MULTIPLE LEARNING FOR GENERALIZED LINEAR MODELS IN BIG DATA

by

Xiang Liu

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Department of Computer and Information Technology

West Lafayette, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Baijian Yang, Co-Chair

Department of Computer and Information Technology

Dr. Tonglin Zhang, Co-Chair

Department of Statistics

Dr. Jin Wei-Kocsis

Department of Computer and Information Technology

Dr. Abdul Salam

Department of Computer and Information Technology

Approved by:

Dr. Kathryne A Newton

Chair of the Graduate Education Committee

To my beloved family for their supports

ACKNOWLEDGMENTS

I want to sincerely acknowledge my dissertation committee members for their guide. My committee co-charis, Dr. Yang and Dr. Zhang are very talented. They helped me to find many innovative ideas. All I need to do was to implement the ideas with a few of modifications. They also helped me modify every paper published. Dr. Yang and Dr. Zhang are proficient in English, they corrected hundreds of grammatical mistakes I made. They also taught me how to word and how to achieve elegance in writing. Everything they imparted will benefit my entire life. For Dr. Jin Wei-Kocsis and Dr. Abdul Salam, they are very professional in their research area and they gave many pieces of inspirational advice to my researches. They are also very kind and patient professors. Thank for again for the help and guide from the professors. I could not finish my researches and dissertation without them.

And thanks for the support from my parents. When I received the offer from Purdue University. It is only an admission letter. No funding is included to support my doctoral study. It is a huge amount of money. But my parents are still very supportive to me. They said they would support my decision to achieve the PhD degree regardless of assistantship or scholarship (Speaking of funding, I would like to appreciate Dr. Yang again for his 5-years funding support). Without their understanding, I probably will just give up studying as PhD student, and I could not achieve my PhD degree. Though the Covid apart us, our love makes us always together as a family.

Also, thank for the help from my friends, Huyunting Huang, Ziyang Tang, Guangyu Shen, Weitao Tang, Wanzi Jiangand, Zhenzhi Xu and Junhan Zhao. Huyunting Huang is a vert smart PhD student, he helped me solve a lot of mathematical problems. I hope he could get to a higher level in the future. Ziyang Tang and I collaborate a lot. We published many papers together. I will not forget PNET, which is a great paper he came up with. PNET received many citations from the public. He also warms me a lot. He is a gentleman. I hope we can collaborate more in the future. Guangyu Shen is a hard-working PhD student, He asked me many questions regarding adversarial attacks. However, I do not know the answers (I feel sad for myself). He is also very talented in playing games. I hope he can publish more papers on top conference and graduate sooner. Weitao Tang is another hard-working PhD student. But he took things too serious. I hope he would live

happier. Also, thank for the encouragement from Wanzhi Jiang and Zhenzhi Xu. They are my ”shi jie“. Shi jie is a Chinese word for fellow apprentices. Although they graduate in a year after I came to DAO2 lab. They keep encouraging me and guide me on my career planning. At last, I would like to especially appreciate JunHan Zhao for him keeping push me to pursuit my goal. I am a lazy person, without his pushing and encouragement, I could not imagine what I would become.

I would also like to thank Purdue University. It is a beautiful university. I like the campus and the spirits the university conveys.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

I/O	Inputs/Outputs
RAM	Random Access Memory
SSD	Solid State Drive
HDD	Hard Disk Drive
IoT	Internet of Things
GLM	Generalized Linear Models
SBR	Sparse Block Regression
SBM	Sparse Block Matrix
SS	Sufficient Statistics
SSE	Sum of Square Error
MSE	Mean Square Error
ANOVCA	Analysis of Covariance
IRWLS	iteratively re-weighted least squares
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
RMS	Root Mean Square Error
OLS	Ordinary Least Squares
Box-Cox Regression	Linear Regression with Box-Cox Transformation
MAE	Mean Absolute Error
RSE	Residual Standard Error
AIC	Akaike's Information Criteria
BIC	Bayesian information criteria
R2	R square
adjusted R2	adjusted R square

ABSTRACT

Big data is an enabling technology in digital transformation. It perfectly complements ordinary linear models and generalized linear models, as training well-performed ordinary linear models and generalized linear models require huge amounts of data. With the help of big data, ordinary and generalized linear models can be well-trained and thus offer better services to human beings. However, there are still many challenges to address for training ordinary linear models and generalized linear models in big data. One of the most prominent challenges is the computational challenges. Computational challenges refer to the memory inflation and training inefficiency issues occurred when processing data and training models. Hundreds of algorithms were proposed by the experts to alleviate/overcome the memory inflation issues. However, the solutions obtained are locally optimal solutions. Additionally, most of the proposed algorithms require loading the dataset to RAM many times when updating the model parameters. If multiple model hyper-parameters needed to be computed and compared, e.g. ridge regression, parallel computing techniques are applied in practice. Thus, multiple learning with sufficient statistics arrays are proposed to tackle the memory inflation and training inefficiency issues.

CHAPTER 1. INTRODUCTION

In Introduction chapter, we will introduce the big data research problems that have been studied in the dissertation. We will define the research scope, significance, and research questions. The assumptions, limitation and delimitation of the research problem are also discussed.

1.1 Scope

Big data research problems arise due to recent advances in computer technologies. As data are collected everyday either online or offline, the size of big data increases, which causes it is impossible to use traditional machine learning approaches. To overcome these challenges, new machine learning approaches are needed. In big data research, the two most important challenges are memory barriers and computational efficiency barriers (Meeker & Hong, 2014). A memory barriers appear if a large scale dataset cannot be fit into hard disk drive (HDD) or solid state drive (SSD) of a computer. Therefore, a large scale dataset is usually kept in the cloud, leading to the need of cloud computing. A computational efficiency barrier appears if one wants to derive the results of machine learning models in a short time. Because a collection of parameters for the datasets are usually evaluated for the models, it is important to have an efficient machine learning method which can provide the results for a number of machine learning models together. This motivates the research problems that have been studied by the dissertation.

Big data is one of the four enabling technologies of the digital transformation (Matt, Hess, & Benlian, 2015). It is rapidly expanding in many fields, e.g. Internet of Things (IoT) and computer vision (Forsyth & Ponce, 2011; Xia, Yang, Wang, & Vinel, 2012). Big data research has great potentials to reveal the patterns and trends of the systems around us. It has impacts in wide applied fields, which can significantly affects our life.

Big data is often characterized by the 5V's (Demchenko, De Laat, & Membrey, 2014). The 5V's characteristics encompass volume, velocity, variety, veracity, and value. Volume stands for the huge amounts of data produced in a daily basis. Velocity stands for the ever-increasing rate of new data. Variety represents various types of data generated everyday. Veracity represents the trustworthiness of data. Value in big data represents the high value of the content of the data. Big

data has brought many extraordinary opportunities for human beings to learn the systems around us.

Big data can be efficiently implemented to ordinary linear models and generalized linear models. Training well-performed ordinary and generalized linear models requires a massive amount of data. Embedded in computing devices, for example smart home and smart phones (Abdulla et al., 2020; Xia et al., 2012), ordinary and generalized linear models can provide excellent services for the improvement of the quality of lives for human beings. A variety of applications have been developed for big data with generalized linear models, e.g. predicting stock prices and house prices (Ponnam, Rao, Srinivas, & Raavi, 2016).

Big data issues are new and innovative. The corresponding research problems for small and moderate data have been studied for decades. These problems are hard if the goal is to train high-performance ordinary or generalized linear models for big data. Various difficulties need to address. In this dissertation, we target to solve the computational challenges with the result to be optimal theoretically.

Computational challenges refer to those in processing and analyzing the data for model training. They arise from extremely large data scales with high increasing rates of new data generation (Oussous, Benjelloun, Lahcen, & Belfkih, 2018). In this case, traditional computing techniques are not appropriate. To overcome the difficulties, parallelization is used. Parallelization stores and process data in many machines individually. The updates of modeling results are aggregated from the machines, such that optimal solutions to the ordinary linear models and generalized linear models models are identified. The goal of parallelization is to make the traditional methods applicable. They do not change the algorithms. Because of this, the training of ordinary linear models and generalized linear models are still inefficient. New training algorithms are needed.

In the dissertation, we focus on designing new algorithms for the ordinary linear models and generalized linear models to be computationally more efficient in the big data settings. We would like the new algorithms to overcome the memory inflation and training time inefficiency issues.

1.2 Significance

Because of the 5V's, the datasets collected for generalized linear models are measured in gigabytes or terabytes. The scale of the datasets leads to high throughput and I/O during data processing and model training. Although distributed systems and cloud computing enable the data processing and model training, the inefficiency of training is still critical (Zaharia, Chowdhury, Franklin, Shenker, & Stoica, 2010). For example, the iterative methods, e.g. gradient descent to solve the linear regression require multiple passes over the datasets. Modification is required to make the methods more efficient.

1.3 Research Question

For some ordinary and generalized linear models, such as linear weighted linear model, it is possible to obtain the exact solutions. However, the exact solutions are usually not applicable in big data as obtaining exact solutions requires loading entire dataset in memory (memory inflation issue). In big data, iterative methods, e.g. gradient descent, are commonly used to solve ordinary linear models and generalized linear models. Iterative methods require many iterations until convergence to solve the ordinary linear models and generalized linear models, and thus are time inefficient. Besides, the selection of hyper-parameters is inevitable under many conditions. For big data projects, it may take several days or even several weeks to fine tune the models for desired performance. In order to integrate the advantages and disadvantages of the exact solutions and the iterative methods, we target to address the following research questions.

The first is the possibility to keep the performance of the original algorithms while reducing the training time. Given dataset of moderately large size, it is efficient to implement the original algorithm to the data individually, as the computation for each of those is fast. As optimal strategies are always a concern in real applications, a number of methods are usually considered in the analysis of the data. Therefore, it is inefficient to implement those methods individually as the computation is time-consuming. This problem can be overcome by multiple learning that has been proposed in the dissertation.

The second is the derivation of the solutions of a number of ordinary linear models or generalized linear models by only a single-pass through the data set. In traditional statistical/ML approach, the fitting of ordinary linear models or generalized linear models usually starts from the reading of the data, which makes it impossible to consider a number of models simultaneously when the data set is accessed. To overcome the difficulty, we propose a method to consider reading data and fitting models procedures.

The third is the derivation of a precise solution for big data, such that it is close or identical to the theoretical result. For traditional regression models, the solutions are obtained via optimization methods given large scale datasets. In this situation, the solutions obtained are not optimal. To obtain exact solutions, the dataset size required could not be too huge, otherwise, it is impossible to get exact solutions. To overcome the difficulty, we propose a method to process the data at per-row level with exact solutions.

In conclusion, there are three research questions:

1. Is it possible to keep the performance of the original algorithms while reducing the training time?

2. Is it possible to obtain the solutions of regression models with only a single-pass through the datasets?

3. Is it possible to obtain the exact solution while the memory inflation issue is overcome?

1.4 Assumptions

The research study assumes the relationship of the expected values and the standard deviations between the samples and the population data. This study also makes an assumption on the dataset distribution.

Below is the assumptions for this study:

1. the expected values of the samples from datasets is assumed to be the same as the population mean.

2. the standard deviation of the samples from datasets is assumed to be the same as the population standard deviation.

3. the datasets used in this dissertation, by default, follow normal distribution if not specifically mentioned.

1.5 Limitations

The limitations of the study contains memory size limitation and dataset source limitation. Because of the budget, the computer cluster we built is relatively small and cannot be compared with the computer clusters used by the top conference authors. Thus, the memory size of the computer cluster is small. And considering the budget, the configuration of the cluster cannot be upgraded any more. For dataset source, as the public datasets are usually not large enough, the datasets we used for experiments are synthesized.

The limitations can be concluded as two points:

1. the memory used for the computer cluster is relatively small as opposed to the highly-configured computer clusters from the papers published in the top conferences or top journals.

2. it is difficult to find a pubic big data level dataset in reality, thus, datasets used in the experiments are synthesized.

1.6 Delimitations

In this dissertation, we mainly focused on the ordinary linear models, including linear model, weighted linear model, linear model with Box-Cox transformation, linear model with categorical independent variable, ridge regression, lasso regression and generalized linear models. Thus, in the experiments, we only compared the performance of our proposed approaches with the traditional (or usual) models. The performance of other machine learning techniques, e.g., decision tree (Fürnkranz, 2010), and deep learning techniques, e.g., resnet (He, Zhang, Ren, & Sun, 2015), are not evaluated and compared.

1.7 Definitions

We have the following terms:

Linear model (or regression) - It is a statistic model that linearly model the relationship between the dependent variable and a set of independent variables.

Weighted linear model (or regression) - It is similar to linear model, however, it puts different weights on different samples.

Box-Cox regression - It is similar to linear model, however, but the linear relationship is assumed for the dependent variable with power transformation and the set of independent variables.

Linear model with categorical independent variable - It is linear model. One or more of the independent variables are categorical. For example, assume one of the independent variables from the dataset is "education level for Purdue university students". The education level variable for Purdue university students is categorical, as it can only be a value from bachelor, master, doctoral.

Lasso regression - It is linear model with ℓ_1 penalty.

Ridge regression - It is linear model with ℓ_2 penalty.

Generalized linear model (or regression) - It is a generalized version of linear model that model relationship between the expectation of the dependent variable and a set of independent variables. For linear model, it assumes the error between the expectation of the dependent variable and the independent variables follows normal distribution. But for generalized linear model, it assumes the error between them follows a exponential family distribution.

Sufficient statistics arrays - It is arrays/matrices used to estimate the model parameters. We can re-write the loglikelihood or SSE as a function of sufficient statistics arrays. With sufficient statistics arrays, we do not need to visit the dataset twice.

Sum Square Error - It is the summation of the squares of the errors between the original values and the predictions.

Mean Square Error - It is the average of the squares of the errors between the original values and the predictions.

Likelihood - It is a function of the joint probability of the samples from the dataset.

Log-likelihood - It is a logarithmic function of the likelihood function.

1.8 Summary

The first chapter introduced the research problem, presenting the research scope, questions, as well as the significance, assumptions, limitations, delimitations. At the end, this chapter also explained the terminologies/definitions used in this dissertation.

CHAPTER 2. REVIEW OF LITERATURE

Fully understanding of big data needs new ways of thinking and novel ideas to address various challenges. In this dissertation, we would like to address two of the most prominent computational challenges, including memory inflation issue and training time inefficiency issue. In this chapter, we will present the literature review of relevant techniques that are in demand of being addressed in big data. The main issues consist of the memory inflation issue and training time inefficiency issue.

For computational challenges, we start from the ordinary linear models that exact solutions can be derived. In the literature review chapter, we first introduce our approach to the ordinary linear models with exact solutions. This include linear and weighted linear model, linear model with Box-Cox transformation, linear model with categorical independent variables, ridge regression model. Then we introduce our optimization techniques based on sufficient statistics arrays that can be used to address the computational and memory challenges for these methods in big data. Then, we will introduce lasso regression and generalize linear models that have no exact solution in general.

2.1 Regression

Regression represents a class of statistical and machine learning methods. It models the dependent variable based on a collection of independent variables. The relationship can be complicated. Therefore, many methods developed under the framework have been proposed. Regression can be used to a continuous or a count response. If the dependent variable is continuous, then normal distribution is often used to model the distribution of the dependent variable; otherwise generalized linear models can be used. Therefore, regression can be implemented to both continuous and count data. Regression methods for continuous data are more straightforward than those for count data. Therefore, we review the methods for continuous data first.

2.1.1 Linear Model

Linear model (Neter, Kutner, Nachtsheim, Wasserman, et al., 1996) is a traditional statistical model to connect a normal distributed dependent variable with a collection of independent variables. It assumes that the dependent variable is a linear function of the independent variables.

Assume there is a large scale dataset. The dataset contains n rows and $p-1$ columns. In other words, the dataset has n samples (or observations), and each of the samples in the dataset has $p-1$ features.

A linear model can be generally expressed in (2.1).

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{2.1}$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^\top$, $\boldsymbol{y} \in \mathbb{R}^n$ is a vector of the dependent variable; $\mathbf{X} = (\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \ldots, \boldsymbol{x}_n^\top)^\top$ with $\boldsymbol{x}_i = (1, x_{i1}, \ldots, x_{i(p-1)})^\top$, $\mathbf{X} \in \mathbb{R}^{n \times n}$ is a matrix of the independent variables; $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{p-1})^\top$, $\boldsymbol{\beta} \in \mathbb{R}^p$ is a vector of parameters; $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)^\top$, $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ is a vector of errors. $\boldsymbol{\varepsilon}$ is, in general, assumed to be composed by independent normal errors, such that there is $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$.

The loglikelihood function of (2.1) is

$$\mathscr{L}_{lr}(\boldsymbol{\beta}, \sigma^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \tag{2.2}$$

where $\|\cdot\|_2$ is an ℓ_2 norm. By maximizing loglikelihood function given by (2.2) for $\boldsymbol{\beta}$ and σ^2, we can get the model parameters. By ordinary least square or gradient decent, we can solve the estimates of $\boldsymbol{\beta}$, σ^2, and the variance-covariance matrix of $\boldsymbol{\beta}$ analytically. The results are

$$\begin{aligned}
&\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}, \\
&\hat{\sigma}^2 = \frac{1}{n-p}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^\top(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) = \frac{1}{n-p}[\mathbf{y}^\top\mathbf{y} - \mathbf{y}^\top\mathbf{X}^\top(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}], \\
&\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\mathbf{X}^\top\mathbf{X})^{-1}.
\end{aligned} \tag{2.3}$$

In (2.3), $\hat{\boldsymbol{\beta}}$ is called the estimate of the model coefficients and $\hat{\sigma}^2$ is called the mean square errors (MSE) of the model. In addition, we need $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$ for the importance of individual explanatory variables. They are also needed in the development of big data algorithms for linear models.

2.1.2 Weighted Linear Model

Weighted linear model is an variation of linear model. Rather than a linear model, a weighted linear regression model assumes the variances of the error terms vary. Therefore, it contains a weight matrix for the observations. As independence is also assumed, the weight matrix is diagonal. This indicates that all the off-diagonal elements are 0.

Let $\mathbf{W}$ be the weighted matrix. w_i is the weight for each observation. A weighted linear model is generally expressed as in (2.4).

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},\ \boldsymbol{\varepsilon} \sim \mathscr{N}(0, \sigma^2\mathbf{W}^{-1}), \tag{2.4}$$

where $\mathbf{W} = \mathrm{diag}(w_1, \ldots, w_n)$ is a known weight matrix.

The loglikelihood function of (2.4) is

$$\ell(\boldsymbol{\beta}, \sigma^2) = -\frac{n}{2}\log(\sigma^2) - \frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\det(\mathbf{W})| - \frac{1}{2\sigma^2}\|\mathbf{W}^{1/2}(\mathbf{W} - \mathbf{X}\boldsymbol{\beta})\|_2^2. \tag{2.5}$$

The SSE cost function of the model is specified by the last term of (2.5) as

$$SSE_{wlr}(\boldsymbol{\beta}_w) = \left\|\mathbf{W}^{1/2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_w)\right\|_2^2. \tag{2.6}$$

By maximizing SSE of weighted linear model in (2.6), the estimates of the model parameters $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ can be solved analytically.

