The Pennsylvania State University
The Graduate School
College of Engineering

TEMPORAL AND STRUCTURAL MACHINE LEARNING FROM TRANSPORTATION DATA

A Dissertation in
Computer Science and Engineering
by
Hongyuan Zhan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2019

The dissertation of Hongyuan Zhan was reviewed and approved* by the following:

Kamesh Madduri
Associate Professor of Computer Science and Engineering
Dissertation Advisor, Chair of Committee

Jesse Barlow
Professor of Computer Science and Engineering

Wang-Chien Lee
Associate Professor of Computer Science and Engineering

Necdet Serhat Aybat
Associate Professor of Industrial and Manufacturing Engineering

Kesheng Wu
Senior Staff Scientist, Lawrence Berkeley National Laboratory
Special Member

Xiaoye Sherry Li
Senior Staff Scientist, Lawrence Berkeley National Laboratory
Special Member

Gabriel Gomes
Research Engineer, California Partners for Advanced Transportation Technology, University of California Berkeley
Special Member

Chita R. Das
Professor of Computer Science and Engineering
Graduate Program Chair, Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Transportation is arguably speaking one of the most critical functions of human society. It has been an important societal problem since the ancient age, yet the solution is still far from perfect in the twenty-first century. The needs for efficient and safe transportation are ever-growing, due to prolonging life expectancy and diminishing reserves of fossil fuels which most transportation modes rely on in the present day.

At the same time, we are facing unprecedented growth of data. Can the society utilize data, a cyber-resource, to solve the physical challenges in modern transportation needs? This question motivates the research in my dissertation. Machine learning, broadly speaking, are algorithms that aim to generalize a set of rules from existent data for describing the data generating process, predicting future events, and producing informed decision making. This dissertation studies previous machine learning methods, improves upon them, and develops new algorithms to contribute in essential aspects of transportation systems. Two important topics in transportation systems are addressed in this dissertation, traffic flow prediction and traffic safety analysis.

Traffic flow prediction is a fundamental component in an intelligent transportation system. Accurate traffic predictions are building blocks to achieve efficient routing, smart city planing, reduced energy consumption and among others. Traffic flows are multi-modal and possibly non-stationary due to unusual events. Hence, the learning algorithms for traffic flow prediction need to be robust and adaptive. In addition, the models must be able to learn from latest traffic flow without severely comprising the computational efficiency, in order to meet real-time computation requirements during online deployment. Therefore, learning algorithms for traffic flow prediction developed in this dissertation are designed with the goal to achieve robustness, adaptiveness, and computational efficiency.

Traffic safety in transportation systems is as important as efficiency. Rather than predicting the outcome of crashes, it is more valuable to prevent future accidents by learning from past experiences. The second theme in this dissertation studies machine learning models for analyzing factors contributing to the outcome of crashes. The same accident factor may have diverse degrees of influence on different people, due to the unobserved individual heterogeneity. Capturing heterogeneous effect is difficult in general. A viable approach is to impose structure on the unobserved heterogeneity of different individuals. Under some structural assumptions, it is possible to account for the individual differences with respect to accident factors.

Temporal learning addressed problems arisen from traffic flow prediction. Structural learning is an approach for modeling individual heterogeneity, aiming to quantify the influence of accident factors.

Table of Contents

List of Figures

List of Tables

Acknowledgments

My graduate school journey would not be possible without the help, encouragement, and mental support from many people. I owe to them.

I am grateful to my advisor, Dr. Kamesh Madduri, for offering me the opportunity to pursue my doctoral study. Kamesh is a great supervisor in cruise-controlled style. He gives me the freedom to look for research topics of my interests, but always provides constructive guidance when I seek out for help. I have known Kamesh since I was an undergraduate student. My journey to pursue a doctoral degree in computer science would not be started without him.

I am also thankful for my committee members. I can never save words on praising Dr. Jesse Barlow as the go-to person for questions related to numerical linear algebra, and as a kind senior faculty when I seek out for academic suggestions. Dr. Necdet Serhat Aybat gave me many inspiring lectures on convex optimization. It is always fun to talk to Serhat and learn about new optimization techniques. Dr. Wang-chien Lee provided many practical suggestions for future work.

The collaboration with the special committee members from Berkeley largely shapes the topics in this dissertation. I owe to Dr. Kesheng (John) Wu, Dr. Sherry Xiaoye Li, and Dr. Gabriel Gomes for the good summers in Berkeley. John and Sherry gave me many advices on interdisciplinary research in machine learning. Gabriel introduced me into transportation and to many great people in the PATH institute at UC Berkeley.

I want to express my gratitude to Dr. Venkataraman Shankar at Texas Tech University for providing many valuable insights from transportation econometrics perspective. The projects lead to Chapter 5 and Chapter 6 in this dissertation are initiated from the collaborations with Dr. Shankar.

Graduate school life would be much less fun without my friends and lab mates, I want to thank them.

I am indebted to my parents and grandparents. My dad's entrepreneur mindset constantly provides me the encouragement to pursue my goals. My mom is always proud of me. I am motivated to achieve my career objectives in return for my

parent's unconditional love. I feel guilty to my grandparents for not being able to accompany with them and taking care of them. Not only they have never blamed me, but also being supportive of me for pursuing my goals. I want to thank my wife, Bowang, for staying with me side-by-side and taking care of me in the past few years.

Work performed in this dissertation was supported in part by the U.S. National Science Foundation grants ACI-1253881, OCI-0821527, CCF-1439057, and by the Office of Science of the U.S. Department of Energy under contract No. DE-AC02-05CH11231, and by a Penn State College of Engineering seed grant. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are my own and do not necessarily reflect the views of the U.S. National Science Foundation, the U.S. Department of Energy, and the Pennsylvania State University.

Dedication

I dedicate this dissertation to my father Songmao Zhan and mother Chuanbin Ji. This dissertation is also dedicated to my grandparents for their forever love.

Chapter 1

Introduction

Transportation efficiency and safety impacts life of everyone in the society. Intelligent transportation systems (ITS) aims to develop and improve transportation systems with synergistic technologies and systems engineering principles [1, 2]. The availability of Big Data fosters data-driven methodologies for ITS. Modern sensors generate large amounts of time series data. Large datasets of traffic accident observations also became available. Therefore, motivated from Intelligent Transportation Systems applications, this dissertation studies prediction and structural learning problems on time series and observational data from transportation domain.

The first theme in this dissertation focuses on time series predictions on traffic flow data. Traffic flow prediction is a key component of an Intelligent Transportation System. Accurate traffic flow prediction provides a foundation to other tasks such as signal coordination and travel time forecasting. Sensors are widely deployed to collect real-time traffic information. Traffic flow is noisy, multi-modal, and possibly non-stationary. Moreover, the computation required for making the predictions has to be efficient enough to produce timely forecast. These aspects motivate the design of methodologies in this dissertation. Chapter 3 addresses the robustness aspect for traffic flow prediction by developing an ensemble learning algorithm, which captures multi-modality through the combination of different sub-models, exploits the temporal characteristics of the data, and enhance the robustness through a covariance-regularizer. Learning or parameter inference in machine learning models are often formulated as optimization problems. Nearly all models require users to tune additional parameters, known as hyperparameters. Since the distribution of traffic flow may change gradually, keeping the hyperparameters static may result in sub-optimal performance of the prediction model. This problem is addressed in

Chapter 4. In Chapter 4, hyperparameter tuning is formulated as a bi-level optimization problem [3–8]. An efficient online hyperparameter optimization algorithm is developed for multiple-kernel ridge regression [9, 10]. This method enables fast model tuning to mitigate the impact of non-stationary traffic observations.

The second theme in this dissertation concerns about parameter optimization and structural learning on logistic models with mixed effect (mixed logit) and latent effect. Structural learning refers to generate hypothesis from data with rich internal structure [11]. Mixed logit model is widely used in demand modeling and transportation safety studies [12–18]. It is a flexible model to describe latent heterogeneity across individuals. Traffic safety researchers often apply mixed logit to model the probability of suffering different severity classes in accidents. It allows researchers to impose personalized heterogeneity for each variables. For example, the effect of age on accident severity is often regarded as heterogeneous, since health conditions of people of the same age may vary a lot. In the mixed logit model, structure of the individual heterogeneity is captured by the mixing distribution. The flexibility of mixed logit model comes at the price of solving an optimization problem with high-dimensional integrals. Chapter 5 develops a scalable approach mixed logit parameter learning based on a combination of simulation-based optimization and stochastic gradient method [19–24]. Traditional formulations of mixed logit model is non-convex [12, 15, 16, 25]. Therefore, the flexibility for capturing heterogeneity comes at the cost of less stable estimates due to the inherent non-convexity. A convex alternative to mixed logit model is proposed in Chapter 6, with the goal to capture the data heterogeneity without losing the convexity of a model. The distributional assumptions in the heterogeneity is relaxed. We show that low dimensional structure is a generalized description of the latent effect under several different settings. A convex learning model is proposed via a low-rank and sparse decomposition between the homogeneous and heterogeneous effects in the parameters.

Chapter 2 provides a literature review on statistical learning methods applied to traffic flow prediction.

Chapter 3 studies ensemble learning schemes to integrate different existent models. This chapter aims to improve the accuracy and robustness of traffic flow prediction by model combination. Time series observations are correlated, recent data may be more representative than older ones. Therefore, an ensemble model is proposed in this chapter to exploit the temporal correlation of observations. The predictions from different base models are also intrinsically correlated. However, the correlation between models indicates some redundant information are produced. Hence, another contribution in this chapter is designing a covariance-regularizer to balance the accuracy and diversity of sub-models. Finally, a pruning scheme is applied to remove anomalous predictions from different base models. Experiments show this ensemble learning model outperforms several other model-combination methods for arterial traffic flow prediction.

Chapter 4 studies model hyperparameter selection problems in streaming setting during the online operations of machine learning models. Machine learning models typically have additional hyperparameters to be tuned, before fitting the model on data. Grid search and random search [26] are two most common methods for tuning hyperparameters. During operations, the hyperparameters are tuned sporadically because the tuning process is computationally expensive. However, when the time series is non-stationary or when the characteristics of data change, the model can be sensitive to hyperparameters and requires frequent hyperparameter tuning. Hence, this chapter is motivated to develop an efficient online hyperparameter learning algorithm with theoretical guarantees. In particular, this algorithm is illustrated on multi-kernel ridge regression model [9, 10]. The proposed online algorithm is analyzed under the non-convex regret minimization framework [27–30]. We show this algorithm achieves the optimal local regret for measuring performance of non-convex online optimization algorithms. Experiments on traffic flow prediction demonstrates the proposed online hyperparameter optimization method has significant savings in computation time relative to other hyperparameter selection procedures, while achieving a similar prediction quality.

