, 2014.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[65] J. Lamb. The Connectivity Map: a new tool for biomedical research. *Nature Reviews Cancer*, 7:54–60, 2007.

[66] J. Lamb, E. D. Crawford, D. Peck, J. W. Modell, I. C. Blat, M. J. Wrobel, J. Lerner, J.-P. Brunet, A. Subramanian, K. N. Ross, M. Reich, H. Hieronymus, G. Wei, S. A. Armstrong, S. J. Haggarty, P. A. Clemons, R. Wei, S. A. Carr, E. S. Lander, and T. R. Golub. The Connectivity Map: using gene-expression signatures to connect small molecules, genes, and disease. *Science*, 313(5795):1929–1935, 2006.

[67] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[68] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In L. Bottou and M. Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616, Montreal, June 2009. Omnipress.

[69] Y. Lin and H. H. Zhang. Component selection and smoothing in multivariate nonparametric regression. *Ann. Statist.*, 34(5):2272–2297, 2006.

[70] Z. Lu. Smooth optimization approach for sparse covariance selection. *SIAM J. Optim.*, 19(4):1807–1827, 2008.

[71] E. R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends in genetics*, 24(3):133–141, 2008.

[72] A. A. Margolin, E. Bilal, E. Huang, T. C. Norman, L. Ottestad, B. H. Mecham, B. Sauerwine, M. R. Kellen, L. M. Mangravite, M. D. Furia, et al. Systematic analysis of challenge-driven improvements in molecular prognostic models for breast cancer. *Science Translational Medicine*, 5(181):181re1–181re1, 2013.

[73] L. Mariani, D. Coradini, E. Biganzoli, P. Boracchi, E. Marubini, S. Pilotti, B. Salvadori, R. Silvestrini, U. Veronesi, R. Zucali, et al. Prognostic factors for metachronous contralateral breast cancer: a comparison of the linear cox regression model and its artificial neural network extension. *Breast Cancer Research and Treatment*, 44(2):167–178, 1997.

[74] T. Martinussen and T. H. Scheike. *Dynamic Regression Models for Survival Data*. Springer, 2006.

[75] P. McCullagh and J. A. Nelder. *Generalized Linear Models*, volume 37. Chapman & Hall/CRC, 1989.

[76] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, 34(3):1436–1462, 2006.

[77] M. L. Metzker. Sequencing technologies?the next generation. *Nature Reviews Genetics*, 11(1):31–46, 2009.

[78] S. Mukherjee, P. Tamayo, D. Slonim, A. Verri, T. Golub, J. Mesirov, and T. Poggio. Support vector machine classification of microarray data. *CBCL Paper*, 182, 1999.

[79] D. Peck, E. D. Crawford, K. N. Ross, K. Stegmaier, T. R. Golub, and J. Lamb. A method for high-throughput gene expression signature analysis. *Genome Biology*, 7:R61, 2006.

[80] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[81] J. Peng, P. Wang, N. Zhou, and J. Zhu. Partial correlation estimation by joint sparse regression models. *Journal of the American Statistical Association*, 104(486):735–746, 2009.

[82] V. Raykar, H. Steck, B. Krishnapuram, C. Dehing-Oberije, and P. Lambin. On ranking in survival analysis: bounds on the concordance index. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1209–1216. MIT Press, Cambridge, MA, 2008.

[83] G. Ridgeway. The state of boosting. *Computing Science and Statistics*, 31:172–181, 1999.

[84] G. Ridgeway. Generalized boosted models: A guide to the gbm package. 2007.

[85] R. M. Ripley, A. Harris, and L. Tarassenko. Neural network models for breast cancer prognosis. *Neural Computing & Applications*, 7(4):367–375, 1998.

[86] R. M. Ripley, A. L. Harris, and L. Tarassenko. Non-linear survival analysis using neural networks. *Statistics in Medicine*, 23(5):825–842, 2004.

[87] R. T. Rockafellar. A dual approach to solving nonlinear programming problems by unconstrained optimization. *Math. Programming*, 5:354–373, 1973.

[88] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial Mathematics, 2003.

[89] K. Scheinberg, S. Ma, and D. Goldfarb. Sparse inverse covariance selection via alternating linearization methods. *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[90] M. Segal, K. Dahlquist, and B. Conklin. Regression approaches for microarray data analysis. *J. Comput. Biol.*, 10(6):961–980, 2003.