The results are

$$\begin{aligned}
&\hat{\boldsymbol{\beta}}_w = (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}\mathbf{y} \\
&\hat{\sigma}_w^2 = \frac{1}{n}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_w)^\top \mathbf{W}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_w) = \frac{1}{n-p}[\mathbf{y}^\top \mathbf{W}\mathbf{y} - \mathbf{y}^\top \mathbf{X}^\top \mathbf{W}(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}\mathbf{y}] \\
&\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w) = \hat{\sigma}_w^2(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}.
\end{aligned} \tag{2.7}$$

In (2.7), $\hat{\boldsymbol{\beta}}_w$ is the estimator of the model coefficients and $\hat{\sigma}_w^2$ is the MSE of the model. Likewise, $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$ is also used to evaluate the importance of explanatory variables.

2.1.3 Box-Cox Regression

Box-Cox transformation (Box & Cox, 1964; Sakia, 1992) is a class of power transformations for the response of linear regression models. The goal is to stabilize the variances of the linear regression models. It is modified from weighted linear regression for the case when the weight matrix is unknown. The power transformation on dependent variable can be found in (2.8).

$$\mathbf{y}^{(c)} = \mathbf{X}\boldsymbol{\beta}_c + \boldsymbol{\varepsilon} \tag{2.8}$$

where $\mathbf{y}^{(c)}$ is the dependent variable with power transformation. The equation can be found in (2.9), the operation is element-wise.

$$\mathbf{y}^{(c)} = \begin{cases} \log \mathbf{y} \text{ (if } c \text{ equals to 0)} \\ (\mathbf{y}^c - 1)/c \text{ (if } c \text{ not equals to 0)} \end{cases} \tag{2.9}$$

In the Box-Cox transformation given by (2.9), c is usually selected from a candidate collection C, such that each $c \in C$ represents a power transformation. If c is selected, then (2.8) becomes a standard regression model.

In general, a collection C for power transform parameters are applied to the response variable to check which power transform parameters c is the best parameter. For every $c \in C$, it provides a power transformation. Because the optimal c is unknown, it is recommended to

estimate the best parameter by the profile maximum likelihood approach. Based on the profile loglikelihood given by

$$\begin{aligned}\mathscr{L}_{bc}(c,\boldsymbol{\beta}_c,\sigma_c^2) &= -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log\sigma_c^2 - (\mathbf{c}-\mathbf{1})^\top \log \mathbf{y} \\ &- \frac{1}{2\sigma_c^2}(\mathbf{y}^{(c)} - \mathbf{X}\boldsymbol{\beta}_c)^\top(\mathbf{y}^{(c)} - \mathbf{X}\boldsymbol{\beta}_c)\end{aligned} \tag{2.10}$$

The profile maximum likelihood algorithm find the best c from a collection of c by maximizing the likelihood. The c that gives the maximal likelihood value is the best c's. The profile maximum likelihood is given by (2.10). Because the maximizer cannot be analytically solved, numerical methods are needed.

After $\hat{c}$, the estimator of c is obtained, the estimator of $\boldsymbol{\beta}$ and σ^2 by exact solutions is also accessible. In particular, for any $c \in C$, we have

$$\begin{aligned}\hat{\boldsymbol{\beta}}_c &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}^{(c)} \\ \hat{\sigma}_c^2 &= \frac{1}{n}(\mathbf{y}^{(c)} - \mathbf{X}\hat{\boldsymbol{\beta}}_c)^\top(\mathbf{y}^{(c)} - \mathbf{X}\hat{\boldsymbol{\beta}}_c) \\ \hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_c) &= \hat{\sigma}_c^2(\mathbf{X}^\top\mathbf{X})^{-1}.\end{aligned} \tag{2.11}$$

Replace c by $\hat{c}$, we obtain the model parameters of $\boldsymbol{\beta}$, σ^2, and the variance-covariance estimator of $\boldsymbol{\beta}$. Therefore, the major issue in Box-Cox transformation is the derivation of the parameter $\hat{c}$.

2.1.4 Ridge Regression

Ridge regression (Hoerl & Kennard, 1970) is a linear model technique with an ℓ_2 penalty term in its estimation equation. The corresponding SSE cost function of ridge regression model is as follows.

$$SSE_{ridge}(\lambda,\boldsymbol{\beta}_\lambda) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_\lambda\|_2^2 + n\lambda\,\|\boldsymbol{\beta}_\lambda\|_2^2 \tag{2.12}$$

where λ is the ridge parameter with the constraint $\lambda \geq 0$. When $\lambda = 0$, ridge regression is degenerated to linear regression.

Ridge regression can be analytically minimized. For any $\lambda \geq 0$, there are exact solutions for (2.12), yielding the estimators of model parameters $\hat{\boldsymbol{\beta}}$, variance $\hat{\sigma}^2$ and variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ as

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_\lambda &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} \\
\hat{\sigma}^2_\lambda &= \frac{1}{n} (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_\lambda)^\top (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_\lambda) \\
\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_\lambda) &= \hat{\sigma}^2_\lambda (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{X} (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1}
\end{aligned} \tag{2.13}$$

Ridge regression is a powerful tool for linear models when independent variables are highly correlated. In this case, $\mathbf{X}^\top \mathbf{X}$ is almost singular, leading to a phenomenon called multicollinearity in linear regression models. Ridge regression can stabilize the computation of the variance-covariance estimator of model coefficients. It increases the power for the detection of important explanatory variables.

In general, a collection of ridge parameters λ are needed, which is similar to Box-Cox regression, to calculate a best ridge parameter λ for ridge regression. Instead of using the ridge parameter λ that maximizes SSE, ridge trace is used to determine the best ridge parameter λ. Choosing a better ridge parameter λ is beneficial to alleviate the multicollinearity issue in the dataset.

2.1.5 Linear Regression with Categorical Independent Variables

Linear model with categorical independent variables is linear model with one or more independent variables being categorical. For example, surveys usually require the information of biological gender from the survey takers. The biological gender is the gender at birth, including male and female. Categorical independent variables are also called factor variables.

When categorical variables are in the model expression, the classical method to solve this problem is different depending on the scale of the dataset. To tackle the categorical variable in the dataset, one hot encoding (OHE) is used (Yu, Zhou, Chen, & Lai, 2020). For example, biological gender is commonly prepossessed as $[0,1]$ for male or $[1,0]$ for female, or $[0,1]$ for female or

$[1,0]$ for male. For small or medium scale dataset, OHE is appropriate. However, for large scale dataset, the model matrix generated after OHE could hardly be fit into the RAM.

Consider a linear regression model with $q+1$ variables. q of them are continuous and one of them is categorical. Assume the categorical independent variable is A with I levels (or categories). n_i is the number of samples (or observations) for categorical independent variable level I. Then the linear model with categorical independent variable can be expressed in the form of (2.14).

$$\mathscr{D} = \{y_{ij}, \boldsymbol{x}_{ij}^T, A_i) : i = 1, \ldots, I, j = 1, \ldots, n_i\}, \tag{2.14}$$

where $\boldsymbol{x}_{ij} = (1, x_{i1}, ...x_{i(q-1)})^T$. It stands for the (i, j)th vector of the independent variables.

To handle this, the interaction effects between the categorical independent variables and the independent variables are generally taken into account. Assume the the categorical independent variables interacted with the first q_0 independent variables. Meanwhile the next $q - q_0$ independent variables are not interacted with the categorical variables. ANOCVA model is proposed to tackle this issue based on the dummy variable approach (Fujita, Takahashi, Patriota, & Sato, 2014). The ANOCVA model is shown in (3.19).

$$y_{ij} = \boldsymbol{w}_{ij}^\top \boldsymbol{\alpha} + \boldsymbol{w}_{ij}^\top \boldsymbol{\omega}_i + \boldsymbol{z}_{ij}^\top \boldsymbol{\delta} + \varepsilon_{ij}, \tag{2.15}$$

where $\varepsilon_{ij} \backsim^{iid} \mathscr{N}(0, \sigma^2)$, $\boldsymbol{w}_{ij} = (1, x_{ij1}, \ldots, x_{ij(q_0-1)})^\top$ are the independent variables that are interacted with the categorical variables. $\boldsymbol{z}_{ij} = (1, x_{ijq0}, \ldots, x_{ij(q-1)})^\top$ are the independent variables that are not interacted with the categorical variables.

Let $\boldsymbol{\alpha} = (1, \alpha_0, \ldots, \alpha_{q-1})^\top$, $\boldsymbol{\omega}_1 = \mathbf{0}$, $\omega_i = (1, \omega_{i0}, \ldots, \omega_{i(q_0-1)})^\top, i \neq 1$, and $\boldsymbol{\delta} = (\delta_{q_0}, \ldots, \delta_{q-1})^T$, the matrix can be expressed as in (3.20):

$$\boldsymbol{y}_i = \mathbf{W}_i \boldsymbol{\alpha} + \mathbf{W}_i \boldsymbol{\omega}_i + \mathbf{Z}_i^\top \boldsymbol{\delta} + \boldsymbol{\varepsilon}_i, \tag{2.16}$$

where $\boldsymbol{\varepsilon}_i \backsim^{iid} \mathscr{N}(0, \sigma^2 \mathbf{I}_{n_i})$, $y_i = (y_{i1}, \ldots, y_{in_i})^\top$, $\mathbf{W}_i = (\boldsymbol{w}_{i1}^\top, \ldots, \boldsymbol{w}_{in_i}^\top)^\top$ and $\mathbf{Z}_i = (\boldsymbol{z}_{i1}^\top, ...\boldsymbol{z}_{in_i}^\top)^\top$.

The corresponding model matrix $\mathbf{X}$ is

$$\mathbf{X} = \begin{pmatrix} \mathbf{W}_1 & 0 & \cdots & 0 & \mathbf{Z}_1 \\ \mathbf{W}_2 & \mathbf{W_2} & \cdots & 0 & \mathbf{Z}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{W}_I & 0 & ... & \mathbf{W}_I & \mathbf{Z}_I \end{pmatrix} \tag{2.17}$$

With $\boldsymbol{\beta} = (\boldsymbol{\alpha}^\top, \boldsymbol{\omega}_2^T, \ldots, \boldsymbol{\omega}_I^\top, \boldsymbol{\delta}^\top)^T$. If $\boldsymbol{\omega}_1 = \mathbf{0}$, $\mathbf{X}$ is in full rank, we can apply least square approach to solve it.

This least square approach to solve linear model with categorical independent variables performs well in data of moderately large size. Similar to least square approach for linear model without categorical independent variables, it can hardly be used in big data due to the large scale of model matrix. The size of $\mathbf{X}$ depends on $n(q_0 I + q - q_0)$. It could be much more huge than the size of dataset itself if the number of the levels of the categorical independent variables are large. Even if the number of levels I is only moderately large, the matrix size can explode for big data problems. The generated model matrix size could be more than 1TB for $n \geq 10^7$ after OHE preprocessing.

2.1.6 Lasso Regression

Lasso regression (Tibshirani, 1996) is motivated from ridge regression, but it uses a different penalty term. It is linear regression technique using an ℓ_1 penalty to replace the ℓ_2 penalty. The SSE cost function of lasso regression can be found in (3.35).

$$SSE_{lasso}(\lambda, \boldsymbol{\beta}_\lambda) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_\lambda\|_2^2 + n\lambda \|\boldsymbol{\beta}_\lambda\|_1 \tag{2.18}$$

where λ is the lasso parameter used with $\lambda \geq 0$. In general, lasso regression has no exact solutions. It is usually solved by optimization methods, e.g. coordinate descent (Fu, 1998; Wu & Lange, 2008). However, if the model matrix $\mathbf{X}$ is orthonormal, lasso regression has exact solutions.

The advantage of the lasso regression is that it automatically sets estimators of some independent variables to be 0, leading to an automatic variable selection procedure for linear models. This task cannot be accomplished by the ridge regression model because it provides non-zero estimates for all regression coefficients. Unlike ridge regression, the lasso estimators cannot be solved analytically. Therefore, numerical computations are needed. Many λ values are usually needed in the implementation of lasso regression. Multiple learning has great advantages comparing to individual learning, because it can provide exact solutions for all of the candidates of λ together.

2.1.7 Generalized Linear Models

Generalized linear models (Agresti, 2003) are extended from ordinary linear models. In ordinary linear models, the dependent variable is assumed to be normal. In generalized linear models, the dependent variable can be not normal. For example, linear model with binomial distribution and linear model with Poisson distribution are members of generalized linear models. In generalized linear models, the errors of the dependent variables and independent variables are assumed to follow specific distributions from the exponential family. For example, the linear model with Poisson distribution is assumed to follow the normal distribution, and the linear model with binomial distribution is assumed to follow the binomial distribution.

An exponential family distribution can be written as a probability mass (or density) function, which is shown in (2.19).

$$f(y_i) = exp\left[\frac{y_i\omega_i - b(\omega_i)}{a(\phi)} + c(y_i,\phi)\right] \tag{2.19}$$

where ω_i is a canonical parameter; ϕ is a dispersion parameter. In 2.19, we have $\mu_i \mathrm{E}(y_i) = b'(\omega_i)$ and $\mathrm{V}(y_i) = a(\phi)b''(\omega_i)$. The goal of generalized linear models is to use a set of independent variables to model μ_i. To define a generalized linear model, we need to provide a random component, a linear component, and a link function.

Let $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_n)$ be the vector of linear component. Then, for each η_i, it is modeled by a linear function of independent variable as

$$\eta_i = \boldsymbol{x}_i^\top \boldsymbol{\beta}, \tag{2.20}$$

where $\boldsymbol{\beta}$ is the coefficients. By a link function $g(\cdot)$, the generalized linear model connects the linear component and μ_i as

$$\eta_i = g(\mu_i) = g[b'(\omega_i)] = \boldsymbol{x}_i^\top \boldsymbol{\beta}, i = 1, \ldots, n. \tag{2.21}$$

If $g(\cdot)$ is the canonical link functions, then there is

$$\eta_i = \omega_i = g(\mu_i) = \boldsymbol{x}_i^\top \boldsymbol{\beta} \tag{2.22}$$

Equivalently, the generalized linear model can be expressed as

$$g(\boldsymbol{\mu}) = \mathbf{X}\boldsymbol{\beta} \tag{2.23}$$

with $\boldsymbol{\mu} = \mathrm{E}(\mathbf{y})$.

The variance of the response variable is

$$\hat{V}(y_i) = a(\phi) v(\mu_i) \tag{2.24}$$

with

$$v(\mu_i) = b''\{h^{-1}[g(\mu)]\} \tag{2.25}$$

where $\omega_i = h(\boldsymbol{x}_i^\top \boldsymbol{\beta})$ is the inverse function.

To tackle all the distributions from the exponential family, an iteratively re-weighted least squares (IRWLS) method is proposed to solve generalized linear models. IRWLS is derived from Fisher scoring method (Schworer & Hovey, 2004).

Let $\boldsymbol{\mu}^{(r)} = (\mu_1^{(r)}, \mu_2^{(r)}, \cdots, \mu_n^{(r)})^\top$ and $\boldsymbol{\beta}^{(r)}$ are the rth iterative values, where $\mu_i^{(r)} = g^{-1}(\eta_i^{(r)})$, $\eta_i^{(r)} = \boldsymbol{x}_i^\top \boldsymbol{\beta}^{(r)}$. And let

$$w(\eta) = \frac{(\partial \mu / \partial \eta)^2}{b''[h(\eta)]} \tag{2.26}$$

and

$$\mu(\eta) = \eta + (y - \mu)(\partial \eta / \partial \mu) \tag{2.27}$$

Let

$$\mathbf{W}^{(r)} = diag(w_1^{(r)}, w_2^{(r)}, \cdots, w_n^{(r)}) \tag{2.28}$$

where $w_i^{(r)} = w(\eta_i^{(r)})$ is the working weight matrix, meanwhile we assume $\mu_i^{(r)} = \mu(\eta_i^{(r)})$ as the working response vector. The estimated model coefficients $\hat{\boldsymbol{\beta}}$ is updated by $(\mathbf{X}^\top \mathbf{W}^{(r)} \mathbf{X}) \boldsymbol{\beta}^{(r+1)} = \mathbf{X}^\top \mathbf{W}^{(r)} \boldsymbol{\mu}^{(r)}$ via Fisher-scoring. After $\hat{\boldsymbol{\beta}}$ is derived, $a(\phi)$ is estimated by moment method (McCullagh, 1983) as

$$a(\hat{\phi}) = \frac{1}{n} \sum_{i=1}^{n} \frac{(y_i - \hat{\mu}_i)^2}{b''[h(\boldsymbol{x}_i^\top \hat{\boldsymbol{\beta}})]}. \tag{2.29}$$

Except for the normal case, the estimate of $\boldsymbol{\beta}$ cannot be solved analytically. Therefore, it is important to investigate the IRWLS that we have reviewed above in generalized linear models for big data. As updates of the working weights and the working responses are involved in iterations of IRWLS when the responses is not normal, it is important to specify those for normal and non-normal responses separately.

2.2 Optimization

First-order and higher-order optimization approaches have been progressing and evolving for decades. In this section, we present first-order, higher-order optimization methods that are

proposed by other researchers which have the potential to solve the memory inflation and training inefficiency issues.

2.2.1 First Order Optimization

Gradient descent is a universal method to solve regression models. It targets to find the minimal point of the loss function (objective function) $\mathscr{L}$.

$$\min_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i}^{n} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i) \tag{2.30}$$

where $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ is the output of datapoint i.

Gradient descent optimizes along the gradient direction of $\mathscr{L}(\boldsymbol{\theta})$. It is able to approach the minimum point. In general, the direction is the negative gradient direction. And the amount taken is proportional to the gradient η. Thus, it can also be called steepest descent. This proportion is called learning rate. Taking linear regression as an example, and the loss used is mean square error (MSE). We have

$$L(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i}^{n} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \tag{2.31}$$

and,

$$\boldsymbol{g} = \frac{\partial L}{\partial \boldsymbol{\theta}} = -\frac{1}{n} \sum_{i}^{n} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \boldsymbol{x}_i \tag{2.32}$$

For clarity of presentation, we let $\boldsymbol{\theta}_t$ and $\boldsymbol{g}_t$ denote the parameters $\boldsymbol{\theta}$ and $\boldsymbol{g}$ at timestep t respectively. Then the parameter is updated via

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{g}_t \tag{2.33}$$

Gradient descent is easy to implement, but its convergence rate is inferior to many other first-order and higher-order iterative methods. Another problem of gradient descent is that the parameter updating requires all training samples in one iteration, which indicates gradient descent

also suffers from memory inflation. To overcome this difficulty, gradient descent has parallelizable variants (Dean et al., 2012; Nocedal & Wright, 2006; Recht, Re, Wright, & Niu, 2011). However, it is still infeasible to adopt gradient descent for big data. Then, a gradient descent method with mini-batch emerges.

The gradient descent with mini-batch overcomes the memory inflation by splitting an iteration over dataset into many small batches and performing calculation in small batches. The gradient for mini-batch gradient descent method is:

$$\boldsymbol{g}_t = -\frac{1}{|\mathscr{B}|}\sum_{i\in\mathscr{B}}(y_i - f_{\boldsymbol{\theta}_t}(\boldsymbol{x}_i))\boldsymbol{x}_i \tag{2.34}$$

where $\mathscr{B}$ is a mini-batch with size $|\mathscr{B}|$. When $|\mathscr{B}| = 1$, min-batch downgrades to stochastic gradient descent (SGD). The computation complexity for one iteration for gradient descent is $O(np)$. For gradient descent with mini-batch, the computation complexity of one iteration is reduced to $O(bp)$, and $O(p)$ when it comes to SGD. By gradient descent with mini-batch, parallelization is easier and more feasible than gradient descent. Meanwhile, mini-batch can be used for online learning.