Chapter 5 studies parameter optimization in mixed logit model under simulation-based inference framework. The computational issues in parameter estimation for mixed logit model is addressed using a two-stage stochastic method. A Stochastic Gradient with Sample Average approximation algorithm is proposed. It first approximates the marginal likelihood estimator with Monte Carlo approximation [15, 25, 31, 32], then applies the Stochastic Gradient Descent algorithm to optimize the approximated objective function. Under the simulation-based framework, many authors solved the simulated problem in the second stage by variants of BFGS algorithm [33–36] or trust-region methods [37]. Stochastic gradient algorithm is a first-order method, which has been shown effective and scalable to solve deterministic finite-sum problems in machine learning [19–24]. This chapter demonstrates that stochastic descent is a scalable alternative to be used inside the simulation-based inference framework as a two-stage stochastic method. This work enables the analysis of factors related to traffic accidents on large datasets.

Chapter 6 introduces a convex formulation for logistic regression with latent effect. The mixed logit model described in the previous chapter leads to a non-convex optimization problem and can be unstable. The goal of the present chapter is to propose an alternative model formulation allowing parameter heterogeneity while preserving convexity. In many big data problems, researchers have discovered that the data can be embedded in intrinsic low dimensional space despite of the high dimensionality of observations [38–42]. It is not surprise to impose low-rank structural assumptions in the latent heterogeneity. We show that low-rankness in the individual heterogeneity is a general result under different data generating process. Therefore, a latent effect logistic model is developed via a decomposition between the sparse common effect and the low-rank heterogeneous effect. The underlying convex optimization problem from the parameter inference is solved by an efficient first-order method using accelerated proximal gradient algorithm [43–47]. We further describe a greedy local continuation scheme to compute the solution of the problem over a path of the hyperparameters to enable efficient exploration.

Chapter 7 and **Chapter 8** consist of my research on network analysis during the early stage of my doctoral study. Transportation data can be represented by networks. For example, traffic flow are collected on network of sensors, therefore traffic flows form multivariate network time series. In addition, the relation between accident observations and attributes can be visualized and analyzed by constructing a bipartite graph, where each node represents either an accident or a factor related to the crash. A matrix factorization method for analyzing clustering structure in networks is proposed in Chapter 7. This method can be used for analyzing location-attribute graph constructed on transportation data. Chapter 8 presents a fast graph sparsification strategy via a knapsack formulation on graph. The knapsack problem is solved by a greedy approximation algorithm. Graph sparsification is helpful for identifying and visualizing structurally important connections on transportation networks.

Table 1.1. Overview of Dissertation Structure

Chapters	Keywords	Motivating applications
Ch. 3	Ensemble Learning Time Series Prediction	Traffic Flow Prediction
Ch. 4	Hyperparameter Optimization Online Learning	Traffic Flow Prediction
Ch. 5	Structural Learning via Latent Variables Simulation-based Optimization Large Scale Optimization	Traffic Accident Analysis Transportation Choice Modeling
Ch. 6	Structural learning via Latent Variables Low-rank and Sparse Decomposition Large Scale Optimization	Traffic Accident Analysis Transportation Choice Modeling
Ch. 7	Matrix Factorization Network Analysis	Data Analysis on Graphs
Ch. 8	Graph Sparsification	Data Analysis on Graphs Data Compression
Ch. 9	Conclusion	

Chapter 2

Review of Statistical Learning for Traffic Flow Prediction

In this chapter, several important statistical learning models for traffic flow prediction are reviewed. These methods are selected representatively from the following categories: 1. time series models 2. latent variable models 3. maximum-margin machine learning methods 4. kernel learning methods 5. Bayesian models.

Let $[y_1, y_2, \cdots, y_{\hat{t}}]$ denote a time series of historical observations of the traffic flow. Predictions for the upcoming measurements $[y_{\hat{t}+1}, y_{\hat{t}+2}, \cdots, y_{\hat{t}+l}]$ are of interest. We assume that there is an underlying autoregressive function f such that

$$\begin{aligned} y_{t+1} &= f(\mathbf{x}_t) + \epsilon_{t+1} \\ \text{where } \mathbf{x}_t &= [y_{t-p+1}, y_{t-p+2}, \cdots, y_t]^T \in \mathbb{R}^p \end{aligned}$$

The measurement y_{t+1} is a mapping from past values with additive i.i.d. Gaussian white noise $\epsilon_{t+1} \sim \mathcal{N}(0, \sigma_\epsilon^2)$. The function f and its order p is unknown. The following notations are used in this chapter.

$$\begin{aligned} \mathbf{x}_t &= [y_{t-p+1}, \cdots, y_t]^T \in \mathbb{R}^p \\ X &= [\mathbf{x}_{\hat{t}-T-1}, \cdots, \mathbf{x}_{\hat{t}-1}]^T \in \mathbb{R}^{T \times p} \\ \mathbf{y}_t &= [y_{t+1}, \cdots, y_{t+l}]^T \in \mathbb{R}^T \\ Y &= [\mathbf{y}_{\hat{t}-l-T+1}, \mathbf{y}_{\hat{t}-l-T+2}, \cdots, \mathbf{y}_{\hat{t}-l}]^T \in \mathbb{R}^{T \times l}. \end{aligned}$$

(X, Y) comprises the training data for each model. A row of X uses the p past observations as the explanatory variables, and the responses are collected in Y. T

Table 2.1. Notation used for different time indices. All variables are positive integers.

Variable	**Description**
t	Indices for time
$\hat{t}$	Present time
l	maximum forecasting horizon
p	Dimension of $\mathbf{x}_t$
T	Number of past observations used for training

denotes the number of flow samples used for training. Also let $\{X_k\}_{k=1,\cdots,p} \in \mathbb{R}^T$ be columns of X, and $\{Y_k\}_{k=1,\cdots,l} \in \mathbb{R}^T$ be columns of Y. The notations for time indices used in the subsequent sections are given in Table 2.1.

There are two main paradigms for multi-step ahead forecasts, the recursive strategy and direct strategy [48]. In the recursive forecasting paradigm, a prediction model $f(\mathbf{x}_{t-1}; \Theta)$ with parameter $\Theta \in \mathbb{R}^d$ is learned to approximate y_t. The parameter Θ is obtained typically by solving

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \sum_{t=\hat{t}-T}^{\hat{t}} \ell\left(y_t, f(\mathbf{x}_{t-1}; \Theta)\right) + \lambda \mathcal{R}(\Theta) \tag{2.1}$$

where $\ell : \mathbb{R} \to \mathbb{R}$ is a loss function on the prediction, $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$ is a regularizer on Θ to penalize the complexity of Θ and prevent overfitting. $\lambda > 0$ is a constant balances the trade-offs between training loss and model complexity. In recursive prediction strategies, after obtaining Θ^*, the h-step-ahead from now forecast is computed by applying model

$$f(\widehat{\mathbf{x}}_{\hat{t}+h}; \Theta^*), \tag{2.2}$$

where $\widehat{\mathbf{x}}_{\hat{t}+h} \in \mathbb{R}^p$ is constructed recursively by

$$\widehat{\mathbf{x}}_{\hat{t}+h} := [y_{\hat{t}-p+1}, y_{\hat{t}-p+2}, \cdots, y_{\hat{t}}, f(\widehat{\mathbf{x}}_{\hat{t}}; \Theta^*), \cdots, f(\widehat{\mathbf{x}}_{\hat{t}+h-2}; \Theta^*)]^T. \tag{2.3}$$

In the recursive scheme, the unknown values from $\hat{t}$ to $\hat{t}+h-1$ are replaced by predictions. Hence, the accumulated prediction error grows exponentially. In contrast to recursive methods, the direct prediction strategies train a model $f_h(\mathbf{x}_{t-1}; \Theta)$ for each forecasting horizon h. For every $1 \leq h \leq l$, the model parameters are learned by

$$\Theta_h^* = \underset{\Theta}{\operatorname{argmin}} \sum_{t=\hat{t}-T}^{\hat{t}} \ell\left(y_t, f(\mathbf{x}_{t-h}; \Theta)\right) + \lambda \mathcal{R}(\Theta) \tag{2.4}$$

An h-step-ahead from now prediction is produced by $f(\mathbf{x}_{\hat{t}-h}; \Theta_h^*)$. Direct predictions provide flexible ways to model multi-step ahead values and possibly avoid exponentially growing prediction error, at the cost of training multiple models.

2.1 Recursive ARMAX

The AutoRegressive-Moving-Average model with eXogenous inputs (ARMAX) for traffic flow prediction was studied in [49]. The ARMAX model describes the evolution of traffic flows over time via the stochastic difference equation

$$A(q^{-1})y_t = B(q^{-1})u_t + C(q^{-1})w_t, \tag{2.5}$$

where y_t is the traffic flow at time step t, u_t is the historical sample average flow value at the same time of day. w_t is assumed to be a zero-mean innovation sequence such that $E\left(w_t w_{t-j}\right) = 0$ for all $0 \leq j \leq t$. Here q^{-1} is the backward shift operator defined by $q^{-1}y_t = y_{t-1}$, $A(q^{-1}), B(q^{-1}), C(q^{-1})$ are scalar polynomials in the backward shift operators

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a} \\ B(q^{-1}) &= 1 + b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b} \\ C(q^{-1}) &= 1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}, \end{aligned}$$

where the order n_a, n_b, n_c are hyperparameters. Coefficients of the polynomial are estimated via the recursive least squares adaptation algorithm [49,50]. This method belongs to the recursive estimation paradigm.

2.2 Partial Least Squares

Coogan et al. [51] recently applied the Partial Least Squares (PLS) technique to short-term traffic flow prediction. The key idea of PLS is to maximally exploit the covariance between flows in the prediction horizon and flows in the memory window. Let $\bar{\mathbf{x}} \in \mathbb{R}^p$ be the sample mean in X, and $\bar{\mathbf{y}} \in \mathbb{R}^l$ be the sample mean in Y. Subtract $\bar{\mathbf{x}}^T$ from each row of X and denote the result as $\tilde{X} \in \mathbb{R}^{T\times p}$. Similarly, denote $\tilde{Y} \in \mathbb{R}^{T\times l}$ the matrix obtained by removing the sample mean $\bar{\mathbf{y}}^T$ from each row of Y. PLS exploits covariance by finding a pair of vectors $(\mathbf{r}^*, \mathbf{s}^*) \in \mathbb{R}^p \times \mathbb{R}^l$, such that

$$(\mathbf{r}^*,\ \mathbf{s}^*) = \underset{(\mathbf{r},\ \mathbf{s})}{\operatorname{argmax}}\, \mathbf{r}^T(\tilde{X}^T\tilde{Y})\mathbf{s}, \quad \text{s. t.} \quad \|\mathbf{r}\|_2^2 = \|\mathbf{s}\|_2^2 = 1, \tag{2.6}$$