[91] S. Setzer. Split bregman algorithm, douglas-rachford splitting and frame shrinkage. *Scale space and variational methods in computer vision*, pages 464–476, 2009.

[92] J. Shendure and H. Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[93] P. K. Shivaswamy, W. Chu, and M. Jansche. A support vector approach to censored targets. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 655–660. IEEE, 2007.

[94] R. Sladek, G. Rocheleau, J. Rung, C. Dina, L. Shen, D. Serre, P. Boutin, D. Vincent, A. Belisle, S. Hadjadj, et al. A genome-wide association study identifies novel risk loci for type 2 diabetes. *Nature*, 445(7130):881–885, 2007.

[95] R. Socher, C. C.-Y. Lin, A. Ng, and C. Manning. Parsing natural scenes and natural language with recursive neural networks. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 129–136, Bellevue, June 2011. ACM.

[96] N. Srivastava. *Improving neural networks with dropout.* PhD thesis, University of Toronto, 2013.

[97] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[98] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, Atlanta, 2013.

[99] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, 58(1):267–288, 1996.

[100] V. Van Belle, K. Pelckmans, S. Van Huffel, and J. A. Suykens. Support vector methods for survival analysis: a comparison between ranking and regression approaches. *Artificial Intelligence in Medicine*, 53(2):107–118, 2011.

[101] V. N. Vapnik. *Statistical learning theory.* Adaptive and Learning Systems for Signal Processing, Communications, and Control. John Wiley & Sons Inc., New York, 1998. A Wiley-Interscience Publication.

[102] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[103] C. Wang, D. Sun, and K. Toh. Solving log-determinant optimization problems by a newton-cg primal proximal point algorithm. *SIAM J. Optimization*, pages 2994–3013, 2010.

[104] K. Wang, M. Li, and M. Bucan. Pathway-based approaches for analysis of genomewide association studies. *The American Journal of Human Genetics*, 81(6):1278–1283, 2007.

[105] L. Wang, J. Zhu, and H. Zou. The doubly regularized support vector machine. *Statistica Sinica*, 16(2):589, 2006.

[106] L. Wang, J. Zhu, and H. Zou. Hybrid huberized support vector machines for microarray classification and gene selection. *Bioinformatics*, 24(3):412, 2008.

[107] C. Wu and X. Tai. Augmented Lagrangian method, dual methods, and split Bregman iteration for ROF, vectorial TV, and high order models. *SIAM Journal on Imaging Sciences*, 3:300, 2010.

[108] L. Yan, D. Verbel, and O. Saidi. Predicting prostate cancer recurrence via maximizing the concordance index. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 479–485. ACM, 2004.

[109] G. Ye, M. Tang, J.-F. Cai, Q. Nie, and X. Xie. Low-rank regularization for learning gene expression programs. *PLOS ONE*, 8(12):e82146, 2013.

[110] G. Ye and X. Xie. Split bregman method for large scale fused lasso. *Computational Statistics and Data Analysis*, 55(4):1552–1569, 2011.

[111] G.-B. Ye, Y. Chen, and X. Xie. Efficient variable selection in support vector machines via the alternating direction method of multipliers. In *International Conference on Artificial Intelligence and Statistics*, pages 832–840, 2011.

[112] G.-B. Ye, Y. Wang, Y. Chen, and X. Xie. Efficient latent variable graphical model selection via split bregman method. *arXiv preprint arXiv:1110.3076*, 2011.

[113] M. Yuan and Y. Lin. Model selection and estimation in the gaussian graphical model. *Biometrika*, 94(1):19–35, 2007.

[114] X. Yuan. Alternating direction methods for sparse covariance selection. *preprint*, 2009.

[115] E. Zeggini, M. N. Weedon, C. M. Lindgren, T. M. Frayling, K. S. Elliott, H. Lango, N. J. Timpson, J. R. Perry, N. W. Rayner, R. M. Freathy, et al. Replication of genome-wide association signals in uk samples reveals risk loci for type 2 diabetes. *Science*, 316(5829):1336–1341, 2007.

[116] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, 2003.

[117] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 67(2):301–320, 2005.

[118] B. Zupan, J. Demšar, M. W. Kattan, J. R. Beck, and I. Bratko. Machine learning for survival analysis: a case study on recurrence of prostate cancer. *Artificial Intelligence in Medicine*, 20(1):59–75, 2000.