One issue in mini-batch descent is that it introduces additional noises for gradient calculation during training stage. Because of this issue, the algorithm can hardly escape the minimal area when it is near the minimal point. And gradient descent with mini-batch can be trapped in the local minimal point. To escape local minimum, gradient descent with momentum is proposed (Qian, 1999).

The main idea of gradient descent with momentum comes from the mechanics of physics. It takes the previous update direction into consideration when calculating new gradient. The momentum method is able to accelerate the convergence speed. It introduces an updating vector $\boldsymbol{v}_t$ and updates $\boldsymbol{\theta}_t$ via minus the updating vector. $\boldsymbol{v}_t$ is determined by its previous updating vector $\boldsymbol{v}_{t-1}$. The updating rule is shown in (2.35).

$$\boldsymbol{v}_{t+1} = \gamma\boldsymbol{v}_t - \eta\boldsymbol{g}_t \tag{2.35}$$

where γ is the momentum factor. γ is often set to 0.9. Then

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \tag{2.36}$$

Gradient descent with momentum (Qian, 1999) is able to escape from the local minimum and converges faster, however, it is not very smart in picking the direction. And if momentum factor is not appropriately selected, this method may jump out of the optimal point Ruder (2016). Nesterov accelerated gradient is then proposed to improve the traditional momentum method Nesterov (1983). It calculates gradient of $\boldsymbol{\theta}_{t+1}$ based on $L(\boldsymbol{\theta}_t + \mu \boldsymbol{v}_t)$ instead of $L(\boldsymbol{\theta}_t)$.

$$\boldsymbol{v}_{t+1} = \gamma \boldsymbol{v}_t - \eta \frac{\partial L(\boldsymbol{\theta}_t + \mu \boldsymbol{v}_t)}{\partial \boldsymbol{\theta}} \tag{2.37}$$

where $\mu \in [0, 1]$.

From the update rule, we can find that nesterov accelerated gradient provides more gradient information for computation. For better performance of nesterov accelerated gradient, the learning rate decay factor is, in general, used to decrease the learning rate with the number of iteration.

Adagrad is another algorithm that adapt learning rate to the parameters (Duchi, Hazan, & Singer, 2011). That is also the main difference between Adagrad and gradient descent. Adagrad calculates the learning rate dynamically using all the historical gradients information. Assume $\boldsymbol{gs}_t$ is the accumulated gradients at timestep t, we have

$$\begin{aligned} \boldsymbol{gs}_t &= (\sum_{i=1}^{t} \boldsymbol{g}_t^2 + \varepsilon)^{\frac{1}{2}} \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \boldsymbol{g}_t \oslash \boldsymbol{gs}_t \end{aligned} \tag{2.38}$$

where $\boldsymbol{gs}_t$ is the element-wise squared gradients, ε is a smoothing term used to avoid the issue of dividing by zero. and $\oslash$ is element-wise division operator.

Adagrad are faster to converge compared to gradient descent methods without dynamic learning rate. However, the historical gradient information may result in zero learning rate when the accumulated gradient increases.

Adadelta and RMSprop are then proposed to solve this issue (Tieleman & Hinton, 2012; Zeiler, 2012). Instead of accumulating all historical gradients, Adadelta and RMSprop restrict the accumulation window of gradients to w. Adadelta stores w gradients via (2.39).

$$E[\boldsymbol{g}^2]_t = \rho E[\boldsymbol{g}^2]_{t-1} + (1-\rho)\boldsymbol{g}_t^2 \tag{2.39}$$

where ρ is a decay constant, $E[\boldsymbol{g}^2]_t$ represents the expected value of $\boldsymbol{g}_t^2$. Meanwhile, Adadelta also stores another exponentially decaying average for squared parameter updates.

$$E[\Delta\boldsymbol{\theta}^2]_t = \rho E[\Delta\boldsymbol{\theta}^2]_{t-1} + (1-\rho)\Delta\boldsymbol{\theta}_t^2 \tag{2.40}$$

Notably, the inistial exponentially decaying averages $E[\boldsymbol{g}]_0$ and $E[\boldsymbol{\theta}]_0$ are both $\mathbf{0}$. And $E[\Delta\boldsymbol{\theta}^2]$ is unknown before calculation $\boldsymbol{\theta}$, Adadelta uses $\mathbb{E}[\Delta\boldsymbol{\theta}^2]_{t-1}$ instead. The resulting parameter update rule is

$$\begin{aligned}
&\mathrm{RMS}[\boldsymbol{g}]_t = (E[\boldsymbol{g}^2]_t + \varepsilon)^{\frac{1}{2}} \\
&\mathrm{RMS}[\Delta\boldsymbol{\theta}]_{t-1} = (\mathbb{E}[\Delta\boldsymbol{\theta}^2]_{t-1} + \varepsilon)^{\frac{1}{2}} \\
&\Delta\boldsymbol{\theta}_t = -\mathrm{RMS}[\Delta\boldsymbol{\theta}]_{t-1} \oslash \mathrm{RMS}[\boldsymbol{g}]_t \odot \boldsymbol{g}_t \\
&\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t
\end{aligned} \tag{2.41}$$

where RMS is the squared root of mean square error (MSE). For Adadelta, the default learning rate is removed from the updating rules. The strategy used by RMSprop is similar to Adadelta and thus is skipped.

Except Adadelta and RMSprop, Adam (adaptive moment estimation) is the second gradient descent approach, which adapts dynamic rate of learning and momentum (Kingma & Ba, 2015). Adam has two parameters for exponentially decaying averages, one for previous gradients $\boldsymbol{m}_t$ and the other for previous squared gradients $\boldsymbol{u}_t$ as is shown in (2.42).

$$\begin{aligned}
\boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \\
\boldsymbol{u}_t &= \beta_1 \boldsymbol{u}_{t-1} + (1-\beta_1)\boldsymbol{g}_t^2
\end{aligned} \tag{2.42}$$

$\boldsymbol{m}_t$ and $\boldsymbol{u}_t$ with $\boldsymbol{m}_0 = \mathbf{0}, \boldsymbol{u}_0 = \mathbf{0}$ are considered the estimates of mean and variance, of the gradients respectively. However, these two estimates are biased. To compute the unbiased estimates, Adam uses (2.43) to update the parameters.

$$\begin{aligned} \hat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_1^t} \\ \hat{\boldsymbol{u}}_t &= \frac{\boldsymbol{u}_t}{1-\beta_2^t} \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \hat{\boldsymbol{m}}_t \oslash \sqrt{\hat{\boldsymbol{u}}_t + \varepsilon} \end{aligned} \tag{2.43}$$

The authors experimentally show that Adam works well in practice. The experiments also support that Adam can outperform other adaptive learning-methods. The authors extend Adam to Adamax based on the infinity norm. The infinity norm makes Adamax more stable than Adam.

Adam is a rapidly popularized optimization method in deep learning. However, it is shown that the regular momentum can be conceptually and empirically inferior to Nesterov's accelerated gradient. Nadam is then proposed to incorporate the advantages of Nesterov accelerated gradient and Adam. They experimentally show that the convergence speed and the performance of the learned models are improved via Nadam.

It's also proven that the learning rate may become infinitesimally small as training progresses. AMSGrad is then proposed to address this issue (Reddi, Kale, & Kumar, 2019). AMSGrad would borrow $\boldsymbol{u}_{t-1}$ from the past if $\boldsymbol{u}_{t-1} > \boldsymbol{u}_t$ to avoid infinitesimally small learning rate.

$$\boldsymbol{u}_t = \max(\boldsymbol{u}_t, \boldsymbol{u}_{t-1}) \tag{2.44}$$

Besides, AMSGrad changes their learning rate dynamically by timestep t.

$$\eta_t = \frac{\eta}{\sqrt{t}} \tag{2.45}$$

The authors provide a theoretical guarantee of convergence. But, the generalization ability is on unseen data is still similar to that of Adam. This means there is still considerable performance gap between AMSGrad and SGD.

New variant of Adam is proposed called AdaBound (Luo, Xiong, Liu, & Sun, 2019). Inspired by gradient clipping, AdaBound clips the learning rate within a range $[\eta_l, \eta_u]$ to accomplish SGD base on a smooth transition from adaptive approaches. In addition, it provides theoretical proofs of convergence. The experiments show that AdaBound could eliminate the generalization gap.

Coordinate descent (Nesterov, 2012; Wright, 2015) is an optimization technique to hanle cases when gradient descent methods cannot be applied. It is similar to gradient descent, except that it optimizes the coefficients along the coordinate directions one by one.

2.2.2 Higher Order Optimization

Although gradient descent has popularized the first-order optimization, the second-order optimization is better than the first-order optimization w.r.t convergence and stability.

Ordinary least squares (OLS) is a classical and traditional higher-order optimization method. It is widely applied to obtain exact solution for linear regression models by taking advantage of the second-order derivative. The OLS method calculates the exact solutions by solving the normal equation (Kenney & Keeping, 1962). The estimator of coefficients $\hat{\boldsymbol{\theta}}$ is then obtained by (2.46).

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y} \tag{2.46}$$

where $\mathbf{X} = (\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \ldots, \boldsymbol{x}_n^\top)^\top$. It is the model matrix, and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^\top$. It stands for the dependent variable. For the ith datapoint, $\boldsymbol{x}_i = (1, x_{i1}, x_{i2}, \ldots, x_{i(p-1)})^\top$ and $p-1$ is the number of features. If $\mathbf{X}^\top \mathbf{X}$ is not invertible, the normal equation becomes unsolvable. Usually, the generalized inverse Barata and Hussein (2012); Ben-Israel and Greville (2003) can be used in this case. Although OLS is very efficient in obtaining the exact solutions, it also occurs the memory inflation issue. For big data problem, the RAM needed to adopt OLS makes the computation impossible. To overcome the memory inflation and keep the closed-form solutions, the distributed matrix could be applied as a remedy (Moler, 1986). But the training time needed

makes this approach inappropriate. Due to this, the implementation is limited. Another strategy is to maintain an array of sufficient statistics during training as is shown in (2.47).

$$\mathscr{S} = (s_{yy}, \boldsymbol{s}_{xy}, \mathbf{S}_{xx}) = (\sum_{i=1}^{n} s_{yy,i}, \sum_{i=1}^{n} \boldsymbol{s}_{xy,i}, \mathbf{S}_{xx,i}) \tag{2.47}$$

where $s_{yy,i} = y_i^2$. Similarly, $\boldsymbol{s}_{xy,i} = y_i \boldsymbol{x}_i$ is a p-dimensional vector for the ith datapoint. $\mathbf{S}_{xx,i} = \boldsymbol{x}_i \boldsymbol{x}_i^\top$ is a $p \times p$ matrix. After a single-pass through the dataset, the closed-form solutions could be obtained via sufficient statistics (X. Liu, Tang, Huang, Zhang, & Yang, to appear in 2019; Zhang & Yang, 2017a).

$$\hat{\boldsymbol{\theta}} = \mathbf{S}_{xx}^{-1} \boldsymbol{s}_{xy} \tag{2.48}$$

But this strategy can only be applied to regression problems currently.

Newton's approach is another well-known higher-order optimization . It is originally proposed as a root-finding algorithm by taking advantages of the Taylor series. To find a minimal/maximal point, the second derivative is used. Second derivatives makes Newton's approach to find the optimal points more convenient than gradient descent. Besides, Newton's approach has been proven that it has superior convergence rate. The update formula of Newton's approach is shown in (2.49).

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla^2 f(\boldsymbol{\theta})^{-1} \nabla f(\boldsymbol{\theta}) \tag{2.49}$$

where $\nabla^2 f$ is the Hessian of f, η_t is rate of learning, can be chosen by the Wolfe conditions (Raydan, 1997; Thacker, 1989).

Newton's approach converges fast. If the Hessian is not challenging to obtain (Wedderburn, 1974), Newton's approach would become more efficient as opposed to gradient descent. But if the Hessian is hard to obtain, getting the second derivatives are often expensive and sometimes intractable in big data optimization problem.

Quasi-newton approaches were proposed in order to tackle the cases when the Hessian is hard to obtain (Avriel, 2003). Instead of obtaining the inverse of the Hessian, quasi-newton approaches calculate an estimated matrix for the inverse of the Hessian which significantly

reduces the computational load. Because of the approximation of the inverse matrix, quasi-newton approaches are much efficient than Newton's approach, especially for big data (Cichocki, 2014).

DFP is a quasi-newton approach proposed in 1950s (Davidon, 1991; Fletcher & Powell, 1963). Assume $\mathbf{B}_t$ is an approximation of the Hessian at timestep t, and $\mathbf{H}_t = \mathbf{B}_t^{-1}$. Let $\boldsymbol{s}_t = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$, Denote $\boldsymbol{u}_t = \nabla f(\boldsymbol{\theta}_{t+1}) - \nabla f(\boldsymbol{\theta}_t)$. Then

$$\boldsymbol{u}_t = \mathbf{B}_{t+1}\boldsymbol{s}_t \tag{2.50}$$

In this way, the approximate matrix can be calculated via

$$\mathbf{B}_{t+1} = (\mathbf{I} - \frac{\boldsymbol{u}_t \boldsymbol{s}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t})\mathbf{B}_t(\mathbf{I} - \frac{\boldsymbol{s}_t \boldsymbol{u}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t}) + \frac{\boldsymbol{u}_t \boldsymbol{u}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t} \tag{2.51}$$

$\mathbf{H}_{t+1}$ can be updated by

$$\mathbf{H}_{t+1} = \mathbf{H}_t - \frac{\mathbf{H}_t \boldsymbol{u}_t \boldsymbol{u}_t^\top \mathbf{H}_t}{\boldsymbol{u}_t^\top \mathbf{H}_t \boldsymbol{u}_t} + \frac{\boldsymbol{s}_t \boldsymbol{s}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t} \tag{2.52}$$

BFGS is another quasi-newton approaches similar to DEP (Avriel, 2003). In BFGS, the updates of $\mathbf{H}_t$ is obtained by taking the complimentary formula of DFP, thus

$$\mathbf{H}_{t+1} = (\mathbf{I} - \frac{\boldsymbol{s}_t \boldsymbol{u}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t})\mathbf{H}_t(\mathbf{I} - \frac{\boldsymbol{u}_t \boldsymbol{s}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t}) + \frac{\boldsymbol{s}_t \boldsymbol{s}_t^\top}{\boldsymbol{u}_t^\top \boldsymbol{s}_t} \tag{2.53}$$

For high dimensional data, both DEP and BFGS canot solve data optimization problem if data scale is large. This is because the methods store too many matrices to approximate the Hessian matrix which are resource-consuming. L-BFGS is then proposed to address this issue (D. C. Liu & Nocedal, 1989). L-BFGS stores only the $\boldsymbol{s}_t$ and $\boldsymbol{u}_t$ within a time window instead of the approxiamte matrices. It is one of the most commonly used quasi-newton methods in big data (Zaharia et al., 2010). Some researchers incorporate stochastic method with L-BFGS to proposed online-LBFGS to address large scale optimization problem Schraudolph, Yu, and Günter (2007).

2.3 Summary

In this chapter, we discussed the computational challenges. We reviewed the regression models and the optimization methods that were proposed to address the computational challenges.

CHAPTER 3. METHODOLOGY

In chapter 3, we will present how we address the memory inflation issue and training time inefficiency issue aforementioned in the last chapter. How to optimize the ordinary linear models, including linear and weighted linear model, Box-Cox regression, as well as ridge regression, linear model with categorical independent variable, lasso regression and generalized linear models for big data problem will be discussed. Linear and weighted linear model, Box-Cox regression, ridge regression and linear model with categorical variable all have exact solutions. But it is infeasible to load the entire dataset into RAM to obtain the exact solutions for big data. By constructing sufficient statistics arrays, we find that the exact solutions of the models can be accessed with only a single-pass through the datasets. However, for lasso regression and generalized linear models, there are no exact solutions. The methods to solve these two models will also be proposed. Based on the sufficient statistics arrays, multiple learning will also be introduced for the regression models (Joyce & Marjoram, 2008).

3.1 Sufficient Statistics Arrays and Multiple Learning

Regression models, including linear model, ridge regression, that have exact solutions are well studied. The methods to obtain the exact solution for those models can be easily implemented for small datasets as small datasets can be entirely loaded into the RAM at once. Generally, the size of RAM of a personal computer ranges approximately from 8 GBs to 32 GBs. Challenges arise when data are larger than 32 GBs. For linear model, the model matrix would be too large to be fit into the RAM for the computers to compute the exact solutions, let alone the linear model with categorical independent variables. With the preprocessing of OHE, we could not even put a single row of the data into RAM. In this case, calculating the exact solutions is pie in the sky (Jie, Jiahao, Xueqin, Yue, & Jiajun, 2019; Rodríguez, Bautista, Gonzalez, & Escalera, 2018).

We propose multiple learning for ordinary linear models and generalized linear regression models that are designed to handle the memory inflation issue and training time inefficiency issue of calculating the exact solutions. The ultimate goal for us is to make multiple learning as broadly

applicable as other optimization methods in the area of big data. To achieve this target, we proposed multiple learning which are inspired by sufficient statistics arrays from statistics. The idea can be found in the calculation of Monte Carlo (Lindqvist & Taraldsen, 2001). The researchers use sufficient statistics to calculate the estimators of the expectations without visit the dataset twice.

Sufficient statistics array can also be used to calculate the parameters of the models. These include coefficients and variance-covariance matrix. It is derived from the loglikelihood function or SSE cost function. Sufficient statistics array requires only a single visit to the dataset. All the estimators of the models can be obtained without a second visit (Zhang & Yang, 2017a, 2017b).

Our basic idea is to simplify the loglikelihood function of (2.1):

$$\begin{aligned} \mathscr{L}(\boldsymbol{\beta},\sigma^2) = & -\tfrac{n}{2}\log(2\pi\sigma^2) - \tfrac{1}{2\sigma^2}\textstyle\sum_{i=1}^{n}(y_i-\boldsymbol{x}_i\boldsymbol{\beta})^2 \\ = & -\tfrac{n}{2}\log(2\pi\sigma^2) - \tfrac{1}{2\sigma^2}(s_{yy}-2\mathbf{s}'_{xy}\boldsymbol{\beta}+\boldsymbol{\beta}'\mathbf{S}_{xx}\boldsymbol{\beta}), \end{aligned} \tag{3.1}$$

where $s_{yy}=\sum_{i=1}^{n}y_i^2$, $\boldsymbol{s}_{xy}=\sum_{i=1}^{n}\boldsymbol{x}_i y_i$, and $\boldsymbol{S}_{xx}=\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i'$. Note that $\mathscr{L}(\boldsymbol{\beta},\sigma^2)$ is only dependent of s_{yy}, $\boldsymbol{s}_{xy}$, and $\mathbf{S}_{xx}$. They are a scalar, a vector in $\mathbb{R}^p$, and a matrix in $\mathbb{R}^{p\times p}$, respectively. According to the factorization theorem, we conclude that $\{s_{yy},\boldsymbol{s}_{xy},\mathbf{S}_{xx}\}$ is a set of sufficient statistics arrays of (2.1). According to the properties of sufficient statistics arrays, the estimates of $\boldsymbol{\beta}$ and σ^2 are exactly obtained from s_{yy}, $\boldsymbol{s}_{xy}$, and $\mathbf{S}_{xx}$ are available. This is obvious as $\hat{\boldsymbol{\beta}}=\mathbf{S}_{xx}^{-1}\boldsymbol{s}_{xy}$, $\hat{\sigma}^2=(s_{yy}-\boldsymbol{s}'_{xy}\mathbf{S}_{xx}^{-1}\boldsymbol{s}_{xy})/(n-p)$, $\tilde{\sigma}^2=(s_{yy}-\boldsymbol{s}'_{xy}\mathbf{S}_{xx}^{-1}\boldsymbol{s}_{xy})/n$, and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})=\hat{\sigma}^2\mathbf{S}_{xx}^{-1}$. As the sizes of s_{yy}, $\boldsymbol{s}_{xy}$, and $\mathbf{S}_{xx}$ are irrelevant to n, it is possible to propose a method which only uses s_{yy}, $\boldsymbol{s}_{xy}$, and $\mathbf{S}_{xx}$ for estimates of model parameters. In order to have the method, we need to have a way to compute the values of s_{yy}, $\boldsymbol{s}_{xy}$, and $\mathbf{S}_{xx}$.