Notice that $\tilde{X}^T\tilde{Y} \in \mathbb{R}^{p\times l}$ is the sample covariance matrix of flows across different times. Intuitively, we seek a pair of projection directions $(\mathbf{r}^*,\ \mathbf{s}^*)$ in Eq. (2.6), which maximizes the sample covariance after the projection. The optimization problem in Eq. (2.6) could be solved by a partial SVD of $\tilde{X}^T\tilde{Y}$; the optimal projection direction $(\mathbf{r}^*,\ \mathbf{s}^*)$ are the first left and right singular vectors respectively. Define $\mathbf{w} := \tilde{X}\mathbf{r}^* \in \mathbb{R}^T$. The orthogonal projection of column vector $\tilde{X}_k$ and $\tilde{Y}_k$ onto $\mathbf{w}$ are

$$\begin{aligned} p_k &:= \frac{\langle \tilde{X}_k, \mathbf{w}\rangle}{\|w\|_2^2}, \quad k = 1, \cdots, p \\ c_k &:= \frac{\langle \tilde{Y}_k, \mathbf{w}\rangle}{\|w\|_2^2}, \quad k = 1, \cdots, l \end{aligned} \tag{2.7}$$

respectively. Collectively we have $\mathbf{p} = \frac{\tilde{X}^T\mathbf{w}}{\|w\|_2^2}$ and $\mathbf{c} = \frac{\tilde{Y}^T\mathbf{w}}{\|w\|_2^2}$, which are called the first predictor component and first prediction component respectively [51]. The outer-product $\mathbf{w}\mathbf{p}^T$ provides a *rank-one* approximation to $\tilde{X}$, and similarly $\mathbf{w}\mathbf{c}^T$ is a rank-one approximation to $\tilde{Y}$. Next, $\tilde{X}$ and $\tilde{Y}$ are deflated to remove the effects contributed by the rank-one matrices,

$$\tilde{X} \leftarrow \tilde{X} - \mathbf{w}\mathbf{p}^T, \qquad \tilde{Y} \leftarrow \tilde{Y} - \mathbf{w}\mathbf{c}^T. \tag{2.8}$$

Equation (2.6) and (2.7) and the deflation (2.8) is repeatedly applied until we get N predictor and prediction components, i.e., a predictor component matrix $P \in \mathbb{R}^{p\times N}$ and a prediction component matrix $C \in \mathbb{R}^{l\times N}$. To make predictions for time $\hat{t} + h$, where $1 \le k \le l$, first project flows in the memory window onto the latent component matrix P:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{x}_{\hat{t}} - \bar{\mathbf{x}} - P\mathbf{w}\|_2^2. \tag{2.9}$$

The PLS predicted flow is then computed by

$$[f_{\text{pls},\hat{t}+1}, \cdots, f_{\text{pls},\hat{t}+l}]^T = \bar{\mathbf{y}} + C\hat{\mathbf{w}} \in \mathbb{R}^l, \tag{2.10}$$

The formulation we apply here is essentially the NIPALS-based PLS [52].

2.3 Support Vector Regression

Support vector machine (SVM) is one of the most successful machine learning methods. The variant of SVM for the regression setting is called the Support Vector Regression (SVR) [53, 54]. Traffic flow prediction with SVR has been previously studied in [55, 56]. In this section, the direct prediction strategy with SVR is described.

The regression model for h-step-ahead prediction is given by

$$f_{h,\text{svr}}(\mathbf{x}_t) = \phi(\mathbf{x}_t)^T \mathbf{w}_h + b_h, \tag{2.11}$$

where ϕ is a user-defined function that maps the flows during the memory window into features in higher dimension, and b_h is a bias term for h-step-ahead forecasts. The SVR models are trained separately for each prediction time-step $\{\hat{t} - 1 + h\}_{h=1}^{l}$. The SVR objective function at time $\hat{t} + h$ is

$$(\mathbf{w}_h, b_h) = \underset{\mathbf{w}_h, b_h}{\operatorname{argmin}} \sum_{t=\hat{t}-T}^{\hat{t}} \ell_\epsilon(f_{h,\text{svr}}(\mathbf{x}_{t-h}) - y_t) + \lambda\|\mathbf{w}_h\|^2. \tag{2.12}$$

This is called the structural risk minimization framework [57], where the term $\ell_\epsilon(f_{\text{svr}}(\mathbf{x}_{t-h}) - y_t)$ is the empirical loss we want to minimize from the training data, $\lambda\|\mathbf{w}_h\|^2$ controls the complexity of the model to avoid overfitting. $\lambda \in \mathbb{R}_+$ is a hyperparameter balancing the two terms.

The loss function used in SVR is defined by

$$\ell_\epsilon(f_{h,\mathrm{svr}}(\mathbf{x}) - y) = \begin{cases} 0, & \text{if } |f_{h,\mathrm{svr}}(\mathbf{x}) - y| < \epsilon \\ |f_{h,\mathrm{svr}}(\mathbf{x}) - y| - \epsilon & \text{otherwise.} \end{cases} \tag{2.13}$$

Therefore, using ℓ_ϵ, we allow the learned model to deviate from the true data by a margin ϵ without penalty, where $\epsilon \geq 0$ is supplied by the user. Equation (2.12) can be transformed into a quadratic programming formulation by introducing slack variables [53, 58]. Many state-of-the-art solvers for SVR use sequential minimal optimization (SMO)-type algorithms [59, 60]. After the optimal $\mathbf{w}_h^*$ and b_h^* for time $\hat{t} + h$ is learned, the SVM predicted flow is produced by

$$f_{\mathrm{svr},\hat{t}+h} = \phi(\mathbf{x}_{\hat{t}})^T \mathbf{w}_h^* + b_h^*. \tag{2.14}$$

2.4 Kernel Ridge Regression

In kernel ridge regression (KRR), traffic flows in the memory window are first transformed by a mapping $\mathbf{x}_t \mapsto \phi(\mathbf{x}_t)$, then future flows $\{y_{t+h}\}_{h=1}^{l}$ are modeled by linear combinations of $\phi(\mathbf{x}_t)$. For simplicity, we assume that $\mathbf{x}_t$ and y_t are mean-centered by subtracting the sample average flows from each data point. In the direct prediction scheme, the Ridge Regression model for h-step-ahead prediction is

$$\mathbf{w}_h^* = \underset{\mathbf{w}_h}{\operatorname{argmin}} \sum_{t=\hat{t}-T}^{\hat{t}} \Big(y_t - \phi(\mathbf{x}_{t-h})^T \mathbf{w}_h \Big)^2 + \lambda \|\mathbf{w}_h\|_2^2, \tag{2.15}$$

where $\lambda \in \mathbb{R}_+$ is a hyperparameter. The regularization term $\lambda \|\mathbf{w}_h\|_2^2$ prevents overfitting of the model. By the Representer theorem [9], there is a vector $\boldsymbol{\alpha} \in \mathbb{R}^T$, such that the optimal solution vector $\mathbf{w}_h^*$ for Eq. (2.15) can be expressed as

$$\mathbf{w}_h^* = \sum_{t=\hat{t}-T-1}^{\hat{t}} \alpha_t \phi(\mathbf{x}_{t-h}) = \Phi^T \boldsymbol{\alpha}, \tag{2.16}$$

where $\Phi^T = [\phi(\mathbf{x}_{\hat{t}-h-1}), \phi(\mathbf{x}_{\hat{t}-h-2}), \cdots, \phi(\mathbf{x}_{\hat{t}-h-T-1})]$. Substituting the weight representation in Eq. (2.16) into Eq. (2.15), we have

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \sum_{t=\hat{t}-T}^{\hat{t}} \Big(y_t - \phi(\mathbf{x}_{t-h})^T \Phi^T \boldsymbol{\alpha}\Big)^2 + \lambda \boldsymbol{\alpha} \Phi \Phi^T \boldsymbol{\alpha} \\ =&\min_{\boldsymbol{\alpha}} \ \ \|Y - \Phi\Phi^T \boldsymbol{\alpha}\|_2^2 + \lambda \boldsymbol{\alpha} \Phi \Phi^T \boldsymbol{\alpha} \\ =&\min_{\boldsymbol{\alpha}} \ \ \|Y - K\boldsymbol{\alpha}\|_2^2 + \lambda \boldsymbol{\alpha} K \boldsymbol{\alpha}, \end{aligned} \tag{2.17}$$

where $Y \in \mathbf{R}^T$ is a vector of flows $[y_t]_{t=\hat{t}-T-1}^{\hat{t}}$, and $K := \Phi\Phi^T$. Notice that we can avoid explicitly constructing the transformed explanatory variables $\phi(\mathbf{x}_t)$ in equation (2.17) by specifying a kernel function k such that $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. In addition, Eq. (2.17) is unconstrained and convex, which allows an analytic solution. Setting the gradient of the objective function with respect to $\boldsymbol{\alpha}$ to zero, the optimal solution is given by

$$\boldsymbol{\alpha}^* = \Big(K + \lambda I\Big)^{-1} Y. \tag{2.18}$$

After $\boldsymbol{\alpha}^*$ is obtained, the optimal solution to equation (2.15) can be computed as $\mathbf{w}_h^* = \Phi^T \boldsymbol{\alpha}^*$. The time $\hat{t} - 1 + h$ prediction is

$$f_{\text{krr},\hat{t}+h} = \phi(\mathbf{x}_{\hat{t}})^T \left(\Phi^T \boldsymbol{\alpha}^*\right) = \sum_{t=\hat{t}-T}^{\hat{t}} \alpha_t^* k(\mathbf{x}_{\hat{t}}, \mathbf{x}_{t-h}). \tag{2.19}$$

Again, the mapping ϕ does not come into play directly, the computation can be entirely done via the kernel.