Appendix A

Supplementary Material for Chapter 5

A.1 Supplementary Table

Table S1: The configurations of three GEIDN models

	GEIDN-1	GEIDN-2	GEIDN-3
Architecture [a]	[978, 3000, 3000, 5323]	[978, 6000, 6000, 5323]	[978, 6000, 8000, 8000, 5323]
Dropout [b]	[1.0, 0.9, 0.9]	[1.0, 0.85, 0.85]	[1.0, 0.9, 0.9, 0.9]
Irange [c]	[0.0388, 0.0316, 0.0268]	[0.0293, 0.0224, 0.0230]	[0.0293, 0.0207, 0.0194, 1e-4]
Learning rate [d]	[1e-3, 4e-6]	[5e-4, 4e-6]	[5e-4, 4e-6]
Momentum [e]	[0.01, 0.5, 20]	[0.01, 0.5, 50]	[0.01, 0.5, 50]
Mini-batch size	200	200	200
Training Epochs	200	300	400

[a] The number of neurons in each layer, from the input layer to the output layer.
[b] The probability p to random include a neuron in each layer during dropout training, from the input layer to the last hidden layer.
[c] The boundary of the uniform distribution to sample the initial weights from in each layer, from the first hidden layer to the output layer. See Equation 5.8 for details.
[d] Each entry specifies the initial learning rate η_0, and the small positive value δ used for learning rate annealing. See Equation 5.7 for details.
[e] Each entry specifies the initial momentum, the final momentum, and the number of epochs to finish the momentum update. The momentum is increased linearly over epochs.