Taking linear regression as an example, the sufficient statistics arrays to calculate the estimator of the model coefficients (2.3) of comes from the row-independent calculation of $\sum_{i=1}^{n}\boldsymbol{x}_i^\top\boldsymbol{x}_i$.For any two observations $\boldsymbol{x}_{i1}$ and $\boldsymbol{x}_{i2}$, calculating the summation of $\boldsymbol{x}_{i1}^\top\boldsymbol{x}_{i1}$ doesn't depend on $\boldsymbol{x}_{i2}$. As long as $\boldsymbol{x}_{i1}$ can be fit into the RAM, we can calculate $\boldsymbol{x}_{i1}^\top\boldsymbol{x}_{i1}$ and the summation

of $\boldsymbol{x}_{i1}^{\top}\boldsymbol{x}_{i1}$. After that, we can calculate the summation of $\boldsymbol{x}_{i2}^{\top}\boldsymbol{x}_{i2}$, $\boldsymbol{x}_{i3}^{\top}\boldsymbol{x}_{i3}$, $\ldots$, $\boldsymbol{x}_{ij}^{\top}\boldsymbol{x}_{ij}$, $\ldots$, $\boldsymbol{x}_{ij}^{\top}\boldsymbol{x}_{ij}$. The estimator of the model coefficients in (2.3) can be written as in (3.2).

$$\hat{\boldsymbol{\beta}} = \left(\sum_{i=1}^{n} \boldsymbol{x}_i^{\top}\boldsymbol{x}_i\right)^{-1}\left(\sum_{i=1}^{n} \boldsymbol{x}_i y_i\right) \tag{3.2}$$

The variance-covariance matrix is also accessible based on this idea.

Following this strategy, The sufficient statistics array can be defined as:

Definition 3.1.1 *Sufficient statistics array is employed to obtain the model parameters, including coefficients, variance, variance-covariance matrix, and the loglikelihood function (or SSE) with only visiting the dataset once. It is computed row-by-row or batch-by-batch so that the memory inflation issue could be addressed.*

Based on sufficient statistics, the exact solutions can be obtained with only a single visit through the dataset. For regression models, such as ridge regression and Box-Cox regression, a collection of the models given different ridge parameters or power transform parameters can also be solved at once. Obtaining the solutions of models given different hyper-parameters at once is called multiple learning.

In this section, sufficient statistics arrays based multiple learning to obtain the exact solutions for linear and weighted linear model, Box-Cox regression, ridge regression and linear model for categorical independent variables.

Based on the strategy used to derive (2.3), the model parameters $\hat{\boldsymbol{\beta}}$, the variance $\hat{\sigma}^2$ and the variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ can all be obtained.

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \left(\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i\right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}_i y_i\right) \\
\hat{\sigma}^2 &= \frac{1}{n-p}\sum_{i=1}^{n} y_i^2 - \frac{1}{n-p}\left[\left(\sum_{i=1}^{n} \mathbf{x}_i y_i\right)^\top \left(\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i\right) \left(\sum_{i=1}^{n} \mathbf{x}_i y_i\right)\right] \\
\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}) &= \hat{\sigma}^2 \left(\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i\right)^{-1}
\end{aligned} \tag{3.3}$$

Thus, $\mathscr{S}_{lr}$ can be written in the form of sufficient statistics array for linear regression.

$$\mathscr{S}_{lr} = (s_{yy}, \mathbf{s}_{xy}, \mathbf{S}_{xx}) = (\sum_{i=1}^{n} s_{yy,i}, \sum_{i=1}^{n} \mathbf{s}_{xy,i}, \mathbf{S}_{xx,i}) \tag{3.4}$$

where $s_{yy,i} = y_i^2$, $s_{yy,i}$ is a scalar; $\boldsymbol{s}_{xy,i} = \boldsymbol{x}_i y_i$, $\mathbf{s}_{xy,i} \in \mathbb{R}^p$ is a vector; and $\mathbf{S}_{xx,i} = \mathbf{x}_i \mathbf{x}_i^\top$, $\mathbf{S}_{xx,i} \in \mathbb{R}^{p\times p}$ is a matrix.

Based on (3.4), we can obtain the model parameters of $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and the variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ as is shown in (3.5).

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \mathbf{S}_{xx}^{-1}\mathbf{s}_{xy} \\
\hat{\sigma}^2 &= \frac{1}{n-p}(s_{yy} - \mathbf{s}_{xy}^\top \mathbf{S}_{xx}^{-1}\mathbf{s}_{xy}) \\
\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}) &= \hat{\sigma}^2 \mathbf{S}_{xx}^{-1}
\end{aligned} \tag{3.5}$$

Obvious, the computation of the three quantities only need the sufficient statistics array $\mathscr{S}_{lr}$. Once it is available, we do not need to visit the dataset for a second time. Therefore, the loaded dataset can be discarded.

Theorem 3.2.1 *$\mathscr{S}_{lr}$ is the sufficient statistics array for linear model to derive the models parameters $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$, the variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ and the loglikelihood function $\mathscr{L}_{lr}(\boldsymbol{\beta}, \sigma^2)$.*

Algorithm 3.1 Linear Regression with Sufficient Statistics

Input: row-by-row of the entire dataset
Output: $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

$s_{yy} = 0, \boldsymbol{s}_{xy} = \mathbf{0}, \mathbf{S}_{xx} = \mathbf{0}$
for $k \leftarrow 1$ to n **do**
    Compute $s_{yy}^{(i)}, \boldsymbol{s}_{xy}^{(i)}, \mathbf{S}_{xx}^{(i)}$ based on (3.4) for each observations i
    $s_{yy} \mathrel{+}= s_{yy}^{(i)}, \boldsymbol{s}_{xy} \mathrel{+}= \mathbf{s}_{xy}^{(i)}, \mathbf{S}_{xx} \mathrel{+}= \mathbf{S}_{xx}^{(i)}$
end for
if $\mathbf{S}_{xx}$ is invertible **then**
    Compute the generalized inverse of $\mathbf{S}_{xx}^{-1}$
else
    Compute $\mathbf{S}_{xx}^{-1}$
end if
Compute the estimators of the model coefficients $\hat{\boldsymbol{\beta}}$, the variance $\hat{\sigma}^2$ and the variance-covariance matrix $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$ based on (3.5)
return $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

Proof Based on (3.4), the loglikelihood can be written as a function of $\mathscr{S}_{lr}$.

$$\begin{aligned} \mathscr{L}_{lr}(\boldsymbol{\beta}, \sigma^2) &= -\frac{n}{2}\log(2\pi\sigma^2) \\ &- \frac{1}{2\sigma^2}(s_{yy} - 2\mathbf{s}_{xy}^{\top}\boldsymbol{\beta} + \boldsymbol{\beta}^{\top}\mathbf{S}_{xx}\boldsymbol{\beta}) \end{aligned} \tag{3.6}$$

which only depends on the sufficient statistics array for linear regression. ■

The pseudo code of the multiple learning method based linear model can be found in **Algorithm** 3.1. In **Algorithm** 3.1, the sufficient statistics arrays are calculated based on (3.4). If $\mathbf{S}_{xx}$ is invertible, the generalized inverse of $\mathbf{S}_{xx}$ is used. Otherwise, the regular inverse is used. Once $\mathbf{S}_{xx}$ is obtained, the model parameters $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and the variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ based on (3.5).

Approximately 99% memory is reduce compared to the traditional algorithm.

Algorithm 3.2 Linear Regression with Sufficient Statistics

Input: batch-by-batch of the entire dataset
Output: $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

$s_{yy} = 0, \mathbf{s}_{xy} = \mathbf{0}, \mathbf{S}_{xx} = \mathbf{0}$
for $k \leftarrow 1$ to m **do**
    Compute $s_{yy}^{(k)}, \mathbf{s}_{xy}^{(k)}, \mathbf{S}_{xx}^{(k)}$ based on (3.7)
    $s_{yy} \mathrel{+}= s_{yy}^{(k)}, \mathbf{s}_{xy} \mathrel{+}= \mathbf{s}_{xy}^{(k)}, \mathbf{S}_{xx} \mathrel{+}= \mathbf{S}_{xx}^{(k)}$
end for
if $\mathbf{S}_{xx}$ is invertible **then**
    Compute the generalized inverse of $\mathbf{S}_{xx}^{-1}$
else
    Compute $\mathbf{S}_{xx}^{-1}$
end if
Compute the estimators of the model coefficients $\hat{\boldsymbol{\beta}}$, the variance $\hat{\sigma}^2$ and the variance-covariance matrix $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$ based on (3.5)
return $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

To make the computation of the summation of the sufficient statistics array faster, row-by-row computation could be optimized to batch-by-batch computation, i.e. $\sum_{i=1}^{n} y_i^2$, $\sum_{i=1}^{n} \mathbf{x}_i y_i$ and $\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i$ could be re-organized into (3.7).

$$
\begin{aligned}
s_{yy} &= \sum_{k=1}^{m} s_{yy}^{(k)} = \sum_{k=1}^{m} \mathbf{y}_k^\top \mathbf{y}_k \\
\mathbf{s}_{xy} &= \sum_{k=1}^{m} \mathbf{s}_{xy}^{(k)} = \sum_{k=1}^{m} \mathbf{X}_k^\top \mathbf{y}_k \\
\mathbf{S}_{xx} &= \sum_{k=1}^{m} \mathbf{S}_{xx}^{(k)} = \sum_{k=1}^{m} \mathbf{X}_k^\top \mathbf{X}_k
\end{aligned}
\tag{3.7}
$$

where m is the batch size number, $s_{yy}^{(k)}$, $\boldsymbol{s}_{xy}^{(k)}$ and $\mathbf{S}_{xx}^{(k)}$ represents the sufficient statistics array in terms of batch k. $\mathbf{y}_k$ is a m_k-dimensional vector, $\mathbf{X}_k$ is a $m_k \times m_k$ matrix and m_k is the batch size. The pseudo code of the batch-by-batch multiple learning method based linear model can be found in **Algorithm** 3.2. In **Algorithm** 3.2, the sufficient statistics arrays are calculated based on (3.7). Instead of calculating the sufficient statistics arrays row-by-row, the sufficient statistics arrays are calculated batch-by-batch.

3.3 Weighted Linear Model

Weighted linear model uses weights to represent the importance of each sample. The sufficient statistics arrays $\mathbf{S}_{wls}$ for weighted linear model is slightly different compared with linear model aforementioned. The sufficient statistics array is shown in (3.8).

$$\begin{aligned} \mathscr{S}_{wlr} &= (s_{wyy}, \mathbf{s}_{wxy}, \mathbf{S}_{wxx}) \\ &= (\sum_{i=1}^{n} s_{wyy,i}, \sum_{i=1}^{n} \mathbf{s}_{wxy,i}, \mathbf{S}_{wxx,i}) \end{aligned} \tag{3.8}$$

where $s_{wyy,i} = w_i y_i^2$, $s_{wyy,i}$ is scalar; $\boldsymbol{s}_{wxy,i} = \boldsymbol{x}_i w_i y_i$, $\boldsymbol{s}_{wxy,i} \in \mathbb{R}^p$ is a vector; and $\mathbf{S}_{wxx,i} = w_i \mathbf{x}_i \mathbf{x}_i^\top$, $\mathbf{S}_{wxx,i}\mathbb{R}^{p\times[}$ is a matrix.

The the model parameters $\hat{\boldsymbol{\beta}}_w$, the variance $\hat{\sigma}_w^2$ and the variance-covariance matrix of $\hat{\boldsymbol{\beta}}_w$ can be re-expressed as in (3.9).

$$\begin{aligned} \hat{\boldsymbol{\beta}}_w &= \mathbf{S}_{wxx}^{-1} \mathbf{s}_{wxy} \\ \hat{\sigma}_w^2 &= \frac{1}{n}(s_{wyy} - \mathbf{s}_{wxy}^\top \mathbf{S}_{wxx}^{-1} \mathbf{s}_{wxy}) \\ \hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w) &= \hat{\sigma}_w^2 \mathbf{S}_{wxx}^{-1} \end{aligned} \tag{3.9}$$

Theorem 3.3.1 *$\mathscr{S}_{wlr}$ is the sufficient statistics array for weighted linear model to derive the estimators of the model coefficients $\hat{\boldsymbol{\beta}}_w$, the variance $\hat{\sigma}_w^2$ and the variance-covariance matrix $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$.*

Proof From (3.8), (2.6) can be expressed as a function of the sufficient statistics array

$$SSE_{wls}(\boldsymbol{\beta}_w) = s_{wyy} - 2\mathbf{s}_{wxy}^\top \boldsymbol{\beta}_w + \boldsymbol{\beta}_w^\top \mathbf{S}_{wxx} \boldsymbol{\beta}_w \tag{3.10}$$

It can be easily told that (3.10) is only dependent of the sufficient statistics array for weighted linear model. ■

The pseudo code of the multiple learning based weighted linear regression is shown in (3.8). In **Algorithm** 3.3, the sufficient statistics arrays are calculated based on (3.8). If $\mathbf{S}_{wxx}$ is invertible, the generalized inverse of $\mathbf{S}_{wxx}$ is used. Otherwise, the regular inverse is used. Once

Algorithm 3.3 Weighted Linear Regression with Sufficient Statistics

Input: row by row of the entire dataset
Output: $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

$s_{wyy} = 0, \boldsymbol{s}_{wxy} = \mathbf{0}, \mathbf{S}_{wxx} = \mathbf{0}$
for $k \leftarrow 1$ to n **do**
    Compute $s^{(i)}_{wyy}, \mathbf{s}^{(i)}_{wxy}, \mathbf{S}^{(i)}_{wxx}$ based on (3.8) for each observation i
    s_{wyy} += $s^{(k)}_{wyy}, \mathbf{s}_{wxy}$ += $\mathbf{s}^{(k)}_{wxy}, \mathbf{S}_{wxx}$ += $\mathbf{S}^{(k)}_{wxx}$
end for
if $\mathbf{S}_{wxx}$ is inversible **then**
    Compute the generalized inverse of $\mathbf{S}^{-1}_{wxx}$
else
    Compute $\mathbf{S}^{-1}_{wxx}$
end if
Compute the estimators of the model coefficients $\hat{\boldsymbol{\beta}}_w$, the variance $\hat{\sigma}^2_w$ and the variance-covariance $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$ based on (3.9)
return $\hat{\boldsymbol{\beta}}_w$, $\hat{\sigma}^2_w$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$

$\mathbf{S}_{wxx}$ is obtained, the model coefficients $\hat{\boldsymbol{\beta}}_w$, the variance $\hat{\sigma}^2_w$ and the variance-covariance matrix $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$ based on (3.9).

Approximately 99% memory is reduce compared to the traditional algorithm.

Similar to multiple learning for linear model, calculating the sufficient statistics arrays batch-by-batch is also obtainable for weighted linear model.

$$\begin{aligned} s_{wyy} &= \sum_{k=1}^{m} s^{(k)}_{wyy} = \sum_{k=1}^{m} \mathbf{y}_k^\top \mathbf{W}_k \mathbf{y}_k \\ \mathbf{s}_{wxy} &= \sum_{k=1}^{m} \mathbf{s}^{(k)}_{wxy} = \sum_{k=1}^{m} \mathbf{X}_k^\top \mathbf{W}_k \mathbf{y}_k \\ \mathbf{S}_{wxx} &= \sum_{k=1}^{m} \mathbf{S}^{(k)}_{wxx} = \sum_{k=1}^{m} \mathbf{X}_k^\top \mathbf{W}_k \mathbf{X}_k \end{aligned} \tag{3.11}$$

where $\mathbf{W}_k$ is a $m_k \times m_k$ weight matrix for the kth batch.

The multiple learning method based weighted linear model can be found in **Algorithm** 3.4. In **Algorithm** 3.4, the sufficient statistics arrays are calculated based on (3.11). If $\mathbf{S}_{wxx}$ is invertible, the generalized inverse of $\mathbf{S}_{wxx}$ is used. Otherwise, the regular inverse is used. Once

Algorithm 3.4 Weighted Linear Regression with Sufficient Statistics batch-by-batch

Input: batch by batch of the entire dataset
Output: $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

1: $s_{wyy} = 0, \mathbf{s}_{wxy} = \mathbf{0}, \mathbf{S}_{wxx} = \mathbf{0}$
2: **for** $k \leftarrow 1$ to m **do**
3: Compute $s^{(k)}_{wyy}, \mathbf{s}^{(k)}_{wxy}, \mathbf{S}^{(k)}_{wxx}$ based on (3.11)
4: s_{wyy} += $s^{(k)}_{wyy}, \mathbf{s}_{wxy}$ += $\mathbf{s}^{(k)}_{wxy}, \mathbf{S}_{wxx}$ += $\mathbf{S}^{(k)}_{wxx}$
5: **end for**
6: **if** $\mathbf{S}_{wxx}$ is inversible **then**
7: Compute the generalized inverse of $\mathbf{S}^{-1}_{wxx}$
8: **else**
9: Compute $\mathbf{S}^{-1}_{wxx}$
10: **end if**
11: Compute the estimators of the model coefficients $\hat{\boldsymbol{\beta}}_w$, the variance $\hat{\sigma}^2_w$ and the variance-covariance $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$ based on (3.9)
12: **return** $\hat{\boldsymbol{\beta}}_w$, $\hat{\sigma}^2_w$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$

$\mathbf{S}_{wxx}$ is obtained, the model coefficients $\hat{\boldsymbol{\beta}}_w$, the variance $\hat{\sigma}^2_w$ and the variance-covariance matrix $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_w)$ based on (3.9)

In both Algorithms, the data set is accessed from Step 2 to Step 5. The goal is the derivation of the sufficient statistics array. It is used for all of the models in our interest. If the model is changed, we do not need to re-compute the sufficient statistics array. Therefore, the fitting of ML/statistical model given by Step 6 to Step 11 is implemented multiple times, but the access of the data (given by Step 2 to Step 5) is only implemented once. The two algorithms can provide exact solutions to all of the models in our interests simultaneously.