2.5 Gaussian Process Regression

Gaussian process regression (GPR) is a non-parametric Bayesian method closely related to kernel ridge regression. Xie et al. applied Gaussian process regression for inter-state highway flow prediction [61]. GPR differs from kernel ridge regression mainly from the model derivation procedure and the use of Bayesian posterior distribution. In this work, we use the Gaussian process regression model described in [58, 62]. For time $\hat{t}+h$, the flow is modeled by

$$\begin{aligned} y_{\hat{t}+h} &= f_{\text{gpr},\hat{t}+h}(\mathbf{x}_{\hat{t}}) + \epsilon \\ f_{\text{gpr},\hat{t}+h} &\sim \mathcal{GP}\Big(0, k(\mathbf{x}, \mathbf{x}')\Big) \\ \epsilon &\overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2) \end{aligned} \tag{2.20}$$

where $\mathcal{GP}\Big(0, k(\mathbf{x}, \mathbf{x}')\Big)$ denotes a Gaussian process with covariance matrix parametrized by the kernel function $k(\mathbf{x}, \mathbf{x}')$. We assume the residual ϵ is independent of $f_{\text{gpr},\hat{t}+h}$. The zero-mean Gaussian process is used here, since, without loss of generality, the sample mean of flow values can be subtracted from y_t [61]. Under model (2.20), the covariance between traffic flows at t and t' is

$$\operatorname{cov}(y_t, y_{t'}) = \sigma_f k(\mathbf{x}_t, \mathbf{x}_{t'}) + \sigma^2 \delta_{tt'}, \tag{2.21}$$

where $\delta_{tt'}$ is a Kronecker delta function which equals 1 if $t = t'$, and 0 otherwise. The joint distribution between historical flows Y_k and the modeled flow $f_{\text{gpr},\hat{t}+h}(\mathbf{x}_{\hat{t}})$ is

$$\begin{bmatrix} Y_k \\ f_{\text{gpr},\hat{t}+h} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(X, X) + \sigma^2 I & k(X, \mathbf{x}_{\hat{t}}) \\ k(\mathbf{x}_{\hat{t}}, X) & k(\mathbf{x}_{\hat{t}}, \mathbf{x}_{\hat{t}}) \end{bmatrix}\right). \tag{2.22}$$

The posterior predictive distribution [62] of $f_{\text{gpr},\hat{t}+h}$, conditional on $\mathbf{x}_{\hat{t}}$ and historical flows, is

$$p\Big(f_{\text{gpr},\hat{t}+h} | \mathbf{x}_{\hat{t}}, Y_k, X\Big) = \mathcal{N}\Big(\mu_{\text{gpr},\hat{t}+h}, \operatorname{cov}\Big(f_{\text{gpr},\hat{t}+h}\Big)\Big)$$

There are closed-form formulas to compute the posterior mean $\mu_{\text{gpr},\hat{t}+h}$ and posterior covariance $\operatorname{cov}\Big(f_{\text{gpr},\hat{t}+h}\Big)$ [62]. We use the posterior mean $\mu_{\text{gpr},\hat{t}+h}$ as Gaussian process point estimation for flows at time k, i.e., $f_{\text{gpr},\hat{t}+h} := \mu_{\text{gpr},\hat{t}+h}$.

Chapter 3
Variance Regularized Ensemble Learning Model

Several traffic flow forecasting methods are reviewed in Chapter 2. In practice, it will often be necessary to select a single forecast. Therefore, combining the results from individual predictors will be valuable in practice - a consensus outcome potentially improves robustness and prediction accuracy. In this Chapter, a new consensus ensemble learning model is proposed with the following algorithmic contributions:

1. a time-dependent loss function exploiting the temporal data characteristics;
2. a new covariance-based regularizer to balance model diversity and accuracy to learn the parameters;
3. a pruning scheme to safe-guard against prediction anomaly.

Work performed in [63] leads to this chapter.

3.1 Ensemble Model

Traditionally, consensus ensemble methods build a meta-model by convex combination of the base models. Use $\{f_{mt}\}_{m=1}^{M}$ to denote a collection of forecasts from M models at time t.

$$\bar{f} = \sum_{m=1}^{M} \beta_m f_m, \quad \sum_{m=1}^{M} \beta_m = 1, \quad \beta_m \geq 0 \ \ \forall m$$

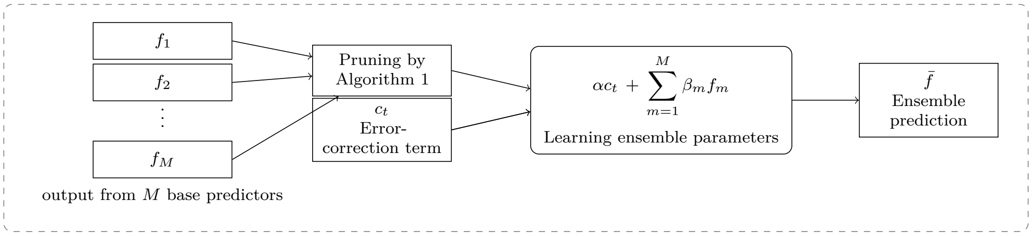

f_1
f_2
$\vdots$
f_M
output from M base predictors
Pruning by Algorithm 1
c_t
Error-correction term
$\alpha c_t + \sum_{m=1}^{M} \beta_m f_m$
Learning ensemble parameters
$\bar{f}$
Ensemble prediction

Figure 3.1. Overview of the consensus ensemble method proposed in this chapter. α and $\{\beta_m\}_{m=1}^{M}$ are ensemble parameters learned from data.

Stack regression is a classical consensus ensemble methods in machine learning for computing the weights $\{\beta_m\}_{m=1}^{M}$. In many machine learning applications, stack regression implicitly assumes that samples in the training set and test set are independently distributed. The training data is shuffled and partitioned. Parts of the training set is used for fitting the sub-models, and the left-out training data is used for computing the ensemble weights.

In the traffic flow time series prediction setting, due to inherent seasonality and other temporal correlations, the shuffling and leave-out operations change the empirical distribution. It is not reasonable to remove some observations $y_{t<\hat{t}}$ and train the base models using samples before and after the removed observations. Therefore, it is necessary to modify stack regression by building the meta-training set in a sequential manner, without data shuffling. We describe a rolling training procedure in more detail in section 3.2.2. In addition, recent data might be more representative than older ones in a temporal prediction task. Finally, the prediction errors may be correlated over time. With these concerns in mind, I now describe the ensemble model for consensus traffic prediction. Figure 3.1 provides an overview of the proposed ensemble learning method.

The model proposed here exploits possible temporally correlated prediction errors. Denote $\bar{f}_t$ the consensus forecast for flows at time t. Define the error-correction term

$$c_t := \frac{\sum_{t'=t-T'}^{t-1} w(t';\theta)\left(y_{t'} - \bar{f}_{t'}\right)}{\sum_{t'=t-T'}^{t-1} w(t';\theta)}, \tag{3.1}$$

where T' is the number of time-steps used to compute c_t from historical predictions and observations. In equation (3.1), the *decay-function* $w(\cdot;\theta)$ is a monotonically non-increasing function in the first argument, which down-weights the difference $y_{t'} - \bar{f}_{t'}$ for large t'. θ is the *decay-rate* hyperparameter which controls how fast $w(\cdot;\theta)$

decreases. There are many decay-functions proposed in the literature, for example, the exponential decay defined by $w_{\exp}(t;\theta) := \exp(-t\theta)$ and the polynomial decay $w_{\text{poly}}(t;\theta) := (1+t)^{-\theta}$ [64]. For both decay-functions, setting $\theta = 1$ is equivalent to placing equal weights for all samples, whereas larger θ discriminate against older ones. The proposed ensemble model is parametrized by

$$\bar{f}_t = \alpha c_t + \sum_{m=1}^{M} \beta_m f_{mt}. \tag{3.2}$$

c_t is a removing average of differences between actual flow values and ensemble predictions from the most recent T' observations, which serves as a error correction term. $\alpha, \{\beta_m\}$ are parameters to be optimized in the model. We proposed the **T**ime **D**ecay **E**rror-**C**orrection Ensemble to learn the model parameters $\alpha, \{\beta_m\}$

$$\begin{aligned}
&\min_{\alpha,\{\beta_m\}} \sum_{t=\hat{t}-T}^{\hat{t}-1} w(t;\theta) \left(y_t - \alpha c_t - \sum_{m=1}^{M} \beta_m f_{mt} \right)^2 \\
&+ \lambda \left(\sum_{m} \beta_m^2 \widehat{\text{var}}(f_m) + \sum_{m} \sum_{m' \neq m} \beta_m \beta_{m'} \widehat{\text{cov}}(f_m, f_m) \right) \\
&\text{subject to } \sum_{m=1}^{M} \beta_m = 1, \quad \beta_m \geq 0 \ \ \forall m, \quad L \leq \alpha \leq U.
\end{aligned} \tag{TDEC}$$

Here T is another user-given hyperparameter to control the number of training samples supplied to the ensemble model. The loss function in TDEC also weights the training samples by a decay term $w(t;\theta)$. The minimization spells more on the loss due to recent data. The term involving λ in the objective is a regularizer. Intuitively, we are seeking for $\{\beta_m\}$ to balance between the weighted $\ell 2$ loss and variance of the ensemble model. $\widehat{\text{cov}}(f_m, f_{m'})$ is the estimated covariance between model m and m'. The choice of $\widehat{\text{cov}}$ is important, but accurate estimation of the covariance is difficult. We described the principle behind the covariance-regularizer and the choice of $\widehat{\text{cov}}$ in the following section. The bounds L and U prevent overfitting by the error-correction term. We discuss how to select hyperparameters θ and λ, and the number of time-steps T' in the error-correction term in section 3.2.

3.1.1 Bias-Variance-Covariance Decomposition

The intuition behind the covariance-regularizer in TDEC is explained in this section. The goal of statistical learning is to select a function $\hat{f}$ to minimize the expected generalization error of a loss function L,

$$\min_{\hat{f}} \mathbb{E}\left(L(y, \hat{f}(\mathbf{x}))\right).$$

The expectation here is averaged over all possible randomness, including the unknown data distribution $(y, \mathbf{x})$ and random training set $\mathcal{T}$. In regression problems, when $L(y, \hat{f}(\mathbf{x})) = (y - \hat{f}(\mathbf{x}))^2$, the generalization error conditional on an input $\mathbf{x}$ could be decomposed by

$$\begin{aligned} &\mathbb{E}_{\epsilon,\mathcal{T}}\left((y_t - \hat{f}(\mathbf{x}))^2\right) \\ =&\sigma_\epsilon^2 + \left(\mathbb{E}_{\epsilon,\mathcal{T}}\hat{f}(\mathbf{x}) - f(\mathbf{x})\right)^2 + \mathbb{E}_{\epsilon,\mathcal{T}}\left(\hat{f}(\mathbf{x}) - \mathbb{E}_{\epsilon,\mathcal{T}}(\hat{f}(\mathbf{x}))\right)^2 \\ =&\sigma_\epsilon^2 + \text{bias}^2(\hat{f}(\mathbf{x})) + \text{var}(\hat{f}(\mathbf{x})), \end{aligned} \tag{3.3}$$

where subscripts under the expectation operator denote the random variables. This is called the bias-variance decomposition [65] and it holds for all data distributions and estimated models $\hat{f}$. For an ensemble model given by $\bar{f} = \sum_m \beta_m f_m$, the variance term reduces to

$$\text{var}(\bar{f}(\mathbf{x})) = \sum_m \beta_m^2 \,\text{var}(f_m) + \sum_m \sum_{m' \neq m} \beta_m \beta_{m'} \,\text{cov}(f_m, f_{m'}). \tag{3.4}$$

Therefore, purpose of the regularizer in TDEC is to strike the right balance between the bias and variance of the ensemble model and achieve a lower expected generalization error. Since the true data generating distribution $(y, \mathbf{x})$ is not known a priori, the minimization of expected $\ell 2$ error is replaced by empirical weighted $\ell 2$ error from the past T time-steps in TDEC. Similarly, $\mathbb{E}_{\mathbf{x}}\left(\text{var}\,\bar{f}(\mathbf{x})\right)$ needs to be estimated. Recall that $\text{cov}(f_m(\mathbf{x}), f_{m'}(\mathbf{x}))$ includes randomness in the training set which produced f_m. Therefore, a straightforward estimation of the covariance between base learners is to retrain the model using *different training data* and compute the sample covariance of predictions for each $\mathbf{x}$. However, this approach is computationally expensive. Note that the error-correction term is not considered as a predictor and the coefficient α does not enter covariance-regularizer and the sum-to-one equality constraint.