A.2 Supplementary Figure

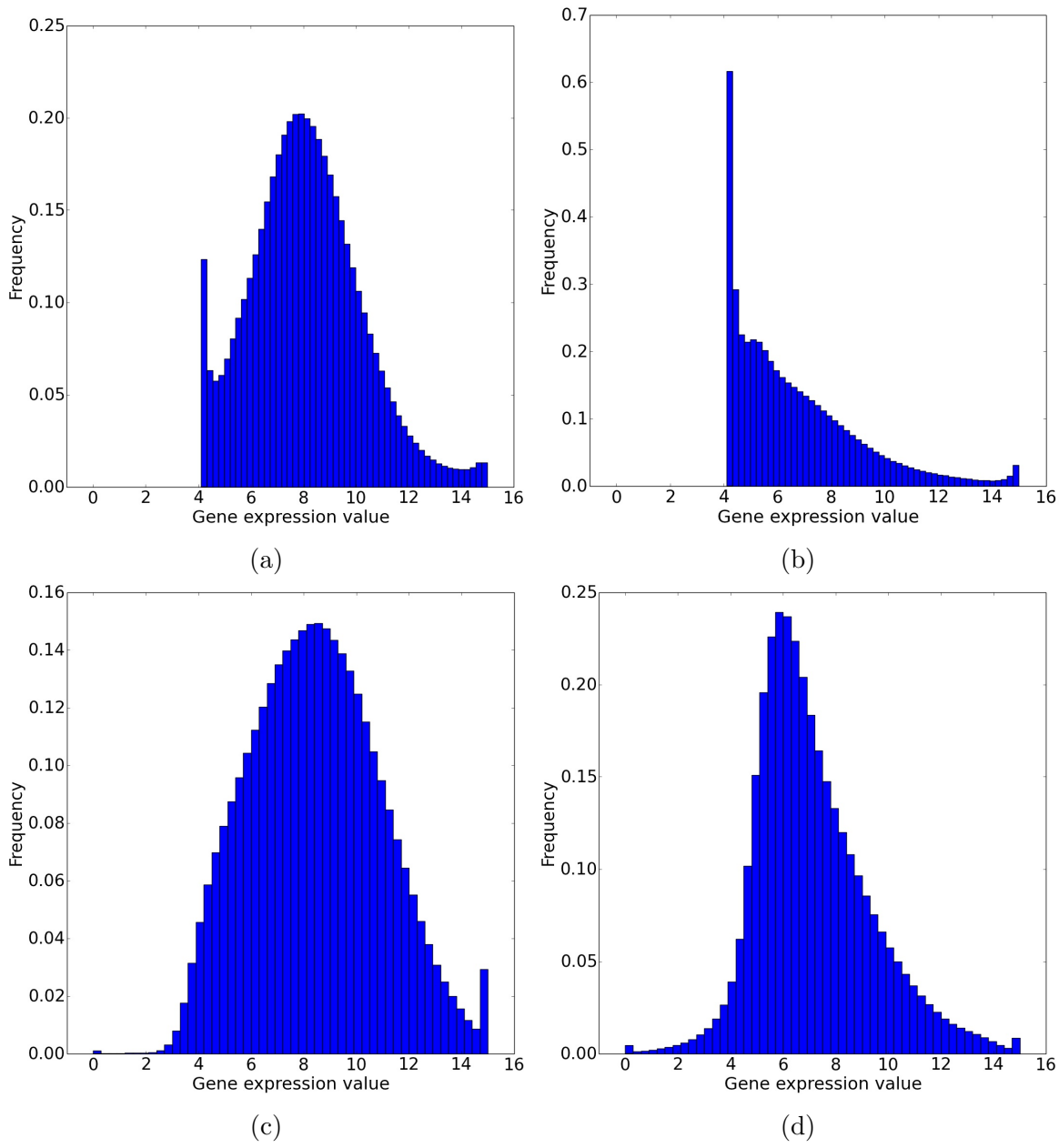


Figure S1: Marginal distribution of gene expression values over all: (a) landmark genes of the GEO data; (b) target genes of the GEO data; (c) landmark genes of the L1k-sub data; (d) target genes of the L1k-sub data. Note that target genes of the L1k-sub data are imputed via linear regression and clipped to be between 0 and 15.