3.4 Box-Cox Regression

Box-Cox regression is linear model with a power transformation on the dependent variable. In general, a collection C of power transformation parameters are used. And the c from a collection of c that has the highest loglikelihood value (2.10) is chosen as the best power transformation parameter. Unlike linear and weighted linear model, the calculation of profile

loglikelihood is required for choosing the best power transformation parameter, $(\mathbf{c}-\mathbf{1})^\top \log \mathbf{y}$ is required for sufficient statistics arrays.

The sufficient statistics arrays for Box-Cox regression can be found in (3.12). For each $c \in C$,

$$\begin{aligned} \mathscr{S}_{c,bc} &= (s_{c,yy}, s_{logy}, \mathbf{s}_{c,xy}, \mathbf{S}_{xx}) \\ &= (\sum_{i=1}^{n} s_{c,yy,i}, \sum_{i=1}^{n} s_{logy,i}, \sum_{i=1}^{n} \mathbf{s}_{c,xy,i}, \mathbf{S}_{xx,i}) \end{aligned} \tag{3.12}$$

where $s_{c,yy,i} = \left(y_i^{(c)}\right)^2$, $s_{logy,i} = \log y_i$. Those two statistics are scalars; $\boldsymbol{s}_{c,xy,i} = \boldsymbol{x}_i y_i^{(c)}$, $\boldsymbol{s}_{c,xy,i} \in \mathbb{R}^p$ is a vector; and $\mathbf{S}_{xx,i} = \mathbf{x}_i \mathbf{x}_i^\top$, $\mathbf{S}_{xx,i} \in \mathbb{R}^{p \times p}$ is a matrix. As $\mathbf{S}_{xx}$ does not contain the power transformation parameter c, it can be used by all the models.

For each $c \in C$,

$$\begin{aligned} \hat{\boldsymbol{\beta}}_c &= \mathbf{S}_{xx}^{-1} \mathbf{s}_{c,xy} \\ \hat{\sigma}_c^2 &= \frac{1}{n}(s_{c,yy} - \mathbf{s}_{c,xy}^\top \mathbf{S}_{xx}^{-1} \mathbf{s}_{c,xy}) \\ \hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_c) &= \hat{\sigma}_c^2 \mathbf{S}_{xx}^{-1} \end{aligned} \tag{3.13}$$

Theorem 3.4.1 *For any $c \in C$, $\mathscr{S}_{c,bc}$ is sufficient statistics arrays for Box-Cox regressoin. It could be employed to calculate the model parameters $\hat{\boldsymbol{\beta}}_c$, the variance $\hat{\sigma}_c^2$, the variance-covariance matrix of $\hat{\boldsymbol{\beta}}_c$ and loglikelihood function $\mathscr{L}_{bc}(c, \boldsymbol{\beta}_c, \sigma_c^2)$.*

Proof Based on (3.13), (2.10) becomes

$$\begin{aligned} \mathscr{L}_{bc}(\boldsymbol{\beta}_c, \sigma_c^2) &= -\frac{n}{2}\log(2\pi\sigma_c^2) \\ &- \frac{1}{2\sigma_c^2}(s_{c,yy} - 2\mathbf{s}_{c,xy}^\top \boldsymbol{\beta}_c + \boldsymbol{\beta}_c^\top \mathbf{S}_{c,xx} \boldsymbol{\beta}_c) + (c-1)s_{logy} \end{aligned} \tag{3.14}$$

which only depends on the sufficient statistics arrays for Box-Cox linear regression. ■

The multiple learning algorithm for Box-Cox regression is presented in **Algorithm** 3.5. In **Algorithm** 3.5, the sufficient statistics arrays are calculated based on (3.12). If $\mathbf{S}_{xx}$ is invertible, the generalized inverse of $\mathbf{S}_{xx}$ is used. Otherwise, the regular inverse is used. Once $\mathbf{S}_{xx}$ is

Algorithm 3.5 Box-Cox Regression with Sufficient Statistics

Input: row by row of the entire dataset
Output: $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$

$\mathbf{S}_{xx} = \mathbf{0}$
for $c \in C$ **do**
    $s_{c,yy} = 0, \boldsymbol{s}_{c,xy} = \mathbf{0}$
end for
for $i \leftarrow 1$ to n **do**
    $s_{logy,i} = \log y_i, \mathbf{S}_{xx,i} = \mathbf{x}_i^\top \mathbf{x}_i$
    $s_{logy} \mathrel{+}= s_{logy,i}, \mathbf{S}_{xx} \mathrel{+}= \mathbf{S}_{xx,i}$
    for $c \in C$ **do**
        $s_{c,yy,i} = \left(y_i^{(c)}\right)^2, \boldsymbol{s}_{c,xy,i} = \boldsymbol{x}_i y_i^{(c)}$
        $s_{c,yy} \mathrel{+}= s_{c,yy,i}, \boldsymbol{s}_{c,xy} \mathrel{+}= \boldsymbol{s}_{c,xy,i}$
    end for
end for
if $\mathbf{S}_{xx}$ is singular **then**
    Compute $\mathbf{S}_{xx}^{-1}$ using generalized inverse
else
    Compute $\mathbf{S}_{xx}^{-1}$
end if
for $c \in C$ **do**
    Compute $\hat{\boldsymbol{\beta}}_c$, $\hat{\sigma}^2_c$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_c)$ based on (3.13)
    Compute $\mathscr{L}_{bc}$ based on (3.14)
end for
return $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$ based on $\mathscr{L}_{bc}$

obtained, the model parameters $\hat{\boldsymbol{\beta}}$, the variance $\hat{\sigma}^2$ and the variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ based on (3.5). Approximately 99% memory is reduce compared to the traditional algorithm.

Batched version of sufficient statistics arrays for any $c \in C$ is shown in (3.15).

$$\begin{aligned} s_{c,yy} &= \sum_{k=1}^{m} s_{c,yy}^{(k)} = \sum_{k=1}^{m} (\mathbf{y}_k^{(c)})^\top \mathbf{y}_k \\ \mathbf{s}_{c,xy} &= \sum_{k=1}^{m} \mathbf{s}_{c,xy}^{(k)} = \sum_{k=1}^{m} \mathbf{X}_k^\top \mathbf{y}_k^{(c)} \\ \mathbf{S}_{xx} &= \sum_{k=1}^{m} \mathbf{S}_{xx}^{(k)} = \sum_{k=1}^{m} \mathbf{X}_k^\top \mathbf{X}_k \end{aligned} \tag{3.15}$$

Algorithm 3.6 Box-Cox Regression with Sufficient Statistics batch-by-batch

Input: batch by batch of the entire dataset
Output: $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$

$\mathbf{S}_{xx} = \mathbf{0}$
for $c \in C$ **do**
    $s_{c,yy} = 0, \mathbf{s}_{c,xy} = \mathbf{0}$
end for
for $k \leftarrow 1$ to m **do**
    Compute $\mathbf{S}_{xx}$ based on (3.15)
    $\mathbf{S}_{xx} \mathrel{+}= \mathbf{S}^{(k)}_{xx}$
    for $c \in C$ **do**
        Compute $s^{(k)}_{c,yy}$ and $\mathbf{s}^{(k)}_{c,xy}$ based on (3.15)
        $s_{c,yy} \mathrel{+}= s^{(k)}_{c,yy}, \mathbf{s}_{c,xy} \mathrel{+}= \mathbf{s}^{(k)}_{c,xy}$
    end for
end for
if $\mathbf{S}_{xx}$ is singular **then**
    Compute $\mathbf{S}^{-1}_{xx}$ using generalized inverse
else
    Compute $\mathbf{S}^{-1}_{xx}$
end if
for $c \in C$ **do**
    Compute $\hat{\boldsymbol{\beta}}_c$, $\hat{\sigma}^2_c$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_c)$ based on (3.13)
    Compute $\mathscr{L}_{bc}$ based on (3.14)
end for
return $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$ based on $\mathscr{L}_{bc}$

where $\mathbf{y}^{(c)}_k$ is a m_k-dimensional vector in batch k.

The pseudo code of the batch-by-batch multiple learning based Box-Cox regression can be found in **Algorithm** 3.6.

3.5 Ridge Regression

Ridge regression is linear model with an ℓ_2 penalty term. Although ridge regression has ridge parameters, it is easy to analytically solve ridge regression model, i.e., it is easy to obtain the exact solutions. And for a collection D of ridge parameters, the sufficient statistics arrays can

be directly used for all ridge parameters. Meanwhile, the sufficient statistics arrays is identical to the sufficient statistics arrays of linear regression.

Let $\mathscr{S}_{ridge} = \mathscr{S}_{lr}$, for each $\lambda \in D$, the model parameters $\hat{\boldsymbol{\beta}}_\lambda$, the variance $\hat{\sigma}^2_\lambda$, the variance-covariance matrix of $\hat{\boldsymbol{\beta}}_\lambda$ and the SSE are shown (3.16).

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_\lambda &= (\mathbf{S}_{xx} + \lambda \mathbf{I})^{-1} \mathbf{s}_{xy} \\
\hat{\sigma}^2_\lambda &= \frac{1}{n}(s_{yy} - \mathbf{s}_{xy}^\top (\mathbf{S}_{xx} + \lambda \mathbf{I})^{-1} \mathbf{s}_{xy}) \\
\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_\lambda) &= \hat{\sigma}^2_\lambda (\mathbf{S}_{xx} + \lambda \mathbf{I})^{-1} \mathbf{S}_{xx} (\mathbf{S}_{xx} + \lambda \mathbf{I})
\end{aligned} \tag{3.16}$$

$$SSE_{ridge}(\lambda, \boldsymbol{\beta}_\lambda) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_\lambda\|_2^2 + n\lambda \|\boldsymbol{\beta}_\lambda\|_2^2 \tag{3.17}$$

The best ridge parameter λ can be chosen by ridge trace (Hoerl & Kennard, 1970).

Theorem 3.5.1 *$\mathscr{S}_{ridge}$ is the sufficient statistics arrays for ridge regression.*

Proof Based on (3.16), (2.12) could be written as

$$\begin{aligned}
SSE_{ridge}(\lambda, \boldsymbol{\beta}_\lambda) = \\
s_{yy} - 2\mathbf{s}_{xy}^\top \boldsymbol{\beta}_\lambda + \boldsymbol{\beta}_\lambda^\top \mathbf{S}_{xx} \boldsymbol{\beta}_\lambda + n\lambda \boldsymbol{\beta}_\lambda^\top \boldsymbol{\beta}_\lambda
\end{aligned} \tag{3.18}$$

It is evident that $\mathscr{S}_{ridge}$ is only dependent of the sufficient statistics arrays of ridge regression. ■

The row-by-row and batch-by-batch multiple learning algorithm for ridge regression is the same as the algorithm of linear model. The pseudo code are shown in **Algorithm** 3.7 and **Algorithm** 3.8. Approximately 99% memory is reduce compared to the traditional algorithm.

3.6 Sparse Block Regression (SBR)

To address the hundred of thousands of levels of the categorical independent variables (or factor variables) in linear model, the general way is to use one hot encoding prepossessing before linear model training. However, for hundred of thousands of levels, the obtained model matrix **X**

Algorithm 3.7 Ridge Regression with Sufficient Statistics

Input: batch-by-batch of the entire dataset
Output: $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$

$s_{yy} = 0, \mathbf{s}_{xy} = \mathbf{0}, \mathbf{S}_{xx} = \mathbf{0}$
for $k \leftarrow 1$ to m **do**
    Compute $s^k_{yy}, \mathbf{s}^k_{xy}, \mathbf{S}^k_{xx}$ based on (3.7)
    s_{yy} += $s^k_{yy}, \mathbf{s}_{xy}$ += $\mathbf{s}^k_{xy}, \mathbf{S}_{xx}$ += $\mathbf{S}^k_{xx}$
end for
for $\lambda \in D$ **do**
    Compute $(\mathbf{S}_{xx} + \lambda \mathbf{I})^{-1}$
    Compute $\hat{\boldsymbol{\beta}}_\lambda$, $\hat{\sigma}^2_\lambda$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_\lambda)$ based on (3.16)
    Compute SSE_{ridge} and ridge trace
end for
return $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$ by ridge trace

Algorithm 3.7 Ridge Regression with Sufficient Statistics

Input: batch-by-batch of the entire dataset
Output: $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$

$s_{yy} = 0, \mathbf{s}_{xy} = \mathbf{0}, \mathbf{S}_{xx} = \mathbf{0}$
for $k \leftarrow 1$ to m **do**
    Compute $s^k_{yy}, \mathbf{s}^k_{xy}, \mathbf{S}^k_{xx}$ based on (3.7)
    s_{yy} += $s^k_{yy}, \mathbf{s}_{xy}$ += $\mathbf{s}^k_{xy}, \mathbf{S}_{xx}$ += $\mathbf{S}^k_{xx}$
end for
for $\lambda \in D$ **do**
    Compute $(\mathbf{S}_{xx} + \lambda \mathbf{I})^{-1}$
    Compute $\hat{\boldsymbol{\beta}}_\lambda$, $\hat{\sigma}^2_\lambda$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_\lambda)$ based on (3.16)
    Compute SSE_{ridge} and ridge trace
end for
return $\hat{\boldsymbol{\beta}}_{best}$, $\hat{\sigma}^2_{best}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}}_{best})$ by ridge trace

could be too large to be loaded RAM. Thus, it would be very challenging to calculate the exact solutions. SBR is proposed to exactly solve linear model with hundred of thousands levels of factor variables.

The fundamental of SBR is to build up a sparse block matrix (SBM) for the categorical variables. SBM is actually a matrix of blocks of sufficient statistics arrays for each level of the categorical variables derived similar to the derivation of sufficient statistics arrays of linear model. Once the SBM is obtained, the estimators of the model parameters of linear model with categorical independent variables can be solved by simple matrix operations. We defined the SBM as follows.

To solve the linear model with categorical independent variables, a general model, ANOCVA model is proposed by Bentler and Bonett (1980) in (3.19) or (3.20):

$$\begin{aligned} &y_{ij} = \boldsymbol{w}_{ij}^{\top}\boldsymbol{\alpha} + \boldsymbol{w}_{ij}^{\top}\boldsymbol{\omega}_i + \boldsymbol{z}_{ij}^{\top}\boldsymbol{\delta} + \boldsymbol{\varepsilon}_{ij} \\ &\boldsymbol{\varepsilon}_{ij} \backsim^{iid} \mathcal{N}(0,\sigma^2), i = 1,\ldots,I, j = 1,\ldots,n_i \end{aligned} \tag{3.19}$$

$$\begin{aligned} &\boldsymbol{y}_i = \mathbf{W}_i\boldsymbol{\alpha} + \mathbf{W}_i\boldsymbol{\omega}_i + \mathbf{Z}_i^{\top}\boldsymbol{\delta} + \boldsymbol{\varepsilon}_i \\ &\boldsymbol{\varepsilon_i} \backsim^{iid} \mathcal{N}(0,\sigma^2\mathbf{I}_{n_i}),\ i = 1,\ldots,I, j = 1,\ldots,n_i \end{aligned} \tag{3.20}$$

To define SBM, we first need re-write (3.19) into (3.21).

$$y_{ij} = \boldsymbol{w}_{ij}^{\top}\boldsymbol{\gamma}_i + \boldsymbol{z}_{ij}^{\top}\boldsymbol{\delta} + \boldsymbol{\varepsilon}_{ij} \tag{3.21}$$

where $\boldsymbol{\varepsilon}_{ij} \overset{iid}{\sim} \mathcal{N}(0,\sigma^2)$, $\boldsymbol{\gamma}_1 = \boldsymbol{\alpha}$ and $\boldsymbol{\gamma}_i = \boldsymbol{\alpha} + \boldsymbol{\omega}_i$ for $i \neq 1$.

Then, for (3.20), we have

$$\boldsymbol{y}_i = \mathbf{W}_i\boldsymbol{\gamma}_i + \mathbf{Z}_i\boldsymbol{\delta} + \boldsymbol{\varepsilon}_i, \tag{3.22}$$

where $\boldsymbol{\varepsilon_i} \overset{iid}{\sim} \mathcal{N}(\mathbf{0},\sigma^2\mathbf{I}_{n_i})$. The model matrix $\mathbf{X}$ can be written as

$$\mathbf{X} = \begin{pmatrix} \mathbf{W}_1 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{Z}_1 \\ \mathbf{0} & \mathbf{W_2} & \cdots & \mathbf{0} & \mathbf{Z}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & ... & \mathbf{W}_I & \mathbf{Z}_I \end{pmatrix} \tag{3.23}$$

with $\boldsymbol{\beta} = (\boldsymbol{\gamma}_1^\top, \ldots, \boldsymbol{\gamma}_I^\top, \boldsymbol{\delta}^\top)^\top$.

Given (3.23), we assume $\boldsymbol{\delta}$ is a constant for all levels of the categorical variable. In other words, only the main effects of $\mathbf{Z}_i$ are included in this linear model, and the interaction effects of $\mathbf{Z}_i$ are not included. Because of the absence of the interaction effects, their significance needs testing.

Consider a interaction effects model,

$$y_{ij} = \boldsymbol{w}_{ij}^\top \boldsymbol{\gamma}_i + \boldsymbol{z}_{ij}^\top \boldsymbol{\delta}_i + \boldsymbol{\varepsilon}_{ij} \tag{3.24}$$

The interaction effects model from (3.24) assumes that $\boldsymbol{\delta}_i$ varies with i. To test if (3.24) can be reduced to (3.21), we have the a hypothesis test as is shown in (3.25).

$$H_0 : \boldsymbol{\delta}_1 = \boldsymbol{\delta}_2 = \cdots = \boldsymbol{\delta}_I \tag{3.25}$$

.

(3.24) is treated as a full model and (3.21) is treated as a reduced model. F-statistic is used to examine H_0. Thanks to SBM, F-statistic can be directly calculated. Once F-statistic is computed, the significance of the interaction effects can be tested.

For the next step, we first need derive the the mathematical formulas of the SBM. For each level $i = 1, \ldots, I$ of the categorical variable, we have

$$\begin{aligned}
s_{yy} &= \sum_{i=1}^{I} \sum_{j=1}^{n_i} y_{ij}^2, \quad \boldsymbol{s}_{i,wy} = \sum_{j=1}^{n_i} y_{ij} \boldsymbol{w}_{ij}, \\
\boldsymbol{s}_{i,zy} &= \sum_{j=1}^{n_i} y_{ij} \boldsymbol{z}_{ij}, \quad \mathbf{S}_{i,wz} = \sum_{j=1}^{n_i} \boldsymbol{w}_{ij} \boldsymbol{z}_{ij}^\top, \\
\mathbf{S}_{i,zz} &= \sum_{j=1}^{n_i} \boldsymbol{z}_{ij} \boldsymbol{z}_{ij}^\top, \quad \mathbf{S}_{i,ww} = \sum_{j=1}^{n_i} \boldsymbol{w}_{ij} \boldsymbol{w}_{ij}^\top.
\end{aligned} \tag{3.26}$$

In (3.26), s_{yy} is a scalar, $\boldsymbol{s}_{i,wy}$ is a q_0-dimensional vector, $\boldsymbol{s}_{i,zy}$ is a $(q-q_0)$dimensional vector. $\mathbf{S}_{i,wz}$ is a $q_0 \times (q-q_0)$ matrix, $\mathbf{S}_{i,zz}$ is a $(q-q_0) \times (q-q_0)$ matrix, and $\mathbf{S}_{i,ww}$ is a $q_0 \times q_0$ matrix. The number of blocks of the SBM is less than $(qI)^2 + qI + 1$. Thus, the memory needed for loading the data and computing (3.21) and (3.24) are significantly reduced.