3.1.2 Covariance Matrix

Based on the above discussion, choice of the covariance matrix is described below. Let $\hat{t}$ be the current time,

$$\widehat{\mathrm{cov}}(f_m, f_{m'}) = \frac{\sum_{t=\hat{t}-T}^{\hat{t}-1} w(t;\theta)(f_{mt}-\mu_m)(f_{m't}-\mu_{m'})}{\sum_{t=\hat{t}-T}^{\hat{t}-1} w(t;\theta)}, \tag{3.5}$$

where μ_m is an extension of the definition of simple mean, defined by

$$\mu_m = \frac{\sum_{t=\hat{t}-T}^{\hat{t}-1} w(t;\theta) f_{mt}}{\sum_{t=\hat{t}-T}^{\hat{t}-1} w(t;\theta)}. \tag{3.6}$$

This definition of $\widehat{\mathrm{cov}}$ takes time-stamps into account and reduces the influence of older predictions. We define the estimated covariance matrix $\widehat{\Sigma} \in \mathbb{R}^{M\times M}$ as

$$\widehat{\Sigma}_{mm'} = \widehat{\mathrm{cov}}(f_m, f_{m'}). \tag{3.7}$$

$\widehat{\Sigma}$ is symmetric and positive semi-definite. Note that the covariance (and variance) function in TDEC is not restricted to be the one in equation (3.5). In general, better approximations to $\mathbb{E}_{\mathbf{x}}\left[\mathrm{cov}(f_m(\mathbf{x}), f_{m'}(\mathbf{x}))\right]$ may serve as more effective regularizers.

3.1.3 Optimization

In this section, I demonstrate how to solve TDEC via transformation into a convex quadratic programming problem.

Define

$$\mathbf{w} := [\alpha, \beta_1, \cdots, \beta_M]^T \in \mathbb{R}^{M+1},$$

$$\mathbf{y} := [y_{\hat{t}-T}, y_{\hat{t}-T+1}, \cdots, y_{\hat{t}-1}]^T \in \mathbb{R}^T,$$

$$P := \begin{pmatrix} c_{\hat{t}-T} & f_{1\hat{t}-T} & \cdots & f_{M\hat{t}-T} \\ c_{\hat{t}-T+1} & f_{1\hat{t}-T+1} & \cdots & f_{M\hat{t}-T+1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{\hat{t}-1} & f_{1\hat{t}-1} & \cdots & f_{M\hat{t}-1} \end{pmatrix}$$

$$\Lambda := \operatorname{diag}\left(w(\hat{t}-T;\theta), \cdots, w(\hat{t}-1;\theta)\right) \in \mathbb{R}^{T\times T}$$

$$S := \begin{pmatrix} 0 & \mathbf{0}_M^T \\ \mathbf{0}_M & \widehat{\Sigma} \end{pmatrix} \in \mathbb{R}^{(M+1)\times(M+1)},$$

where $\mathbf{0}_M \in \mathbb{R}^M$. Also, let $\mathbf{1}_M$ be a vector of ones, and $\mathbf{I}_{M\times M}$ be the identity matrix of size M. TDEC can then be written as a quadratic programming problem

$$\begin{aligned} \min_{\mathbf{w}} \, & (\mathbf{y} - P\mathbf{w})^T \Lambda (\mathbf{y} - P\mathbf{w}) + \lambda \mathbf{w}^T S \mathbf{w} \\ \begin{bmatrix} 0 & \mathbf{1}_M^T \end{bmatrix} & \mathbf{w} = 1 \\ \begin{bmatrix} \mathbf{0}_M & \mathbf{I}_{M\times M} \end{bmatrix} & \mathbf{w} \geq \mathbf{0}_{(M+1)} \\ \begin{bmatrix} 1 & \mathbf{0}_M^T \end{bmatrix} & \mathbf{w} \geq L \\ \begin{bmatrix} 1 & \mathbf{0}_M^T \end{bmatrix} & \mathbf{w} \leq U. \end{aligned} \tag{TDEC-QP}$$

Therefore, solving TDEC-QP is not a more difficult problem than stack regression.

3.1.4 Unsupervised Pruning

The ensemble model described above uses supervised learning approach to compute the model parameters from data. Each base prediction method has its own degree of robustness against noise and corruptions in the data. Some base learners may occasionally produce unexpected abnormal predictions due to observation noise. A base model may behave well in training, but make an anomalous prediction due to noise or corruptions in the most recently collected data. Any nonzero weight β_m assigned to this base predictions will cause anomalous result in the ensemble prediction as well. This reflects a key assumption behind the consensus approach: the individual members in the ensemble should produce "reasonable" results. Hence, a simple rule-based pruning step is applied before solving TDEC-QP as a safe-guard against prediction outliers.

The pruning (algorithm 1) takes a threshold γ as input. It then discards predicted values that are outside of an interval about the median with size proportional to γ. This pruning scheme is similar to scoring rules used in many sports. For example in synchronized swimming, the highest score and the lowest score are cancelled, and the team's final score is based on the average of the remaining. The experiments show the ensemble predictions are much more robust with the help of

Algorithm 1 Pruning prediction outliers

```
function PRUNING($\gamma, \{f_{mt}\}_{m=1}^{M}$)
    $f_{\max} := \max\{f_{mt}, m = 1, \cdots, M\}$
    $f_{\min} := \min\{f_{mt}, m = 1, \cdots, M\}$
    $f_{\text{median}} := \text{median}\{f_{mt}, m = 1, \cdots, M\}$
    if $f_{\max} > \gamma * f_{\text{median}}$ then
        remove $f_{\max}$ at this timestep
    else if $f_{\min} < (1/\gamma) * f_{\text{median}}$ then
        remove $f_{\min}$ at this timestep
    end if
end function
```

pruning.

3.2 Experiments

To assess the performance of consensus ensemble prediction, the base methods described in Chapter 2 and different ensemble methods are tested on arterial traffic flow. Arterial flow forecasting is a more difficult task than freeway flow forecasting, because of its more variable road conditions.

3.2.1 Data Description

We conducted experiments on traffic flow data collected from arterial sensors in Arcadia, CA in 2015. The raw data is processed into traffic flow time series whose consecutive measurements are separated by a fifteen minute interval, measured in number of cars per hour.

3.2.2 Experimental Procedure

Traffic control centers receive flow measurements periodically. Using historical data and sensor readings from the recent past, traffic operators wish to make multi-step traffic forecasts into the near future. The experiments study a case in which traffic flow measurements are sent to the control center every hour, and forecasts for the following hour is desired. Hence, each prediction consists of four time-steps separated by intervals of fifteen minutes. For instance, using historical data and today's traffic flow up to 7 AM, the flow predictions at 7:15 AM, 7:30 AM, 7:45

AM and 8:00 AM are computed. After that, the true flow at these times are "observed" by the algorithms, and predictions for the next hour are made. Using the notation from the previous sections, the verification and prediction horizon is $l = 4$ in this setting. Notice that under this rolling procedure, flows after the forecasting time-step are never fed into the prediction algorithms.

In the proposed consensus prediction system, there are two levels of training required - one for the base methods and additional training for the ensemble TDEC. For each prediction step t, the base models are trained and predictions for this step are produced. Next, the pruning procedure (Algorithm 1) removes anomalous base forecasts. After that, historical flow observations and the past T base predictions are queried to formulate problem TDEC. Optimal solutions from TDEC-QP are then used to construct the consensus forecast. The base model parameters and ensemble parameters $\alpha, \{\beta_m\}_{m=1}^{M}$ are updated in every time-step.

The following metrics are used for comparing the performance of different base methods and the consensus method. The absolute error of method m at time t is defined as

$$\texttt{AE}(t) = |y_t - f_{mt}|$$

which quantifies for the magnitude of the prediction error. The mean absolute error (`MAE`) is the mean of `AE` in all tested time steps. In addition to the mean, the standard deviation of absolute error is of interest:

$$\texttt{StdAE} = \sqrt{\frac{\sum_t (\texttt{AE}(t) - \texttt{MAE})^2}{\text{number of steps evaluated} - 1}}.$$

`StdAE` provides a view on the robustness of a model. When the `MAE` of two models are close, the one with lower `StdAE` is preferred.

3.2.3 Automatic Hyperparameter Search

We now describe the selection of hyperparameters in TDEC. In general, hyperparameter optimization is expensive and requires repeated model evaluation. We use two hyperparameter search strategies, grid search and random search. Given the traffic flow time series, let $t_{\mathcal{H}}$ be a cut-off time such that measurements before $t_{\mathcal{H}}$ are used for constructing the hyperparameter validation set $\mathcal{V}$, and measurements after $t_{\mathcal{H}}$ are use for testing. Note that with the rolling training process described in section 3.2.2, there is a *cold-start* period, which is the minimum number of time-steps needed to train the base models, plus an additional T steps needed to verify the base predictions and build the consensus model.

In grid search, a set of candidate hyperparameters are specified. Denote $\mathcal{H}(\theta)$ the set of decay rate hyperparameters, $\mathcal{H}(\lambda)$ the set of regularization hyperparameters in TDEC, and $\mathcal{H}(T')$ the number of time-steps used to compute the error-correction term in equation (3.1). In the experiments, let

$$\begin{aligned} \mathcal{H}_{\text{grid}} &= \mathcal{H}(\theta) \times \mathcal{H}(\lambda) \times \mathcal{H}(T'), \\ \mathcal{H}(\theta) &= \{0, 0.05, 0.1, 0.15\}, \\ \mathcal{H}(\lambda) &= \{0, 1, 3, 5\}, \\ \mathcal{H}(T') &= \{8, 40, 80\}. \end{aligned} \tag{3.8}$$