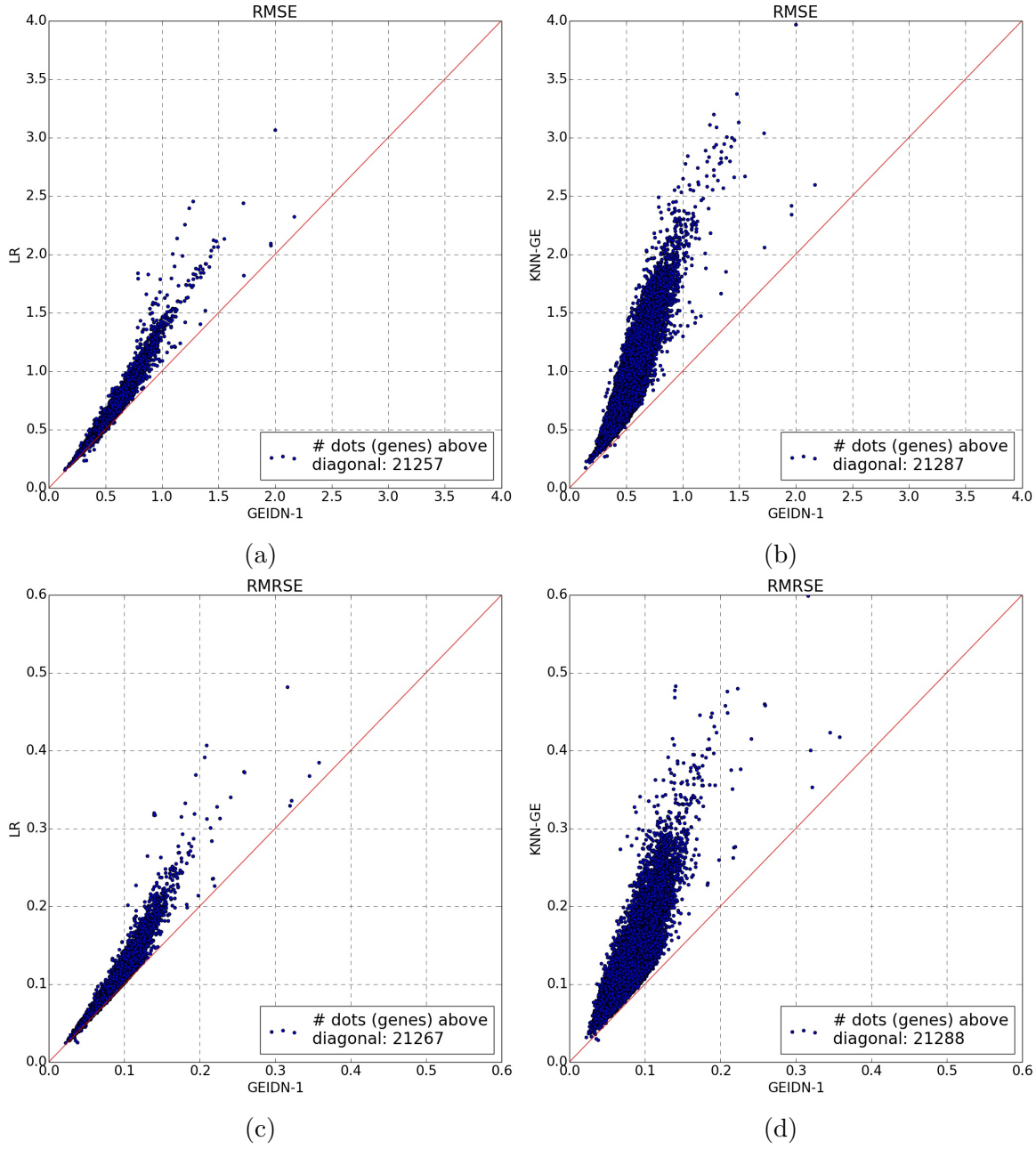


Figure S2: The predictive errors of GEIDN-1 compared to LR and KNN-GE. (a) RMSEs of GEIDN-1 versus those of LR; (b) RMSEs of GEIDN-1 versus those of KNN-GE; (c) RMRSEs of GEIDN-1 versus those of LR; (d) RMRSEs of GEIDN-1 versus those of KNN-GE.