For each block of sufficient statistics arrays in the SBM, the sufficient statistic arrays are shown in (3.27).

$$\mathscr{S}_i = (s_{yy}, \boldsymbol{s}_{i,wy}, \boldsymbol{s}_{i,zy}, \mathbf{S}_{i,wz}, \mathbf{S}_{i,zz}, \mathbf{S}_{i,ww}) \tag{3.27}$$

It offers a collection of blocks of sufficient statistics as $\{\mathscr{S}_i : i = 1, \ldots, I\}$.

SBM can be defined as follows.

Definition 3.6.1 *A sparse block matrix (SBM) is a matrix of blocks, each block is sufficient statistics arrays for each level of the categorical independent variable.*

Theorem 3.6.1 *SBM can then be written as*

$$\mathbf{S} = \text{diag}(\mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_i, \ldots \mathscr{S}_I). \tag{3.28}$$

It is equivalent to $\mathscr{S} = (\mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_i, \ldots \mathscr{S}_I)$. Both of them can be applied to calculate the estimators of the model.

Proof According to the sufficient statistics arrays, SBM can be expressed as in (3.28). For $i = 1, \ldots, I$, the model coefficients $\boldsymbol{\delta}$ and $\hat{\boldsymbol{\gamma}}_i$, the variance-covariance matrix of $\boldsymbol{\delta}$ $\hat{\mathrm{V}}(\hat{\boldsymbol{\delta}})$ and the loglikelihood can be directly calculated based on the SBM with only a single visit to the dataset. The loglikelihood is shown in (3.29).

$$\begin{aligned}
&\mathscr{L}(\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_I, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_I, \sigma^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\sigma^2 \\
&- \frac{1}{2\sigma^2}\left[\sum_{i=1}^{I}(\boldsymbol{\gamma}_i^\top \mathbf{S}_{i,ww}\boldsymbol{\gamma}_i + 2\boldsymbol{\gamma}_i \mathbf{S}_{i,wz}\boldsymbol{\delta}_i + \boldsymbol{\delta}_i^\top \mathbf{S}_{i,zz}\boldsymbol{\delta}_i)\right] \\
&- \frac{1}{2\sigma^2}\left[s_{yy} - 2\sum_{i=1}^{I}(\boldsymbol{s}_{i,wy}^\top \boldsymbol{\gamma}_i + \boldsymbol{s}_{i,zy}^\top \boldsymbol{\delta}_i)\right].
\end{aligned} \tag{3.29}$$

It is evident that $\mathscr{L}(\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_I, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_I, \sigma^2)$ is only dependent of $\mathscr{S}$. In other wirds, $\mathscr{S}$ is a collection of blocks of sufficient statistics arrays. The model parameters $\boldsymbol{\delta}$, $\hat{\boldsymbol{\gamma}}_i$, the variance-covariance matrix $\boldsymbol{\delta}$ $\hat{\mathrm{V}}(\hat{\boldsymbol{\delta}})$ are also accessible by calculating $\mathscr{S}$. Based on this, we have $\mathbf{S}$ as a SBM. ■

F-statistic can also be computed via SBM for hypothesis test. We can then obtain the corresponding p-value from the F-distributions.

By maximizing (3.29) regarding $\boldsymbol{\gamma}_i$, we can get the estimator of model coefficients $\boldsymbol{\gamma}_i$. For $i = 1, \ldots, I$,

$$\hat{\boldsymbol{\gamma}}_i = \mathbf{S}_{i,ww}^{-1}(\boldsymbol{s}_{i,wy} - \mathbf{S}_{i,wz}\hat{\boldsymbol{\delta}}). \tag{3.30}$$

Let $\mathbf{S}_{zz} = \sum_{i=1}^{I}(\mathbf{S}_{i,zz} - \mathbf{S}_{i,zw}\mathbf{S}_{i,ww}^{-1}\mathbf{S}_{i,wz})$, $\mathbf{S}_{wz} = \sum_{i=1}^{I}(\boldsymbol{s}_{i,zy} - \mathbf{S}_{i,zw}\mathbf{S}_{i,ww}^{-1}\boldsymbol{s}_{i,wy})$, and $\mathbf{S}_{i,zw} = \mathbf{S}_{i,wz}^{\top}$. By maximizing (3.29) regarding $\boldsymbol{\delta}$, we have

$$\hat{\boldsymbol{\delta}} = \mathbf{S}_{zz}^{-1}\mathbf{S}_{wz}. \tag{3.31}$$

Meanwhile, we get

$$\hat{\mathrm{V}}(\hat{\boldsymbol{\delta}}) = \hat{\sigma}^2\mathbf{S}_{zz}^{-1} \tag{3.32}$$

where $\hat{\sigma}^2 = \mathrm{SSE}/[n - q - q_0(I-1)]$. Meanwhile $\mathrm{SSE} = s_{yy} - \sum_{i=1}^{I}\boldsymbol{s}_{i,wy}^{\top}\mathbf{S}_{i,wy}^{-1}\boldsymbol{s}_{i,wy} - \mathbf{S}_{wz}^{\top}\mathbf{S}_{zz}^{-1}\mathbf{S}_{wz}$.

We are able to solve the model parameters the variance-covariance matrices of $\hat{\boldsymbol{\gamma}}_i$. For $i = 1, \ldots, I$.

$$\hat{\mathrm{V}}(\hat{\boldsymbol{\gamma}}_i) = \hat{\sigma}^2(\mathbf{S}_{i,ww}^{-1} + \mathbf{S}_{i,ww}^{-1}\mathbf{S}_{i,wz}\mathbf{S}_{zz}^{-1}\mathbf{S}_{i,zw}\mathbf{S}_{i,ww}^{-1}). \tag{3.33}$$

The pseudo code of SBR is shown in **Algorithm** 3.9. In **Algorithm** 3.9, the sufficient statistics arrays is calculated for each level of the categorical independent variable. Approximately 99% memory is reduce compared to the traditional algorithm.

Batched version of sufficient statistics arrays for any $c \in C$ is shown in (3.34).

Algorithm 3.9 Sparse Block Regression

Input: row-by-row of the entire dataset
Output: $\hat{\boldsymbol{\gamma}}_1, \hat{\boldsymbol{\gamma}}_2, \ldots, \hat{\boldsymbol{\gamma}}_I, \hat{\sigma}^2, \hat{\boldsymbol{\delta}}$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\delta}})$

$s_{yy} = 0, \mathbf{S}_{wy} = \mathbf{0}, \mathbf{S}_{zy} = \mathbf{0}$
$\mathbf{S}_{ww} = \mathbf{0}, \mathbf{S}_{wz} = \mathbf{0}, \mathbf{S}_{zz} = \mathbf{0}$
for $i \leftarrow 0$ to $I-1$ **do**
    $\mathbf{S}_{wy}[i] = \mathbf{0}, \mathbf{S}_{zy}[i] = \mathbf{0}, \mathbf{S}_{ww}[i] = \mathbf{0}$
    $\mathbf{S}_{wz}[i] = \mathbf{0}, \mathbf{S}_{zz}[i] = \mathbf{0}$
end for
for $j \leftarrow 0$ to $n-1$ **do**
    $i \leftarrow$ level of jth observation
    $s_{j,yy} = y_j^2, \boldsymbol{s}_{j,wy} = \boldsymbol{w}_j y_j, \boldsymbol{s}_{j,zy} = \boldsymbol{z}_j y_j$
    $\mathbf{S}_{j,ww} = \boldsymbol{w}_j^\top \boldsymbol{w}_j, \mathbf{S}_{j,wz} = \boldsymbol{w}_j \boldsymbol{z}_j^\top$
    $\mathbf{S}_{j,zz} = \boldsymbol{z}_j \boldsymbol{z}_j^\top$
    $s_{yy} \mathrel{+}= s_{yy,i}, \mathbf{S}_{wy}[i] \mathrel{+}= \mathbf{s}_{j,wy}$
    $\mathbf{S}_{zy}[i] \mathrel{+}= \boldsymbol{s}_{j,zy}, \mathbf{S}_{ww}[i] \mathrel{+}= \mathbf{S}_{j,ww}$
    $\mathbf{S}_{wz}[i] \mathrel{+}= \mathbf{S}_{j,wz}, \mathbf{S}_{zz}[i] \mathrel{+}= \mathbf{S}_{j,zz}$
end for
for $i \leftarrow 0$ to $I-1$ **do**
    if $\mathbf{S}_{ww}[i]$ is not invertible **then**
        Compute $\mathbf{S}_{ww}^{-1}[i]$ using SVD inverse
    else
        Compute $\mathbf{S}_{ww}^{-1}[i]$
    end if
end for
Compute $\hat{\boldsymbol{\gamma}}_i, \hat{\boldsymbol{\delta}}, \hat{\sigma}^2, \hat{\mathrm{V}}(\hat{\boldsymbol{\delta}})$

$$
\begin{aligned}
s_{yy} &= \sum_{i=1}^{I} \sum_{k=1}^{m_i} s_{yy}^{(k)} = \sum_{i=1}^{I} \sum_{k=1}^{m_i} \mathbf{y}_{ik}^\top \mathbf{y}_{ik}, \quad \boldsymbol{s}_{i,wy} = \sum_{k=1}^{m_i} \mathbf{W}_{ik}^\top \boldsymbol{y}_{ik}, \\
\boldsymbol{s}_{i,zy} &= \sum_{k=1}^{m_i} \mathbf{Z}_{ik} \boldsymbol{y}_{ik}, \quad \mathbf{S}_{i,wz} = \sum_{k=1}^{m_i} \mathbf{Z}_{ik}^\top \boldsymbol{W}_{ik}, \\
\mathbf{S}_{i,zz} &= \sum_{k=1}^{m_i} \mathbf{Z}_{ik}^\top \mathbf{Z}_{ik}, \quad \mathbf{S}_{i,ww} = \sum_{k=1}^{m_i} \mathbf{W}_{ik}^\top \mathbf{W}_{ik}.
\end{aligned}
\tag{3.34}
$$

In (3.34), m_i is the total number of batches for level i.

Table 3.1. *An example of converting multiple categorical variables into a single categorical variable.*

State	Major	Converted categorical variable
IN	Math	Category 1
NY	Computer Science	Category 2
IN	Computer Science	Category 3
FL	Physics	Category 4
FL	Physics	Category 4
IN	Math	Category 1

Sparse block regression with multiple categorical variables The proposed SBR algorithm is designed only for datasets with a single categorical independent variables. SBR cannot handle multiple categorical independent variable. Thus, we intend to extend the SBR so that multiple categorical independent variables can be addressed.

The basic idea to handle multiple categorical independent variables is to add a prepossessing step before SBR. In prepossessing step, we convert all categorical independent variables into one categorical independent variable. An example is shown in Table 3.1. The categorical independent variable "State" and categorical independent variable "Major" are converted into a new categorical independent variable with 4 categories.

3.7 Lasso regression

Lasso regression is linear model with an ℓ_1 penalty. The cost function of lasso regression model w.r.t. SSE is shown in (3.35).

$$SSE_{lasso}(\lambda, \boldsymbol{\beta}_\lambda) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_\lambda\|_2^2 + n\lambda \|\boldsymbol{\beta}_\lambda\|_1 \tag{3.35}$$

where λ is the lasso parameter with $\lambda \geq 0$. In general, lasso regression has no exact solutions.

Let $\|\boldsymbol{\beta}_\lambda\|_1 = \sum_i^p |\boldsymbol{\beta}_{\lambda,i}|$, coordinate descent minimizes over $\boldsymbol{\beta}_{\lambda,i}$ with $\boldsymbol{\beta}_{\lambda,j}, j \neq i$ fixed:

$$\mathbf{X}_i^\top \mathbf{X}_i \boldsymbol{\beta}_{\lambda,i} + \mathbf{X}_i^\top (\mathbf{X}_{-i}\boldsymbol{\beta}_{\lambda,-i} - \boldsymbol{y}) + \lambda s_i = 0 \tag{3.36}$$

where $\mathbf{X}_i \in n \times 1$ is the ith column of $\mathbf{X} \in n \times p$. And $\mathbf{X}_{-i} \in n \times p-1$ represents the the dataset $\mathbf{X}$ without the ith column. Similarly, $\boldsymbol{\beta}_{\lambda,i}$ refers to the ith element of $\boldsymbol{\beta}_\lambda$, and $\boldsymbol{\beta}_{\lambda,-i}$ refers to the $\boldsymbol{\beta}_\lambda$ without the ith element. s_i is the subgradient of $|\boldsymbol{\beta}_{\lambda,i}|$ with respect to $\boldsymbol{\beta}_{\lambda,i}$.

$$s_i = \begin{cases} 1 \text{ if } \boldsymbol{\beta}_{\lambda,i} > 0 \\ 0 \text{ if } \boldsymbol{\beta}_{\lambda,i} = 0 \\ -1 \text{ if } \boldsymbol{\beta}_{\lambda,i} < 0 \end{cases} \tag{3.37}$$

The corresponding solution is:

$$\boldsymbol{\beta}_{\lambda,i} = \frac{\mathbf{X}_i^\top (\boldsymbol{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{\lambda,-i}) - \lambda s_i}{\mathbf{X}_i^\top \mathbf{X}_i} \tag{3.38}$$

We then have

$$\boldsymbol{\beta}_{\lambda,i} = \begin{cases} \frac{\mathbf{X}_i^\top (\boldsymbol{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{\lambda,-i}) + \lambda}{\mathbf{X}_i^\top \mathbf{X}_i} \text{ if } \mathbf{X}_i^\top (\boldsymbol{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{\lambda,-i}) < -\lambda \\ \frac{\mathbf{X}_i^\top (\boldsymbol{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{\lambda,-i}) - \lambda}{\mathbf{X}_i^\top \mathbf{X}_i} \text{ if } \mathbf{X}_i^\top (\boldsymbol{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{\lambda,-i}) > \lambda \\ 0 \text{ Otherwise} \end{cases} \tag{3.39}$$

Notably, $\mathbf{X}_i^\top \mathbf{X}_i$, $\mathbf{X}_i^\top \boldsymbol{y}$ and $\mathbf{X}_i^\top \mathbf{X}_{-i}$ can be expressed via the sufficient statistics similar to $\mathbf{X}^\top \mathbf{X}$ and $\mathbf{X}^\top \boldsymbol{y}$ in Chapter 4. Thus, we could calculate the coefficients $\boldsymbol{\beta}_\lambda$ with only a single visit through the dataset. However, due to the limitation of coordinate descent, we need to loop over the computation of $\boldsymbol{\beta}_\lambda$ until the convergence requirements are met. Based on experiments, we found approximately 99% memory is reduce compared to the traditional algorithm.

3.8 Generalized linear regression

Generalized liner models are models extended from linear regression models. Linear regression models assume that the data follow normal distributions, but for the generalized liner models, the data distribution could be different, e.g. Poison distribution. Formally speaking, the

Algorithm 3.10 Generalized Linear Regression with Sufficient Statistics

Input: row by row of the entire dataset
Output: $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

$s_{zz,F}^{(t)} = 0, \boldsymbol{s}_{xy,F}^{(t)} = \mathbf{0}, \mathbf{S}_{xx,F}^{(t)} = \mathbf{0}$
for $i \leftarrow 1$ to n **do**
    $s_{zz,F}^{(t)} \mathrel{+}= w_{F,i}^{(t)} \{z_{F,i}^{(t)}\}^2, \boldsymbol{s}_{xz,F}^{(t)} \mathrel{+}= w_{F,i}^{(t)} z_{F,i}^{(t)} \boldsymbol{x}_i, \mathbf{S}_{xx,F}^{(t)} \mathrel{+}= w_{F,i} \boldsymbol{x}_i \boldsymbol{x}_i^\top$
end for
if $\mathbf{S}_{xx,F}^{(t)}$ is singular **then**
    Compute $\{\mathbf{S}_{xx,F}^{(t)}\}^{-1}$ using generalized inverse
else
    Compute $\{\mathbf{S}_{xx,F}^{(t)}\}^{-1}$
end if
Compute $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$ based on (3.9)
return $\hat{\boldsymbol{\beta}}$, $\hat{\sigma}^2$ and $\hat{\mathrm{V}}(\hat{\boldsymbol{\beta}})$

generalized liner models are defined on exponential family distributions, including Bernoulli, binomial, Poisson distributions and etc. Each target y_i for $i \in [1, \ldots, n]$ is from an exponential family. The loglikelihood of the generalized linear models is:

$$\mathscr{L}_{glm}(\boldsymbol{\beta}, \phi) = \sum_i^n c(y_i, \phi) + \frac{1}{a(\phi)} \sum_i^n \{y_i h(\boldsymbol{x}_i^\top \boldsymbol{\beta}) - b[h(\boldsymbol{x}_i^\top \boldsymbol{\beta})]\} \tag{3.40}$$

The model coefficients $\boldsymbol{\beta}$ is accessible via Newton-Raphson algorithm

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \mathscr{L}_{glm}''^{-1}(\boldsymbol{\beta}^{(t)}) \mathscr{L}_{glm}'^{-1}(\boldsymbol{\beta}^{(t)}) \tag{3.41}$$

where $\boldsymbol{\beta}^{(t)}$ is the estimation of $\boldsymbol{\beta}$ at tth iteration. The Hessian matrix in the equation is very expensive to calculate and can be estimated via the Fisher information matrix, which is $\mathbf{I}(\boldsymbol{\beta}, \phi) = -E(\mathscr{L}_{glm}''(\boldsymbol{\beta}, \phi))$, thus we have

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \mathbf{I}(\boldsymbol{\beta}^{(t)}) \mathscr{L}_{glm}'^{-1}(\boldsymbol{\beta}^{(t)}) \tag{3.42}$$

However, Fisher information matrix is still expensive to obtain. In IRWLS algorithm, the Fisher information matrix is estimated via a weighted least square estimation equation $\mathbf{I}(\boldsymbol{\beta}^{(t)}) = \mathbf{X}^\top \mathbf{W}_F^{(t)} \mathbf{X}$, where $\mathbf{W}_F^{(t)} = \text{diag}(w_{F,1}^{(t)}, w_{F,2}^{(t)}, \ldots, w_{F,n}^{(t)})$.

As $w_{F,i}^{(t)} = \frac{(\partial \mu_i^{(t)} / \partial \eta_i^{(t)})^2}{b''^{[h(\boldsymbol{x}_i^\top \boldsymbol{\beta}^{(t)})]}}$.

Based on this, we get

$$(\mathbf{X}^\top \mathbf{W}_F^{(t)} \mathbf{X}) \boldsymbol{\beta}^{(t+1)} = \mathbf{X}^\top \mathbf{W}_F^{(t)} \boldsymbol{\beta}^{(t)} + \mathscr{L}'^{-1}_{glm}(\boldsymbol{\beta}^{(t)}) \tag{3.43}$$

By setting $\mathscr{L}'^{-1}_{glm}(\boldsymbol{\beta}^{(t)}) = \mathbf{X}^\top \mathbf{W}_F^{(t)} (\boldsymbol{y} - \boldsymbol{u}^{(t)})$, and $\boldsymbol{z}_F^{(t)} = (z_{F,1}^{(t)}, \ldots, z_{F,n}^{(t)})^\top$, we have

$$(\mathbf{X}^\top \mathbf{W}_F^{(t)} \mathbf{X}) \boldsymbol{\beta}^{(t+1)} = \mathbf{X}^\top \mathbf{W}_F^{(t)} \boldsymbol{z}_F^{(t)} \tag{3.44}$$

Therefore, $\boldsymbol{\beta}^{(t+1)}$ is the weighted least squares solution to $\boldsymbol{\beta}$ with

$$\boldsymbol{z}_F^{(t)} = \mathbf{X}\boldsymbol{\beta} + \varepsilon \tag{3.45}$$

where ε follows $N(0, \sigma^2 (\mathbf{W}_F^{(t)})^{-1})$. We can then construct sufficient statistics arrays to obtain $\boldsymbol{\beta}$.