Grid search evaluates all configurations in $\mathcal{H}_{\text{grid}}$. The ensemble model enumerates all possible hyperparameter configurations in the grid search space $\mathcal{H}_{\text{grid}}$ and uses the exponential decay-function $w_{\exp}(t;\theta) = \exp(-t\theta)$. For each hyperparameter choice, the `MAE` on validation set $\mathcal{V}$ is recorded. The one that achieves minimum `MAE` on $\mathcal{V}$ is chosen. Hyperparameter grid search in high dimension is computationally expensive. For N hyperparameters and each with c possible values, grid search procedure results in $O(|\mathcal{V}|c^N)$ model evaluations. Random search is an alternative to grid search, which does not enumerate all possible hyperparameter settings. Rather, each pass over the validation set randomly selects a configuration from the search space. Many researchers have suggested that random search is very competitive in high dimension, due to the *curse of dimensionality* [26]. Recall that the decay-function appears in the error-correction term (equation (3.1)), in the weighted $\ell 2$ loss term of TDEC, and in the estimated covariance matrix (equation (3.5)). The dimension of hyperparameters are expanded if each component is allowed to adapt its own parametrization of decay-function $w(\cdot;\theta)$ and decay-rate θ. In addition, the lower bound L and upper bound U for the error-correction coefficient could also be tuned. We apply random search for hyperparameters from the following search space:

$$\begin{aligned}
\mathcal{H}_{\text{random}} &= \mathcal{H}(w_{\text{loss}}) \times \mathcal{H}(w_{\text{ec}}) \times \mathcal{H}(w_{\text{cov}}) \\
&\quad \times \mathcal{H}(\theta_{\text{loss}}) \times \mathcal{H}(\theta_{\text{ec}}) \times \mathcal{H}(\theta_{\text{cov}}) \\
&\quad \times \mathcal{H}(\lambda) \times \mathcal{H}(T') \times \mathcal{H}(L, U), \\
\mathcal{H}(w_{\text{loss}}), \mathcal{H}(w_{\text{ec}}), \mathcal{H}(w_{\text{cov}}) &= \{\exp(-t\theta), (1+t)^{-\theta}\} \\
\mathcal{H}(\theta_{\text{loss}}), \mathcal{H}(\theta_{\text{ec}}), \mathcal{H}(\theta_{\text{cov}}) &= \{0, 0.05, 0.1, 0.15\}, \\
\mathcal{H}(\lambda) &= \{0, 1, 3, 5\}, \\
\mathcal{H}(T') &= \{8, 40, 80\}, \\
\mathcal{H}(L, U) &= \{L, U \in [0,1], L \leq U\}.
\end{aligned} \tag{3.9}$$

Similar to grid search, the hyperparameter configurations producing the lowest `MAE` from the random search will be used in the testing set. In the experiments, 50 uniform random draws from $\mathcal{H}_{\text{random}}$ are made. Note that the embedding dimension p for $\mathbf{x}_t$, and the training set size $\hat{T}$, T may also be tuned. For comparison in later sections, p is set to 48 (12 hours), $\hat{T}$ is $120 * 24 * 4$ (120 days), $T = 20 * 4$ (80 hours).

The function `fitrsvm` for SVM and `fitrgp` for GPR from MATLAB Machine Learning and Statistics toolbox with their heuristic default values for the hyperparameters [66, 67] are used in the experiments in this chapter. Additionally, PLS and KRR are implemented in MATLAB, and similar default heuristics for hyperparameter configuration are applied. The tables and figures reported here are obtained from base methods with their default hyperparameters.

3.2.4 Overview of Results

One of the motivations for developing a consensus method is that it is unlikely that a single base prediction could consistently outperform others all the time. We verify this hypothesis by comparing the absolute error obtained by the base methods across different time and on different detectors. For each prediction step, a method is marked as the winner if it achieves the lowest absolute error. We compare the percentage of testing days achieving the lowest absolute error by each method at different time. The result is visualized by the area plot in figure 3.2, in which each method is represented by a shaded strip. The width and area of each strip is proportional to the percentage of testing days won by the respective method.

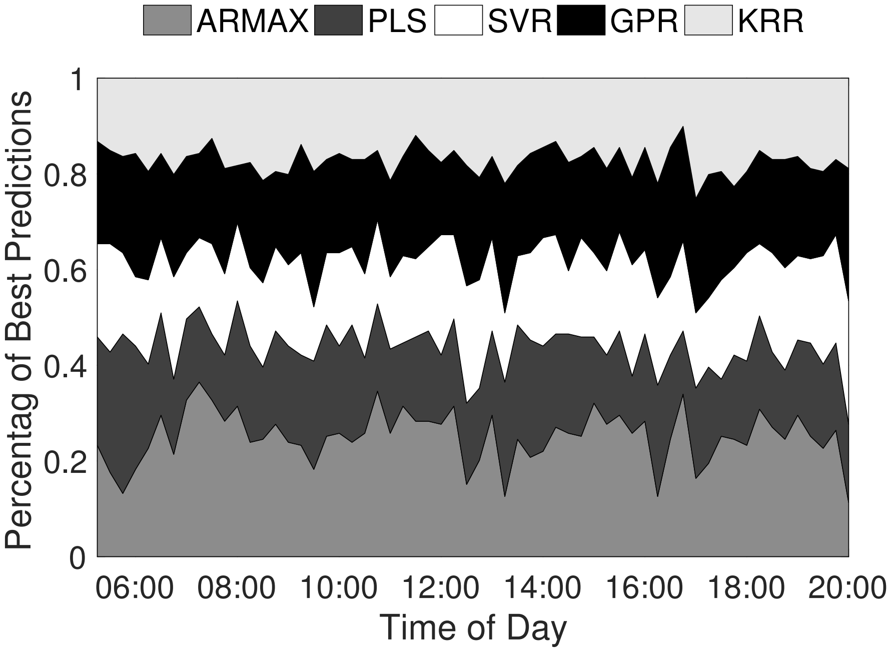


Figure 3.2. Area plot of percentage of best predictions by each method. Area of the strips are proportional to the percentage.

From figure 3.2, there is no single strip whose area dominates the plot. In addition, the strips are in zig-zag shapes. It is not easy to identify a base method that consistently won over a continuous portion of the day. This motivates us to study a model combination approach for flow prediction. Figure 3.3 displays a showcase of the results on three consecutive days randomly selected on a detector. Despite wide ranges of base predictions around the morning rush hours, the ensemble predictions TDEC closely aligned with the actual value of flows.

3.2.5 Baseline and Experimental Goals

Model combination has been studied in many domains, for example, machine learning [65, 68, 69] and econometrics [70–72]. Despite a large body of literature in this area, a common empirical observation in many areas is that the simple average combination which assigns equal weights for the base methods often outperforms complicate combination schemes [71, 73]. This is known as the *"forecast combination puzzle"* in the statistical forecasting literature [71, 73]. Some authors suggested that the weights learned from historical data are unstable and unreliable, as a

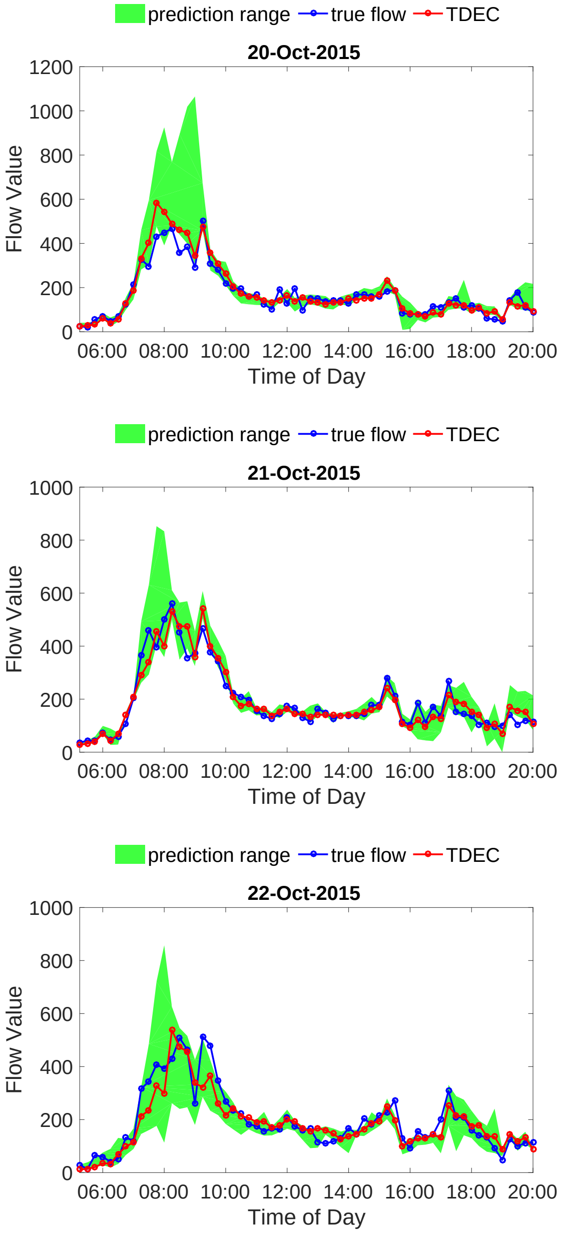


Figure 3.3. Predicted flow values and the true flow in three consecutive days. Despite a wide range of base predictions, the ensemble is closer to the true flow.

consequence of overfitting [72, 73]. Therefore, simple average combination is used as a baseline. A second baseline is the base method achieving the lowest `MAE` and `StdAE` for each detector. We are interested in 1. examining whether the proposed ensemble prediction improves over the simple average combination and best base method.

2. studying which components in the proposed ensemble model contribute to the performance improvement or decline. The ensemble learning method proposed in this chapter is also compared with two multi-model combination methods in traffic flow forecasting literature in section 3.2.8. Note that I do not compare the performance of TDEC with bagging and random forest [74, 75], since this study focus on multi-model combination schemes, whereas random forest uses the same "weak learner" together with data sub-sampling strategy.

3.2.6 Effect of Pruning

In the proposed ensemble system, the pruning scheme may be applied prior to solving the optimization problem TDEC in each time-step. We run the rolling experiments to compare the performance with and without the pruning step. The motivation of unsupervised pruning step is to safeguard the procedure against unrealistic base predictions. Table 3.1 displays the mean absolute error and standard deviation for the following methods: • TDEC-rs, model TDEC where the hyperparameters are automatically selected by the random search procedure outlined in section 3.2.3 • TDEC-gs such that the hyperparameters are set by grid search • SR, Stack regression [76] applied to the rolling experiment setting • AVG, simple average combination of the base methods. In addition, the sub-columns marked by $\gamma = 5$ indicate threshold of the pruning scheme, $\gamma = \infty$ indicate no pruning.

For each detector, the lowest `MAE` is achieved either by TDEC-gs or TDEC-rs. On 12 out of 14 detectors, TDEC (-gs and -rs combined) obtained the smallest `StdAE`. Table 3.2 lists the percentage reductions in `MAE` and `StdAE` with different values of γ, compared to no pruning for each method, averaged from all tested detectors. A positive percentage change denotes an improvement, a negative percentage implies decline in performance. The pruning criterion is designed to be less sensitive with larger γ. Note that there are improvements to the `MAE` and `StdAE` with all three values of γ.