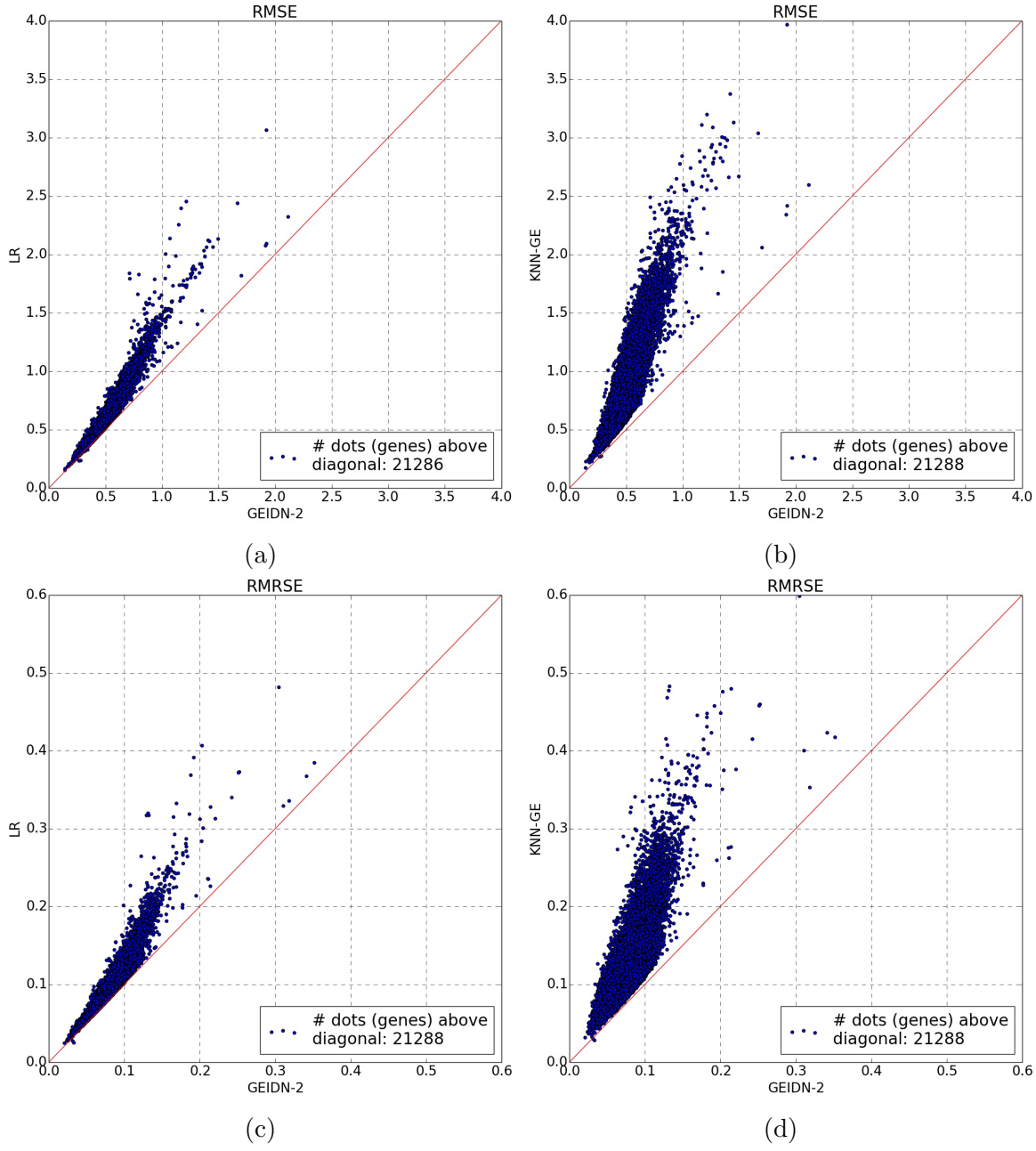


Figure S3: The predictive errors of GEIDN-2 compared to LR and KNN-GE. (a) RMSEs of GEIDN-2 versus those of LR; (b) RMSEs of GEIDN-2 versus those of KNN-GE; (c) RMRSEs of GEDIDN-2 versus those of LR; (d) RMRSEs of GEIDN-2 versus those of KNN-GE.

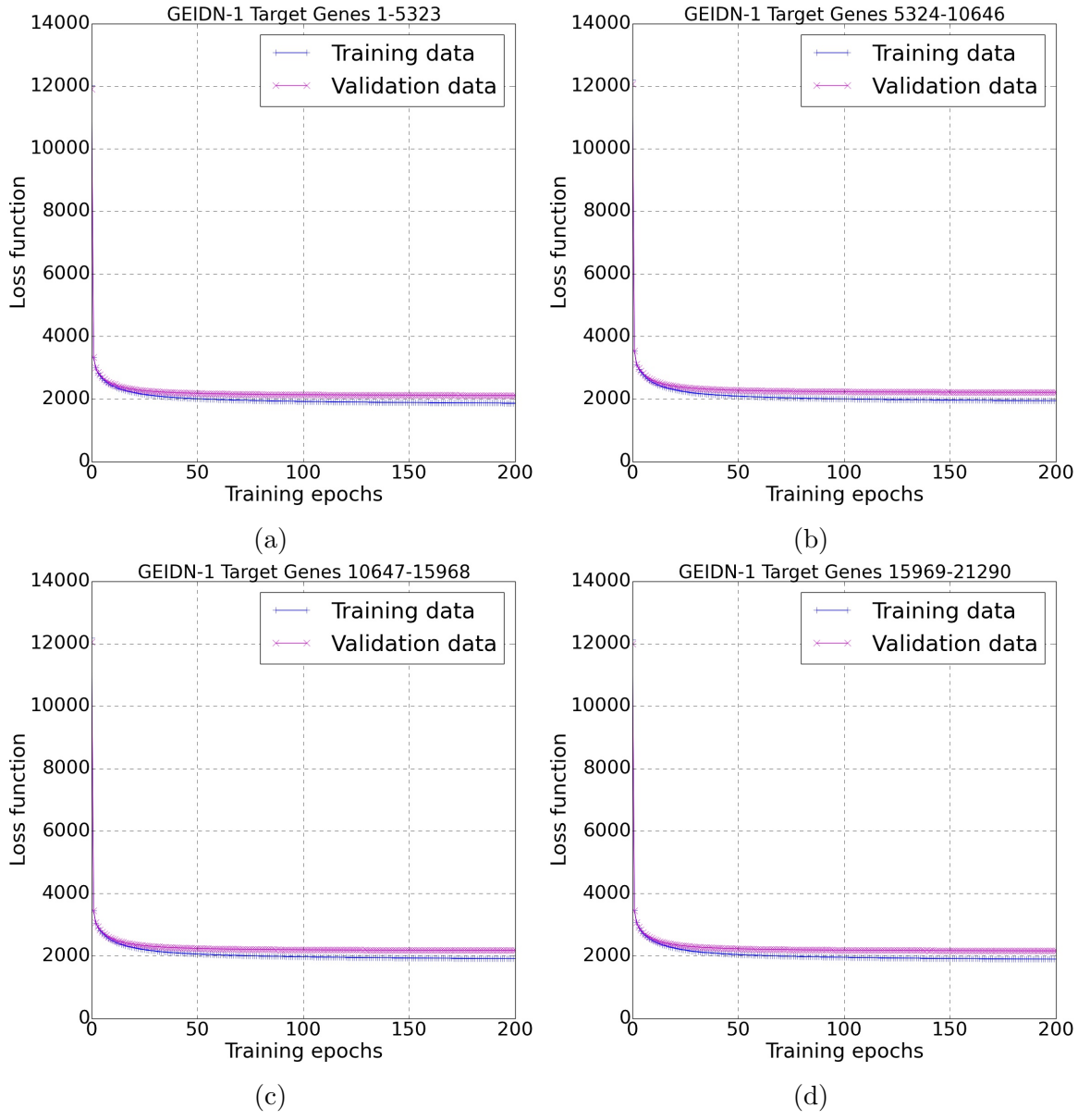


Figure S4: Learning curves of GEIDN-1. The target genes are randomly permuted and then assigned with labels 1˜21,290. After that, they are evenly distributed into four neural networks for training and prediction, whose learning curves are shown by (a), (b), (c), and (d) separately. See Section 5.3.2 for the motivation of this design.