The sufficient statistics arrays under $\boldsymbol{\beta}^{(t)}$ is

$$\mathscr{S}_{glm,F}^{(t)} = (s_{zz,F}^{(t)}, \boldsymbol{s}_{xz,F}^{(t)}, \mathbf{S}_{xx,F}^{(t)}) \tag{3.46}$$

where $s_{zz,F}^{(t)} = \sum_i^n w_{F,i}^{(t)} \{z_{F,i}^{(t)}\}^2$ is a scalar, $\boldsymbol{s}_{xz,F}^{(t)} = \sum_i^n w_{F,i}^{(t)} z_{F,i}^{(t)} \boldsymbol{x}_i$ is a p-dimensional vector, $\mathbf{S}_{xx,F}^{(t)} = \sum_i^n w_{F,i} \boldsymbol{x}_i \boldsymbol{x}_i^\top$ is a $p \times p$ matrix.

Based on $\mathscr{S}_{glm,F}^{(t)}$, we have

$$\begin{aligned} \boldsymbol{\beta}^{(t+1)} &= \{\mathbf{S}_{xx,F}\}^{-1} \boldsymbol{s}_{xz,F}^{(t)} \\ \{\sigma_F^2\}^{(t+1)} &= \frac{1}{n} \{s_{zz,F}^{(t)} - \{\boldsymbol{s}_{xz,F}^{(t)}\}^\top \mathbf{S}_{xx,F}^{(t)}\}^{-1} \boldsymbol{s}_{xz,F}^{(t)}\} \end{aligned} \tag{3.47}$$

The estimation of variance-covariance matrix $\hat{V}(\hat{\boldsymbol{\beta}})$ depends on the presence of ϕ. If ϕ is present, $\hat{V}(\hat{\boldsymbol{\beta}}) = \mathbf{I}^{(-1)}(\hat{\boldsymbol{\beta}}, \hat{\phi})$. For $t+1$th iteration,

$$\hat{V}(\hat{\boldsymbol{\beta}})^{(t+1)} = \{\sigma_F^2\}^{(t)}(\mathbf{X}^\top \mathbf{W}_F^{(t)} \mathbf{X})^{-1} \tag{3.48}$$

Otherwise, $\hat{V}(\hat{\boldsymbol{\beta}}) = \mathbf{I}^{(-1)}(\hat{\boldsymbol{\beta}})$. For $t+1$th iteration,

$$\hat{V}(\hat{\boldsymbol{\beta}})^{(t+1)} = (\mathbf{X}^\top \mathbf{W}_F^{(t)} \mathbf{X})^{-1} \tag{3.49}$$

The row-by-row multiple learning method based generalized linear models can be found in **Algorithm** 3.10 (as introducing new notations for the batch-by-batch version algorithm could be too complicated for the pseudo code, we used pseudo code for row-by-row multiple learning approach here). Approximately 99% memory is reduce compared to the traditional algorithm.

3.9 Summary

In this chapter, we proposed the multiple learning models for linear model, weighted linear model, Box-Cox regression and ridge regression, lasso regression and generalized linear models.

CHAPTER 4. RESULTS FOR REGRESSION MODELS WITH EXACT SOLUTIONS

This chapter illustrates the results of regression models with exact solutions, including linear and weighted linear model, Box-Cox regression and ridge regression. To measure the performance of the four multiple learning method based ordinary linear models and generalized linear models, two set of experiments were conducted.

4.1 Experiment Design

All the multiple learning methods were implemented on Spark and run on a Spark cluster with four nodes. To carry out the experiments on the Spark cluster, a computer cluster with 1 master and 3 workers are built. The hardware specifications of four nodes are shown in Table 4.1.

We mainly designed two set of experiments to evaluate the performance of the traditional regression models and the proposed multiple learning models on Spark for each regression model. The first experiment is to compare the training time cost and the other is to compare accuracy. In the two sets of experiments, we used three datasets with different scales as the train sets. The train sets are 1GB, 10GB, and 100GB datasets. We used 0.2GB data as test set for 1GB train set, 2GB data as test set for 10GB train set, and 20GB data as test set for 100GB train set (the train sets and test sets are generated under the same distribution). The algorithm to generate the train and test sets is shown in **Algorithm** 4.1.

However, for SBR, the algorithm to generate the dataset is slightly different. In order to understand how categorical variables impacts computing, we generated 12 datasets for the experiments. The samples in the datasets contains F independent variables . One of the F features

Table 4.1. *Configurations of Clusters*

	Master	Slave1	Slave2	Slave3
CPU	i7-3770	i7-3770	Quad Q8400	Quad Q9400
Memory	16GB	16GB	4GB	4GB
Disk	1TB	1TB	250GB	250GB

Table 4.2. *Details of Datasets for SBR*

$n_i = 1$		$n_i = 100$	
I	Size	I	Size
1×10^4	1.1MB	1×10^5	123.2MB
5×10^4	5.5MB	5×10^5	620.7.2MB
1×10^5	12.4MB	1×10^5	1.2GB
5×10^5	67.2MB	5×10^5	6.3GB
1×10^6	135.6MB	1×10^6	13.6GB
5×10^6	686.8MB	5×10^6	68.7GB

Algorithm 4.1 Data Simulation Algorithm

Input: data size ds, number of features nf
Output: csv file of simulated data

Float size $fs = 4$ byte
Number of lines $nl = \frac{ds}{fs \times (nf+1)}$
Initialize weights $\boldsymbol{w}$ with 0
for $i \leftarrow 1$ to nf **do**
    $\boldsymbol{w}[i] \sim \mathscr{U}nif(1, 100)$
end for
Initialize errors $\boldsymbol{\varepsilon}$ with 0
for $i \leftarrow 1$ to nl **do**
    $\boldsymbol{\varepsilon} \sim \mathscr{N}(0, \mathbf{I})$
    Initialize observation $\boldsymbol{x}$ and response y with 0
    for $j \leftarrow 1$ to nf **do**
        $\mathbf{x}[j] \sim \mathscr{U}nif(1, 10000)$
    end for
    Generate $\boldsymbol{y}$ for different models
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{\varepsilon}$
    Write $\boldsymbol{x}$ and $\boldsymbol{y}$ as a line to the file
end for

is a categorical independent variable with I levels. For each level, there are n_i observations. The datsets are listed in Table 4.2. Each sample contains 7 features. 6 of the 7 features are continuous independent variables, and 1 of the 7 features is categorical independent variable. 3 of the 7 continuous features are interacted with the categorical feature. For each pair of $\boldsymbol{x}$ and y, the error follows a normal distribution $\mathscr{N}(0, 0.625)$.

Experiment I: Training Time Evaluation

The first experiment is called training time evaluation. It is to measure the time cost for training the multiple learning models. We compared the model training time consumed for each of the traditional models with the multiple learning models. For the multiple learning models, we measured the training time took regarding two different batch sizes 1 and 128.

Experiment II: Accuracy Evaluation

The second experiment is called accuracy evaluation. This experiment is to examine if the results of our multiple learning models are as accurate as the traditional models with a single visit to the datasets. To compare the accuracy, mean squared error (MSE), is used as the metric. The definition of MSE can be found in (4.1).

$$MSE = \frac{\sum_i^n (y_i - \hat{y}_i)^2}{n} \tag{4.1}$$

where y_i is a scalar, it is the ground truth for sample i. $\hat{y}_i$ is also a scalar, it is the prediction. n represents the total number of samples.

4.2 Regression models with exact solutions

Table 4.3 and Table 4.4 showed the results of the two sets of experiments measured on the generated datasets.

Experiment I: Training Time Evaluation

Table 4.3 compared time cost for model training. In Table 4.3, (i) Spark represents Spark built-in implementation of traditional models. Note that Box-Cox regression has no official implementation on Spark, thus we provided an alternative version; (ii) $SS1$ and $SS128$ refer to the multiple learning models are trained with batch size set to 1 and 128 respectively; (iii) $W = \mathbf{I}$ refers to the weights of the samples; (iv) As Box-Cox regression accepts multiple power parameters, a collection $C = [-1.5 \text{ to } 1.5]$ by an interval of 0.1 is selected as the hyper-parameters. For ridge regression, a collection of $D = [0 \text{ to } 0.9]$ is used. A bar chart of training time comparison of ridge regression is also drawn as a figurative illustration example to demonstrate the efficiency of multiple learning models. The figure is shown in Fig. 4.2. The time

cost is in logarithmic scale. The blue bar and purple bar stand for Spark built-in implementation with $D = [0.1]$ and Spark built-in implementation with $D = [0 \text{ to } 0.9]$ respectively. The red bar and green bar stand for multiple learning with $D = [0.1]$ and $D = [0 \text{ to } 0.9]$. The light green bar and green bar stand for multiple learning with $D = [0.1]$ and $D = [0 \text{ to } 0.9]$ and batch size 128.

Based on Table 4.3 and Fig. 4.2, multiple learning models were at least twice efficient than the traditional models. For models with hyper-parameters selected from a collection of models, e.g. ridge regression, multiple learning methods had a great advantage. From Table 4.3, the training time consumed to obtain the desired ridge regression were dependent of the number of ridge parameters. More hyper-parameters we had, longer training need was required. In contrast, the training time of multiple learning models was only increased to a small extent as the number of hyper-parameters increased. According to Table 4.3, multiple learning models for Box-Cox regression and ridge regression with batch size set to 1 were $20x$ efficient than the traditional models on Spark for 10 hyper-parameters.

Larger batch size also reduced the time for training multiple learning models. For batch size set to 128, it could be approximately $27x$ efficient. Comparing the training time of 128 batch size against 1 batch size, the time cost for 1GB, 10GB and 100GB datasets were decreased by approximately 15%, 21%, and 31%, respectively. It could be inferred that larger batch size leads to less training time.

From experiment I, if training models with a collection of hyper-parameters was required for big data, multiple learning models could beat the usual models by only visiting the dataset once, i.e., reducing the disk Inputs/Outputs. This is very useful for training model with a collection of hyper-parameters.

Experiment II: Accuracy Evaluation

Table 4.4 showed the accuracy evaluation, using MSE, for the multiple learning models and the traditional models for 1, 10, and 100 GBs datasets. The MSE of our models was identical to the traditional models. Given the identical MSE, multiple learning models outperformed the traditional models because of faster training time. And the larger the datasets were given, the more advantageous multiple learning methods were.

Table 4.3. *Training time cost for the built-in Apache Spark approaches and the corresponding multiple learning approaches*

Model		**Time Used** (s)		
		1GB	10GB	100GB
LR	Spark	41.86	338.27	3266.16
	SS 1	19.59	154.16	1505.64
	SS 128	15.67	126.33	1267.96
Weighted LR $W = \mathbf{I}$	Spark	42.23	339.54	3263.37
	SS 1	19.76	155.47	1528.75
	SS 128	16.73	125.35	1289.54
Box-Cox $C = [1]$	Spark	42.63	341.31	3264.33
	SS 1	19.16	156.41	1532.00
	SS 128	15.19	122.49	1200.49
Box-Cox $C = [-1.5 \text{ to } 1.5]$	Spark	431.29	3429.34	**33701.51**
	SS 1	19.87	160.13	**1674.62**
	SS 128	16.52	122.21	**1206.17**
Ridge $D = [0.1]$	Spark	41.58	328.48	3276.10
	SS 1	19.87	152.47	1620.46
	SS 128	16.10	127.92	1213.64
Ridge $D = [0 \text{ to } 1.9]$	Spark	423.63	3342.58	**32688.28**
	SS 1	20.56	154.34	**1651.33**
	SS 128	16.80	125.63	**1230.45**

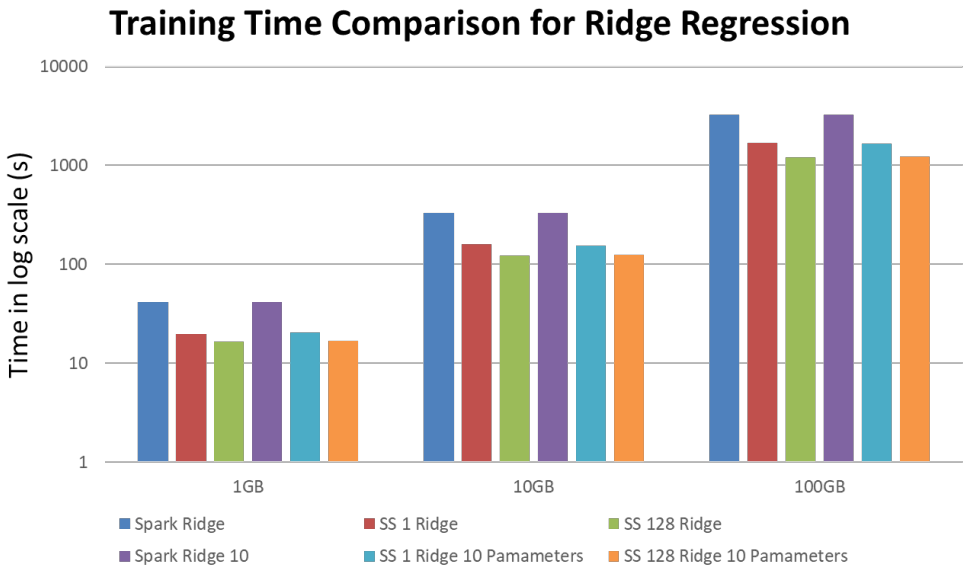

Fig. 4.2. Training time comparison for ridge regression

Table 4.4. *MSE comparison of the built-in Apache Spark approaches and the corresponding multiple learning approaches*

Model		**MSE** (s)		
		1GB	10GB	100GB
LR	Spark	1009520.77	993455.96	994025.56
	SS	1009520.77	993455.96	994025.56
Weighted LR	Spark	1009520.77	993455.96	994025.56
$W = \mathbf{I}$	SS	1009520.77	993455.96	994025.56
Box-Cox	Spark	1138432.54	1053491.23	1011557.43
$C = [1]$	SS	1138432.54	1053491.23	1011557.43
Ridge	Spark	1009520.77	993455.96	994025.56
$D = [0.1]$	SS	1009520.77	993455.96	994025.56

4.3 Spark Block Regression

In Experiment I and II for SBR, the model we employed was evaluated based on the a dataset with only 3 continuous independent variables interacted with the categorical one. Table 4.5 evaluated the training time, while Table 4.6 evaluated MSE.

Experiment I: Training Time Evaluation

Table 4.5 compared training time for SBR and the usual linear model with categorical independent variable given different datasets. In Table 4.5, (i) SBR refers to sparse block regression algorithm (ii) Spark stands for Spark built-in usual linear model for factor variable; (iii) The symbol "-" indicates the model crashed during training without results.

According to Table 4.5, SBR was more efficient than the Spark built-in traditional linear model with categorical variable. For $n_i = 1$ and $n_i = 100$, the training time consumed for SBR increased marginally when I grows from $I = 10^4$ to 10^5. As I grew, the training time consumed was less than 100 seconds even for the factor variable with 5×10^6 level. As for the traditional linear model with categorical variable, the training time cost grew dramatically while I grew. The time consumed for model training grew by over 5 times as I grew from 10^4 to 10^5. For $n_i = 100$, the training time grew by higher than 10 times for I from 10^4 to 5×10^4.

Meanwhile, for categorical independent variable with $\geq 5 \times 10^5$ levels, the Spark built-in traditional linear model with categorical variable crashed without returning the results which was

Table 4.5. *Training time cost for the built-in Apache Spark method and SBR*

$n_i = 1$			$n_i = 100$		
I	SBR	Spark	I	SBR	Spark
1×10^4	2.23s	14.63s	1×10^4	2.98s	108.87s
5×10^4	2.55s	91.69s	5×10^4	9.16s	1028.05s
1×10^5	3.20s	261.39s	1×10^5	21.22s	-
5×10^5	16.28s	-	5×10^5	130.31s	-
1×10^6	34.77s	-	1×10^6	226.44s	-
5×10^6	63.25s	-	5×10^6	879.61s	-

caused by memory inflation. The model matrix $\mathbf{X}$ after OHE beat the largest memory size even if the matrix was sparse, where OHE means one hot encoding.

Based on Experiment I, we made a couple of conclusions. The first conclusion was that SBR outperformed the traditional linear model for categorical in terms of training time. The second conclusion was that SBR could obtain the results while the traditional linear model for factor crashed for a dataset with a factor variable contains hundreds of thousands of levels.

Experiment II: Accuracy Evaluation

Table 4.6 compared the MSE with usual linear model for categorical variable on spark. The experiment was conducted on the 12 dataset. As was shown, the MSE was close to the MSE of traditional model on Spark. For some cases, the MSE was slightly better. For $n_i = 1$, SBR slightly outperformed the traditional linear model for categorical variable on spark. As the number of levels grew, the MSE slightly grew. It is because higher dimensional data were more difficult to fit. For $n_i = 100$, the MSE and the traditional model was equivalent with values are equal to 0.07.

Both training time and MSE experiments provided experimental supports to the proof presented in (3.28) that there are exact solutions for SBR. Meanwhile, the training time cost for SBR was much less than Spark built-in traditional models. Moreover, the higher dimensions the datasets had, the more superior the multiple learning approaches were.

4.4 Summary

We conducted the experiments for all the multiple learning models with exact solutions in this dissertation. We compared the performance regarding time and precision (MSE). The results

Table 4.6. *MSE comparison for the built-in Apache Spark method and SBR*

$n_i = 1$			$n_i = 100$		
I	SBR	Spark	I	SBR	Spark
1×10^4	2.23s	14.63s	1×10^4	2.98s	108.87s
5×10^4	2.55s	91.69s	5×10^4	9.16s	1028.05s
1×10^5	3.20s	261.39s	1×10^5	21.22s	-
5×10^5	16.28s	-	5×10^5	130.31s	-
1×10^6	34.77s	-	1×10^6	226.44s	-
5×10^6	63.25s	-	5×10^6	879.61s	-

demonstrated that our proposed multiple learning models were at least twice time-efficient than the traditional approaches. And we can achieve the same precision (same MSE) compared with the traditional approaches.

CHAPTER 5. RESULTS FOR REGRESSION MODELS WITHOUT EXACT SOLUTIONS

In this chapter, the results of regression models without exact solutions, including lasso regression and generalized linear models are demonstrated.

5.1 Experiment Design

All the algorithms were implemented on the same platform as is shown in the last chapter. The configurations of the platform can be found in Table 4.1.

The algorithm to generate data for lasso regression can be found in **Algorithm** 4.1. The strategy for data generation for generalized linear models is similar to the simulation of data for ordinary linear models, however, the error can be specified to follow a different distribution besides normal distribution as long as the distribution is a member of exponential family. The pesudo code for data generation for generalized linear models is in **Algorithm** 5.1

5.2 Lasso regression

The results of lasso regression for experiment I and II were shown in Table 5.1 and Table 5.2.