In table 3.1, TDEC-rs and TDEC-gs with $\gamma = 5$ outperforms simple average combination with $\gamma = 5$ in almost all the detectors. However, if the pruning scheme is removed, TDEC-gs with $\gamma = \infty$ produced higher standard deviation than AVG with $\gamma = \infty$ in 5 of 14 cases. Therefore, the pruning scheme is necessary to produce stable results and consistent improvements over simple average combination. In

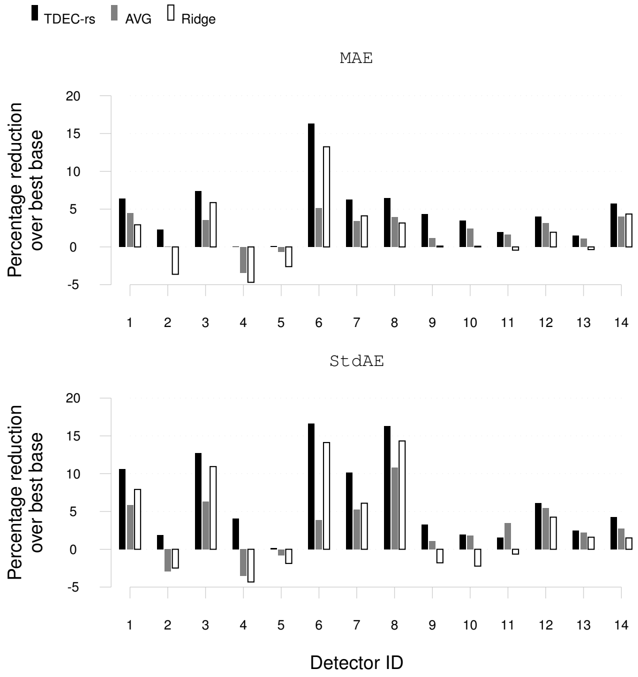


Figure 3.4. Percentage Reduction in `MAE` and `StdAE` of ensemble methods with pruning ($\gamma = 5$), compared to the best base method. Higher values are better for both metricscd.

addition, TDEC (-rs and -gs combined) with the pruning step achieved lower mean and standard deviation of absolute error on all detectors compared to the best base method. This shows that the proposed ensemble model could indeed be used to integrate existent base methods.

3.2.7 Effect of Hyperparameters

The upper panel in Table 3.3 displays the average percentage reductions in `MAE` and `StdAE` obtained by each ensemble method combined with the pruning scheme compared to the best base model on all detectors, the lower panel shows the maximum improvement of `MAE` and `StdAE` among the fourteen tested detectors. A higher value indicates greater improvements. On average, TDEC, with hyperparameters selected either by random search (-rs) or grid search (-gs), outperformed stack regression (SR) and simple average combination (AVG). The improvements by the ensemble methods are less pronounced when $\gamma = 10$. In addition, TDEC with random search resulted in greater improvements over the best base model than TDEC with grid search scheme, likely due to the enlarged hyperparameter search

Table 3.1. Comparison of consensus and base predictors. Lower values are better for both metrics. Best values for each detector are shown in bold font.

Mean Absolute Error `MAE`

ID	TDEC-rs	TDEC-gs		SR		AVG		Ridge		Lasso		Best Base
—	$\gamma = 5$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	
1	**126.6**	126.8	128.2	127.9	129.1	129.2	141.3	131.2	133.3	152	154.3	135.2 KRR
2	**55.8**	56.2	60.3	56.7	60.9	57.1	64.8	59.2	62.4	73.1	75.5	57.1 KRR
3	**42.4**	**42.4**	46.7	42.8	47.5	44.1	46.5	43.1	47.8	47.2	51.5	45.7 KRR
4	**45.2**	**45.2**	47.5	45.5	47.7	46.7	59	47.3	49.7	63.4	66.5	**45.2** KRR
5	**45.1**	**45.1**	47.3	45.2	47.7	45.4	51.8	46.3	48.7	62.7	64.8	45.1 SVR
6	35.2	35.3	**34.6**	35.9	34.8	39.9	39.8	36.4	35.9	60.2	60.2	42 KRR
7	35	35.1	**34.8**	35.2	34.9	36.1	35.9	35.8	35.7	42.3	42.1	37.4 KRR
8	**34.3**	34.5	34.5	**34.3**	**34.3**	35.2	35.4	35.5	35.5	37.1	37.2	36.6 KRR
9	**29.4**	29.5	31.5	30.1	31.9	30.3	35.2	30.7	32.7	33.5	34.9	30.7 KRR
10	**29.2**	**29.2**	31.2	29.3	30.9	29.5	31.1	30.2	32.1	31.1	32.6	30.3 SVR
11	**17.3**	17.4	18.2	17.4	18.1	**17.3**	17.9	17.7	18.3	23.6	24.2	17.6 SVR
12	**10.4**	**10.4**	10.5	**10.4**	10.5	**10.4**	10.5	10.6	10.7	12.7	12.8	10.8 SVR
13	**9.7**	**9.7**	9.8	**9.7**	9.8	9.8	9.8	9.9	10	12.2	12.3	9.9 GPR
14	**8.4**	8.5	**8.4**	8.5	**8.4**	8.6	8.6	8.6	8.5	9.9	9.8	9 SVR

Standard Deviation of Absolute Error `StdAE`

ID	TDEC-rs	TDEC-gs		SR		AVG		Ridge		Lasso		Best Base
—	$\gamma = 5$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	$\gamma = 5$	$\gamma = \infty$	
1	**131.2**	131.6	146.6	133.4	147.5	138.2	562.4	135.2	155	165.3	177.5	146.8 KRR
2	**54.3**	55	153.8	55.6	160.1	56.9	157.3	56.7	133.6	66.1	111.3	55.3 KRR
3	38.7	**38.4**	264.2	39.2	298.9	41.5	156.5	39.5	297.7	41.8	257.3	44.3 KRR
4	**42.1**	43.5	67.4	44	62.9	45.6	228.3	45.9	68.6	56.5	66.7	44 KRR
5	39.2	**39.1**	86.6	39.3	110.8	39.6	355.9	40	112.5	51.6	109.1	39.3 SVR
6	45.6	46.3	45.3	47.4	**45.2**	52.6	49.1	47	46.2	56.6	56.7	54.7 KRR
7	32	32.2	**31.8**	32.6	**31.8**	33.8	32.6	33.5	33	38.6	37.9	35.7 KRR
8	32.6	**32.3**	32.4	32.6	32.6	34.7	35.3	33.4	35.5	33.4	33.4	39 KRR
9	**32.5**	32.8	63.2	33.5	62.2	33.3	155.2	34.2	63.5	36.6	53.5	33.6 KRR
10	**32.9**	33.2	142.1	33.3	134.2	**32.9**	120.6	34.3	131.3	34.4	112.6	33.6 SVR
11	15.9	16.2	52	15.9	45.5	**15.6**	36.1	16.3	38.7	20.2	39.6	16.2 SVR
12	**8.6**	8.7	10.3	**8.6**	10.4	8.7	9	8.8	10.5	10.6	11.7	9.2 SVR
13	**7.8**	7.9	8.7	7.9	8.5	7.9	8.5	7.9	8.5	9.7	10.1	8 GPR
14	**6.9**	7	**6.9**	7.1	**6.9**	7.1	**6.9**	7.1	7	8.1	8	7.3 SVR

space allowed. In the experiment, 50 random draws from the search space $\mathcal{H}_{\text{random}}$ are used. As a result, this requires $50|\mathcal{V}|$ model evaluations of TDEC, where $|\mathcal{V}|$ is the number of time-steps in the hyperparameter validation set $\mathcal{V}$. Note that grid search from $\mathcal{H}_{\text{grid}}$ requires $|\mathcal{V}| \times 3^2 \times 4^2$ model evaluations. Hence, random

Table 3.2. Average percentage reductions in `MAE` and `StdAE` of ensemble methods with pruning, compared to no pruning ($\gamma = \infty$).

	`MAE`				`StdAE`			
γ	TDEC-rs	TDEC-gs	SR	AVG	TDEC-rs	TDEC-gs	SR	AVG
3	3.3	3	2.9	6.1	33.2	33.2	32.7	42.1
5	3.4	3	2.8	5.8	33.3	33.1	32.6	42.1
10	2.9	2.5	2.5	5.1	31.8	31.8	31.5	41.5

Table 3.3. Average and maximum percentage reductions in `MAE` and `StdAE` of ensemble methods with pruning, compared to the best base method. Higher values are better for both metrics. Best values for each detector are shown in bold font.

Average Percentage Reduction								
	`MAE`				`StdAE`			
γ	TDEC-rs	TDEC-gs	SR	AVG	TDEC-rs	TDEC-gs	SR	AVG
3	**4.6**	4.5	4	2.5	**6.5**	6	5.3	3.4
5	**4.7**	4.4	3.9	2.1	**6.6**	5.9	5.1	3
10	**4.3**	4	3.6	1.3	**4.2**	3.3	3.2	0.5
Maximum Percentage Reduction								
	`MAE`				`StdAE`			
γ	TDEC-rs	TDEC-gs	SR	AVG	TDEC-rs	TDEC-gs	SR	AVG
3	15.6	**15.9**	14.5	6.5	16.7	**16.9**	16.2	10.6
5	**16.3**	15.9	14.6	5.1	16.3	**17**	16.3	10.8
10	**16.3**	16.2	15.3	8.7	**15.9**	17	16.4	9.4

search is more efficient and effective than grid search for selecting hyperparameters in the experiments. This observation is consistent with others in the literature on hyperparameter optimization. Some theoretical analysis suggests that because models typically have non-homogeneous sensitivity with respect to different hyperparameters and data distributions, grid search spends too much time exploiting less sensitive hyperparameters [26].

3.2.8 Compare with Other Multi-Model Methods

The proposed ensemble learning model TDEC is compared with other multi-model combination strategies for traffic forecasting to further evaluate its performance. The Ridge Regression Ensemble and Lasso Ensemble were proposed by Li et al. [77]

for freeway traffic estimation. This work shares the same motivation with the studies in this chapter, that "any models existing are imperfect and have their own strengths and weakness". Using the same notations from section 3.1.3, the Ridge Regression Ensemble solves

$$\min_{\mathbf{w}} \|\mathbf{y} - P\mathbf{w}\|_2^2 + \lambda_{\text{ridge}} \|\mathbf{w}\|_2^2 , \tag{3.10}$$

and Lasso Ensemble solves

$$\min_{\mathbf{w}} \|\mathbf{y} - P\mathbf{w}\|_2^2 + \lambda_{\text{lasso}} \|\mathbf{w}\|_1 , \tag{3.11}$$

to obtain the ensemble weights. In Ridge Regression Ensemble, the penalty term $\lambda_{\text{ridge}} \|\mathbf{w}\|_2^2$ forces shrinkage of the solution to avoid overfitting. The l_1-norm penalty in Lasso Ensemble produces a sparse solution, hence fewer base methods will be selected in the ensemble than the one obtained from a least square fitting. Comparing method TDEC-QP and equation (3.10) in Ridge Regression Ensemble, the covariance penalty term in TDEC-QP could be viewed as a generalization to the euclidean norm penalty. Also, it is noteworthy to point out that neither Ridge Regression Ensemble nor Lasso Ensemble requires the weights to be summed-to-one and non-negative. We run the same base methods and compute the ensemble prediction with equation (3.10) and equation (3.11), with the regularization parameter λ_{ridge} and λ_{lasso} selected via grid search from $\{0.1, 1, 3, 5\}$ on the validation data. Both ensemble methods are tested with and without applying the pruning procedure (Algorithm 1). The `MAE` and `StdAE` for each detector under the Ridge Regression Ensemble and Lasso Ensemble are listed in Table 3.1. The Lasso Ensemble predictions, somewhat surprisingly, underperformed the best base method in all detectors; however, this result is consistent with the one reported in [77], in which the authors found Lasso Ensemble improves freeway traffic density (in vehicles per kilometer per lane) estimate but worsen flow rate prediction in many cases. The relative improvements of TDEC-QP, simple averaged combination, and Ridge Regression Ensemble over the best base method for each detector are displayed in figure 3.4. For each bar, positive value denotes improvement and higher is better, vice versa. Our method outperforms Ridge Regression Ensemble and simple averaging in almost all the detectors in both `MAE` and `StdAE`. Moreover, TDEC-QP offers improvement over the best base method even when the other two multi-model methods fail (detector 2, 4, 9, 10, 11). In addition, simple averaging performs better than Ridge Regression Ensemble in more than half of the cases. This observation confirms simple average combination is indeed a very strong baseline [71, 73].