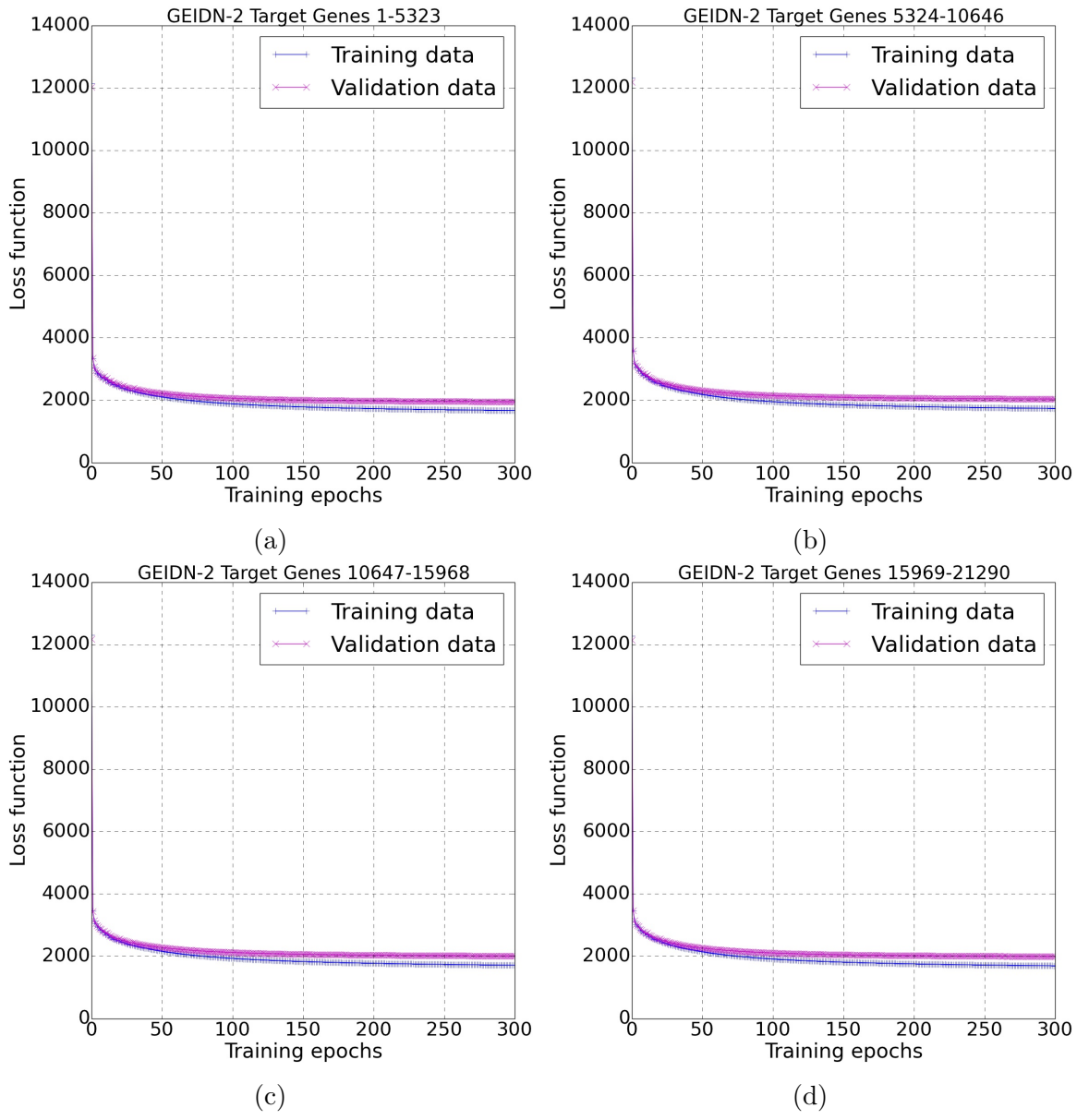
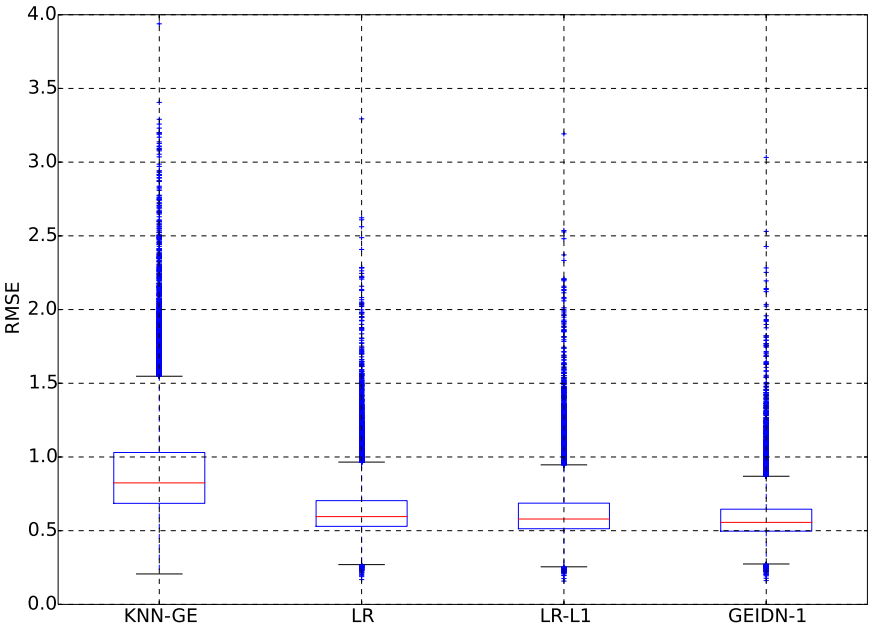
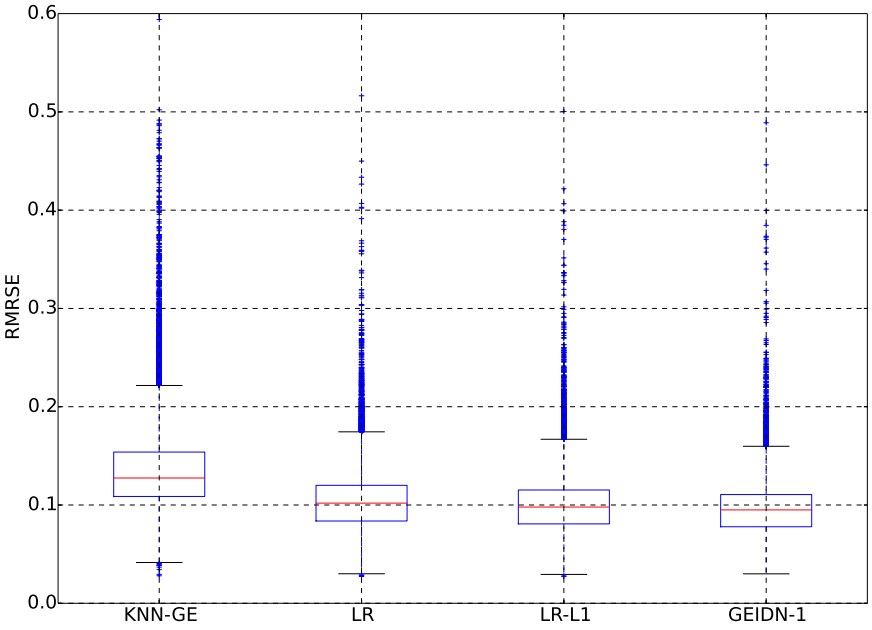


Figure S5: Learning curves of GEIDN-2. The target genes are randomly permuted and then assigned with labels 1~21,290. After that, they are evenly distributed into four neural networks for training and prediction, whose learning curves are shown by (a), (b), (c), and (d) separately. See Section 5.3.2 for the motivation of this design.



(a)



(b)

Figure S6: The prediction error box plot over all target genes for LR, LR-L1, KNN-GE and GEIDN-1. GEO-tr-sub, GEO-va-sub, and GEO-te are used for model training, validation, and testing. (a) Using RMSE as the evaluation metric; (b) using RMRSE as the evaluation metric.