Experiment I: Training Time Evaluation

Table 5.1 and Fig. 5.2 compared the training time consumed for lasso regression model. In Table 5.1, (i) Spark represents the built-in lasso regression of Apache Spark; (ii) SS refers to the multiple learning lasso regression; (iii) $D = [0 \text{ to } 0.9]$ by an interval of 0.1 is used for lasso regression.

Based on Table 5.1, the training time consumed of multiple learning lasso regression was twice to three times less than that of the traditional lasso regression on Spark. And for multiple parameters of lasso regression, i.e., $D = [0 \text{ to } 0.9]$ in the table, the computation time needed for multiple learning lasso regression was far less than that of traditional lasso regression. It was because the traditional lasso regression needs to train a different model for a different parameter.

Algorithm 4.1 Data Simulation Algorithm

Input: data size ds, number of features nf
Output: csv file of simulated data

Float size $fs = 4$ byte
Number of lines $nl = \frac{ds}{fs \times (nf+1)}$
Initialize weights $\boldsymbol{w}$ with 0
for $i \leftarrow 1$ to nf **do**
    $\boldsymbol{w}[i] \sim \mathscr{U}nif(1, 100)$
end for
Initialize errors $\boldsymbol{\varepsilon}$ with 0
for $i \leftarrow 1$ to nl **do**
    $\boldsymbol{\varepsilon} \sim \mathscr{N}(0, \mathbf{I})$
    Initialize observation $\boldsymbol{x}$ and response y with 0
    for $j \leftarrow 1$ to nf **do**
        $\mathbf{x}[j] \sim \mathscr{U}nif(1, 10000)$
    end for
    Generate $\boldsymbol{y}$ for different models
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{\varepsilon}$
    Write $\boldsymbol{x}$ and $\boldsymbol{y}$ as a line to the file
end for

Table 5.1. *Training time evaluation of Spark built-in lasso regression and multiple learning lasso regression*

Model		**Time Used** (s)		
		1GB	10GB	100GB
Lasso	Spark	120.32	912.16	10025.10
$D = [0.1]$	SS	60.11	320.88	3253.47
Lasso	Spark	1614.35	11025.83	**71817.28**
$D = [0 \text{ to } 1.9]$	SS	63.53	377.44	**3451.28**

However, for multiple learning lasso regression, the solution for 10 parameters could be obtained at once. Fig. 5.2 visualized the time cost for lasso regression Table 5.3. The time cost is in logarithmic scale. The blue bar and green bar stand for Spark built-in implementation with $D = [0.1]$ and Spark built-in implementation with $D = [0 \text{ to } 0.9]$ respectively. The red bar and purple bar stand for multiple learning with $D = [0.1]$ and $D = [0 \text{ to } 0.9]$. As was shown in Table 5.1 and Fig. 5.2, the multiple learning approach was more time-efficient than the traditional approach.

Table 5.2. *MSE evaluation of Spark built-in lasso regression and multiple learning lasso regression*

Model		**MSE**		
		1GB	10GB	100GB
Lasso	Spark	1013542.15	995474.82	988693.66
$D = [0.1]$	SS	1006862.21	1015256.54	992156.16

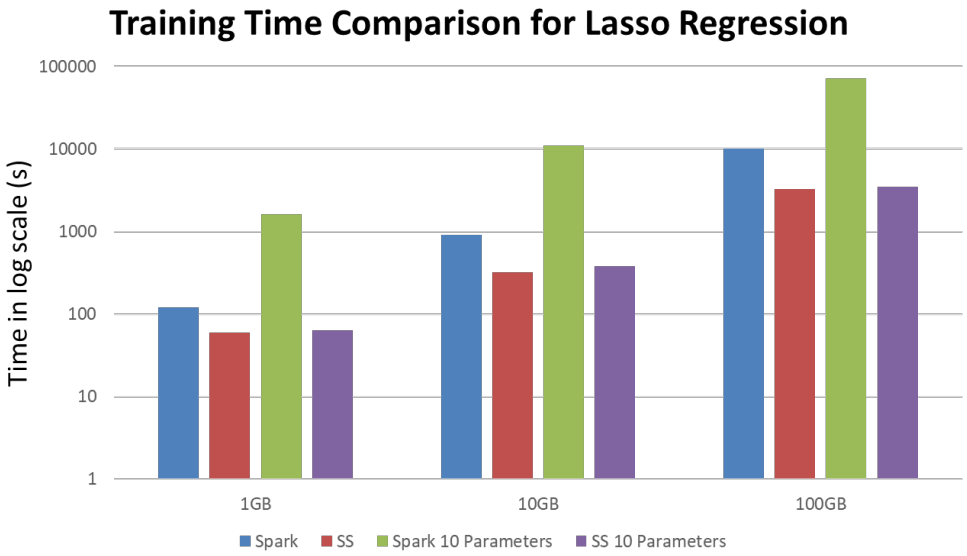

Fig. 5.2. Training time evaluation for lasso regression

Experiment II: Accuracy Evaluation Table 5.2 showed the MSE for the multiple learning lasso regression and the usual approach for 1, 10, and 100 GB datasets. Our approach was slightly better than the built-in spark algorithms as the MSE of our approach was smaller.

5.3 Generalized linear models

The results of the generalized linear models for experiment I and II were shown in Table 5.3 and Table 5.4.

Experiment I: Training time evaluation for traditional approach implement in Spark and the multiple learning approach

Table 5.3 compared training time consumed for the generalized linear models. In Table 5.3, (i) Spark stands for the usual approach in Spark; (ii) SS 1 and SS 128 refer to the multiple

Table 5.3. *Training time evaluation for traditional approach implement in Spark and the multiple learning generalized linear models*

Model		**Time Used** (s)		
		1GB	10GB	100GB
Poisson	Spark	183.45	1328.46	13354.32
	SS 1	80.36	611.58	6024.38
	SS 128	68.42	599.83	5874.87
Normal	Spark	184.64	1341.26	13387.43
	SS 1	81.62	624.11	6101.37
	SS 128	74.5	601.13	5924.74

Table 5.4. *MSE evaluation for traditional approach implement in Spark and the multiple learning generalized linear models*

Model		**MSE**		
		1GB	10GB	100GB
Poisson	Spark	1413894.55	1362749.41	1268031.46
	SS	1403626.46	1345674.41	1245650.46
Normal	Spark	1367863.18	1299342.62	1175912.86
	SS	1405984.18	1300145.62	1192218.86

learning generalized linear models with batch size 1 and 128 respectively. We could find that the multiple learning generalized linear models were time-efficient than the built-in Spark implementation for both both distributions. And if a larger batch size was set, the training time was also slightly reduced. Fig. 5.3 visualized the time cost for generalized linear models with Poisson distribution in Table 5.3. The time cost is in logarithmic scale. The blue bar stands for Spark built-in implementation. The red bar stands for multiple learning generalized linear models with batch size 1. The green bar stands for multiple learning generalized linear models with batch size 128. As was shown in Table 5.3 and Fig. 5.3, the multiple learning approach was more time-efficient than the traditional approach. For multiple learning with batch size 1 and batch size 128, the time consumed for training was similar.

Experiment II: Accuracy Evaluation Table 5.4 showed the MSE for the multiple learning GLMs and the traditional approaches given 1GB, 10GB, and 100GB datasets. Our approach was slightly better than the built-in spark algorithms for both distributions including Poisson and Normal distributions.

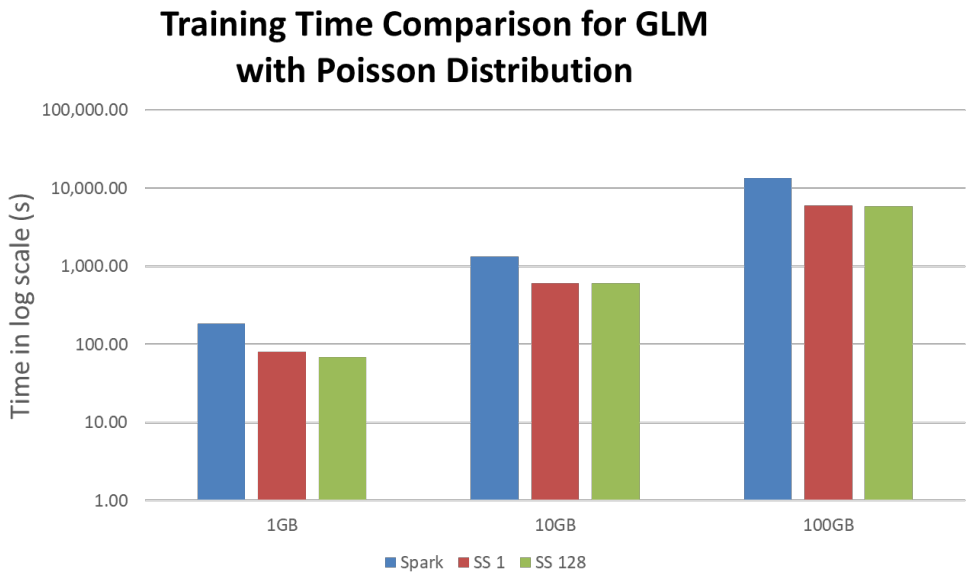

Fig. 5.3. Training time evaluation for generalized linear models

5.4 Summary

In this chapter, we conducted the experiments for all the multiple learning models without exact solutions, including lasso regression and generalized linear models. We compared the performance in terms of time and precision (MSE). The results demonstrated that our proposed multiple learning models were at least three times time-efficient than the traditional lasso regression and generalized linear regression. Meanwhile, the MSE of multiple learning approaches and the traditional approaches was similar.

CHAPTER 6. CONCLUSION AND FUTURE PLAN

6.1 Conclusion

We introduced computational challenges, including memory inflation issue and training time inefficiency issue from training ordinary linear models and generalized linear models for big data and proposed big data optimized algorithms for regression models, including linear and weighted linear model, Box-Cox regression, sparse block regression and generalized linear models. Based on sufficient statistics, we proposed multiple learning for regression models. Multiple learning allows the regression models to obtain the solutions of the models at once. Experiments regarding training time comparison and precision (MSE) comparison for all the multiple learning models are conducted. The results demonstrated that our proposed multiple learning models are at least twice time-efficient than the traditional approaches. And we can achieve the same precision (same MSE) compared with the traditional approaches.

We believe that the multiple learning based ordinary linear models and generalized linear models have huge potentials to be applied in big data to solve related problems. Meanwhile, we think the multiple learning based algorithms can be broadly applicable to not only ordinary linear models and generalized linear models, but also Bayesian model.

6.2 Future Works

Currently, the multiple learning algorithms, including sparse block regression, lasso regression and generalized linear models still require multiple iterations through the dataset to obtain the solutions. Besides, the solutions are not analytical. If we can find methods for sparse block regression, lasso regression and generalized linear models that require only a single-pass through the dataset similar to the other regression models with exact solutions, the training time could be significantly reduced.

For all multiple learning algorithms, those algorithms are limited on the regression models. In the future, we would like to extend the multiple learning algorithms to other models, for example, decision tree and support vector machines.

For the experiments section, it can be improved from the following perspectives.

1. all the experiments can be conducted on computer cluster with high configurations. Due to budget, the computer cluster is in low configuration. In this case, it is hard to compare the performance of multiple learning models with the state-of-art algorithms published in the top conference and journals.

2. the datasets simulated can be more than three types. The datasets we used are 1GB, 10GB abd 100GB. The biggest dataset we used is the 100GB dataset. However, it is still relatively small for big data. In the future, we would like to generate dataset with 1TB, 10TB and 100TB for experiments.

3. there are only two sets of experiments to compare the accuracy and the training time of multiple learning methods and the traditional methods. For each set of the experiment, we only used MSE to evaluate the accuracy and training time to evaluate the efficiency. In the future, we would like to employ more metrics, such as MAE (Mean Absolute Error), RSE (Residual Standard Error), R2 (R square), adjusted R2, AIC (Akaike's Information Criteria), BIC (Bayesian information criteria), Mallows Cp. R square, AIC, BIC and Mallows Cp are metrics generally used in statistics to test the accuracy of regression models. We would like to investigate the disk inputs/outputs as a metric to evaluate the efficiency of the models.

4. For generalized linear models, we only generated datasets from Poisson distribution and normal distribution. However, another important distribution from exponential family is binomial distribution. In the future, we would like to generate datasets from binomial distribution. Accuracy, sensitivity and specificity from classification models will be employed as the metric to measure the performance of generalized linear models with binomial distribution.

5. For lasso regression and ridge regression, we only tested the performance with a collection of 10 hyper-parameters. In the future, we would like to test the performance for 20 or 100 hyper-parameters.

REFERENCES

Abdulla, A. I., Abdulraheem, A. S., Salih, A. A., Sadeeq, M., Ahmed, A. J., Ferzor, B. M., ... Mohammed, S. I. (2020). Internet of things and smart home security. *Technol. Rep. Kansai Univ*, *62*(5), 2465–2476.

Agresti, A. (2003). *Categorical data analysis* (Vol. 482). John Wiley & Sons.

Avriel, M. (2003). *Nonlinear programming: analysis and methods*. Courier Corporation.

Barata, J. C. A., & Hussein, M. S. (2012). The moore–penrose pseudoinverse: A tutorial review of the theory. *Brazilian Journal of Physics*, *42*(1-2), 146–165.

Ben-Israel, A., & Greville, T. N. (2003). *Generalized inverses: theory and applications* (Vol. 15). Springer Science & Business Media.

Bentler, P. M., & Bonett, D. G. (1980). Significance tests and goodness of fit in the analysis of covariance structures. *Psychological bulletin*, *88*(3), 588.

Box, G. E., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, *26*(2), 211–243.

Cichocki, A. (2014). Tensor networks for big data analytics and large-scale optimization problems. *arXiv preprint arXiv:1407.3124*.

Davidon, W. C. (1991). Variable metric method for minimization. *SIAM Journal on Optimization*, *1*(1), 1–17.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... others (2012). Large scale distributed deep networks. In *Advances in neural information processing systems* (pp. 1223–1231).

Demchenko, Y., De Laat, C., & Membrey, P. (2014). Defining architecture components of the big data ecosystem. In *2014 international conference on collaboration technologies and systems (cts)* (pp. 104–112).

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121–2159.

Fletcher, R., & Powell, M. J. (1963). A rapidly convergent descent method for minimization. *The computer journal*, *6*(2), 163–168.

Forsyth, D., & Ponce, J. (2011). *Computer vision: A modern approach.* Prentice hall.

Fu, W. J. (1998). Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, *7*(3), 397–416.

Fujita, A., Takahashi, D. Y., Patriota, A. G., & Sato, J. R. (2014). A non-parametric statistical test to compare clusters with applications in functional magnetic resonance imaging data. *Statistics in medicine*, *33*(28), 4949–4962.

Fürnkranz, J. (2010). Decision tree. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of machine learning* (pp. 263–267). Boston, MA: Springer US. Retrieved from `https://doi.org/10.1007/978-0-387-30164-8_204` doi: 10.1007/978-0-387-30164-8_204

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Jie, L., Jiahao, C., Xueqin, Z., Yue, Z., & Jiajun, L. (2019). One-hot encoding and convolutional neural network based anomaly detection. *Journal of Tsinghua University (Science and Technology)*, *59*(7), 523–529.

Joyce, P., & Marjoram, P. (2008). Approximately sufficient statistics and bayesian computation. *Statistical applications in genetics and molecular biology*, *7*(1).

Kenney, J. F., & Keeping, E. (1962). Linear regression and correlation. *Mathematics of statistics*, *1*, 252–285.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. international conference on learning representations (2015).

Lindqvist, B., & Taraldsen, G. (2001, 09). Monte carlo conditioning on a sufficient statistic. doi: 10.13140/RG.2.2.19200.58885

Liu, D. C., & Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, *45*(1-3), 503–528.

Liu, X., Tang, Z., Huang, H., Zhang, T., & Yang, B. (to appear in 2019). Multiple learning for regression in big data. *18th IEEE International Conference on Machine Learning and Applications (ICMLA)*.

Luo, L., Xiong, Y., Liu, Y., & Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*.

Matt, C., Hess, T., & Benlian, A. (2015). Digital transformation strategies. *Business & Information Systems Engineering*, *57*(5), 339–343.

McCullagh, P. (1983). Quasi-likelihood functions. *The Annals of Statistics*, *11*(1), 59–67.

Meeker, W. Q., & Hong, Y. (2014). Reliability meets big data: opportunities and challenges. *Quality Engineering*, *26*(1), 102–116.

Moler, C. (1986). Matrix computation on distributed memory multiprocessors. *Hypercube Multiprocessors*, *86*(181-195), 31.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr* (Vol. 269, pp. 543–547).

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, *22*(2), 341–362.

Neter, J., Kutner, M. H., Nachtsheim, C. J., Wasserman, W., et al. (1996). Applied linear statistical models.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (second ed.). New York, NY, USA: Springer.

Oussous, A., Benjelloun, F.-Z., Lahcen, A. A., & Belfkih, S. (2018). Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences*, *30*(4), 431–448.

Ponnam, L. T., Rao, V. S., Srinivas, K., & Raavi, V. (2016). A comparative study on techniques used for prediction of stock market. In *2016 international conference on automatic control and dynamic optimization techniques (icacdot)* (pp. 1–6).

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, *12*(1), 145–151.

Raydan, M. (1997). The barzilai and borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal on Optimization*, *7*(1), 26–33.

Recht, B., Re, C., Wright, S., & Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems* (pp. 693–701).

Reddi, S. J., Kale, S., & Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Rodríguez, P., Bautista, M. A., Gonzalez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, *75*, 21–31.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sakia, R. M. (1992). The box-cox transformation technique: a review. *Journal of the Royal Statistical Society: Series D (The Statistician)*, *41*(2), 169–178.

Schraudolph, N. N., Yu, J., & Günter, S. (2007). A stochastic quasi-newton method for online convex optimization. In *Artificial intelligence and statistics* (pp. 436–443).

Schworer, A., & Hovey, P. (2004). Newton-raphson versus fisher scoring algorithms in calculating maximum likelihood estimates.

Thacker, W. C. (1989). The role of the hessian matrix in fitting models to measurements. *Journal of Geophysical Research: Oceans*, *94*(C5), 6177–6196.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

Wedderburn, R. W. (1974). Quasi-likelihood functions, generalized linear models, and the gauss—newton method. *Biometrika*, *61*(3), 439–447.

Wright, S. J. (2015). Coordinate descent algorithms. *Mathematical Programming*, *151*(1), 3–34.

Wu, T. T., & Lange, K. (2008). Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, *2*(1), 224–244.

Xia, F., Yang, L. T., Wang, L., & Vinel, A. (2012). Internet of things. *International journal of communication systems*, *25*(9), 1101.

Yu, L., Zhou, R., Chen, R., & Lai, K. K. (2020). Missing data preprocessing in credit classification: One-hot encoding or imputation? *Emerging Markets Finance and Trade*, 1–11.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. *HotCloud*, *10*(10-10), 95.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, T., & Yang, B. (2017a). Box–cox transformation in big data. *Technometrics*, *59*(2), 189–201.

Zhang, T., & Yang, B. (2017b). An exact approach to ridge regression for big data. *Computational Statistics*, *32*(3), 909–928.