3.2.9 Discussion on Computational Time

There are two major computational stages when running the ensemble system proposed in this chapter. The first stage is to train the base models and generate predictions from each of them. The second stage is to obtain the ensemble parameters via solving a convex quadratic programming problem TDEC-QP. The first stage is common for most multi-model based ensemble methods, for example, Ridge Regression Ensemble [77] discussed in the previous section. Table 3.4 shows the running time in seconds spent by different components of the proposed system in a one-hour-ahead traffic forecast scenario. The numbers reported are the average from ten runs. In the problem setting (section 3.2.2) described earlier, the base models and ensemble model TDEC-QP are refitted every hour. Four predictions spanning one hour are produced after the model fitting. Solving convex quadratic programming based problem is much more efficient than parameter optimization in neural network, which makes the ensemble method computationally more feasible than neural network-based ensemble model for online traffic flow forecast [75, 78]. In the Matlab implementation, the total time needed to finish an ensemble four-step-prediction is in the order of seconds (table 3.4). The running time for solving TDEC-QP is 0.02 seconds on average. Therefore in real operation, the computational time attributed to model fitting and predictions is only a tiny fraction of the one hour time budget. Obtaining a solution for Ridge Regression Ensemble takes only 0.003 seconds on average, since there is a closed-form formula available. A non-smooth convex optimization problem needs to be solved for Lasso Ensemble. On average, the running time with Matlab built-in `lasso` function takes 0.02 seconds.

Table 3.4. Ten-run-averaged Running time (seconds) spent by different components of the system for one-hour-ahead forecast, model fitting and prediction combined. Results measured on a Intel i5 2.40 GHz dual core processor.

ARMAX	PLS	SVM	KRR	GPR	TDEC-QP	Total
0.09	0.32	0.42	0.38	1.19	0.02	2.42

3.3 Chapter Conclusion

This chapter addressed an important practical problem in traffic flow prediction: how to combine the advantage of multiple flow forecasting models to yield a result that is at least as accurate and stable as the best one. An ensemble learning model was proposed to this end. The method described in this chapter was based on three core ideas: 1. learning from mistakes in the recent past, 2. balancing model diversity and accuracy, and 3. applying a pruning scheme to remove extreme forecasts. On the tested arterial traffic sensors, the proposed ensemble model achieved as much as 16.3% and 17% improvements, and on average 4.7% and 6.6% improvements, respectively in mean and standard deviation of absolute error over the best base model. The ensemble learning model TDEC consistently outperformed two recently published ensemble prediction schemes based on Ridge Regression and Lasso, and produced more accurate and robust predictions even in scenarios which the other ensemble methods backfire. In addition, the proposed framework does not have restrictions on the type of sub-models used.

Chapter 4

Online Hyperparameter Optimization for Traffic Flow Prediction

Modern sensors generate large amounts of timestamped measurement data. These data sets are critical in a wide range of applications including traffic flow prediction, transportation management, GPS navigation, and city planning. Machine learning-based prediction algorithms typically adjust their parameters automatically based on the data, but also require users to set additional parameters, known as hyperparameters. For example, in a kernel-based regression model, the (ordinary) parameters are the regression weights, whereas the hyperparameters include the kernel scales and regularization constants.

These hyperparameters have a strong influence on the prediction accuracy. Often, their values are set based on past experience or through time-consuming grid searches. In applications where the characteristics of the data change, such as unusual traffic pattern due to upcoming concert events, these hyperparameters have to be adjusted dynamically in order to maintain prediction quality. In this paper, we use the term hyperparameter learning, hyperparameter optimization, and hyperparameter selection/tuning interchangeably, referring to the process of configuring the model specification before model fitting.

Existing hyperparameter optimization approaches [3–6, 26, 79–86] are designed for offline applications where the data are split into training and validation sets, making them unsuitable for online applications. Therefore, we aim to construct online hyperparameter learning strategies.

This work was motivated by online traffic flow prediction problem. In this context as is many others, a set of learning algorithms are used in the traffic stream

Algorithm 2 A **rolling** hyperparameter tuning and model fitting protocol for deploying machine learning model for time series prediction. For simplicity, we show the 1-step-ahead prediction setting here, but multistep settings are similar.

Input: Model $\mathcal{M}$, hyperparameter tuning interval n (time-steps), model fitting interval m (time-steps).

Output: Predictions $\hat{y}_t$, $t = 0, 1, 2, \ldots, T$.

```
1:  for $t = 0$ to $T$ do
2:      if $((t \bmod n) = 0)$ then
3:          $V_t \leftarrow$ Historical Data for Model Evaluation
4:          $\lambda \leftarrow$ Hyperparameter Tuning$(\mathcal{M}, V_t)$          ▷ costly
5:      end if
6:      if $((t \bmod m) = 0)$ then
7:          $S_t \leftarrow$ Historical Data for Model Training$(t)$
8:          $\theta^*(\lambda) \leftarrow$ Train Model$(\mathcal{M}, S_t, \lambda)$
9:      end if
10:     $\hat{y}_t \leftarrow$ Predict$(\mathcal{M}, \theta^*(\lambda))$
11:     Observe $y_t$
12: end for
```

prediction engine, and the model hyperparameters are often reset periodically. The model re-training is scheduled according to the operation cycle. We summarize this deployment protocol in Algorithm 2 (for 1-step-ahead prediction due to simplicity, multi-steps-ahead are similar). Under this protocol, operators re-select the model hyperparameters every n time-steps, and retrain the model every m time-steps. Note that hyperparameter tuning is much more time-consuming than model fitting. For example, the widely-used grid search strategy selects different hyperparameter configurations based on trial-and-error over the validation data V_t (line 4 in Algorithm 2). In each trial of hyperparameters, the model needs to be re-trained and re-evaluated.

The implication of the high computation cost of most hyperparameter tuning methods is that traffic controllers cannot afford frequent adjustments on hyperparameters. Since the distribution of traffic flow may change gradually, keeping the hyperparameters static may result in sub-optimal performance of the prediction model. However, traffic sensors collect data at a high frequency and the data stream arrives at the control center continuously, the serial correlation of measurements suggest there is a potential for optimizing the hyperparameters in an online manner. Therefore, this work proposes an online method for hyperparameter learning motivated by the need for efficient traffic time series prediction.

Online optimization [27–30] emerged as powerful tools to reduce the computational complexity of model fitting and provide theoretical guarantees. However, when online optimization techniques applied on streaming prediction problems, one often assumes that either the learner $\mathcal{M}$ has no hyperparameters or the hyperparameters are fixed in advance. Despite of the advances in online convex optimization, the rolling prediction scheme outlined in Algorithm 2 is still widely used in practice since almost any learning models can be deployed in this manner. Given the justification that cost of hyperparameter search dominates cost of model learning, speeding up hyperparameter selection will be very useful in practice. Much of the existing work in online optimization are designed for convex objective functions, while the relationship between hyperparameters and prediction accuracy is generally unknown and very unlikely to be convex. Therefore, a key challenge to address is the development of an online optimization strategy for non-convex functions. The major contribution of this work is an online hyperparameter learning algorithm (called OHL) for Kernel Ridge Regression. The algorithm can also be applied to certain class of models where the objective functions satisfies some smoothness assumptions. We analyze our algorithm in non-convex regret minimization framework and prove that it achieves the optimal *local regret* [30] under suitable assumptions.

We make the following contributions in this paper:

- We design a Multiple-Kernel Ridge Regression approach for short-term traffic time time series prediction, which aims to learn the long-term periodicity, short-term deviation and trending of traffic flows simultaneously via the combination of kernels (section 4.2).

- To learn the model hyperparameters effectively and efficiently, we propose an online hyperparameter learning (OHL) algorithm. Our strategy is to adaptively update the model hyperparameters with streaming data (section 4.3.1)

- We first provide an abstraction of the OHL algorithm for a class of models where the objective function satisfies some smoothness requirements, and on which the hyper-gradients can be computed. We then analyze the algorithm under regret minimization framework and show the optimality of the algorithm in terms of *local regret* (section 4.4).

- We tested the multiple-kernel model with the proposed OHL algorithm for traffic flow prediction on I-210 highway, and compared the performance of Multiple-Kernel Ridge Regression under other popular hyperparameter tuning methods. Our method achieves similar and sometimes better prediction accuracy compared to a state-of-art hyperparameter tuning method, while using one-seventh of the computation time (section 4.5).

Work presented in this chapter are based on the paper [8, 87] I published during my doctoral study.

4.1 Common Hyperparameter Tuning Algorithms

4.1.1 Grid Search

Grid Search is the simplest and most widely used hyperparameter tuning strategy. Given a validation set V_t and training set S_t from the historical data, grid search enumerates a user-provided list of hyperparameter settings. For each configuration, the model is fitted on S_t and evaluated on V_t. The configuration yields the best performance on V_t is selected. Suppose there are c possible choices for each hyperparameter, the cost of grid search is $O(c^d \cdot \texttt{cost}(\theta^*(\lambda)))$, where $\texttt{cost}(\theta^*(\lambda))$ is the cost of obtaining $\theta^*(\lambda)$. Hence the computational cost of grid search grows exponentially. When grid search is applied periodically in every n steps (Algorithm 2), the accumulated cost of hyperparameter tuning is

$$O\Big(\frac{T}{n} \cdot c^d \cdot \texttt{cost}(\theta^*(\lambda))\Big), \tag{4.1}$$

where T is the total number of predictions made, d is the dimension of hyperparameters.

4.1.2 Random Search

Random Search has been shown to be effective in high dimensions despite being intuitively simple [26]. Given a budget of R random draws per hyperparameter selection period, instead of enumerating a pre-defined list of configurations, random search trials different hyperparameters. Following the analysis in [26], let the volume of the hyperparameter space be $\mathrm{Vol}(\mathcal{H})$, and let volume containing targeted hyperparameters be $\mathrm{Vol}(\mathcal{T})$, the probability of finding a target out of R random draws is: $1-\left(1-\frac{\mathrm{Vol}(\mathcal{T})}{\mathrm{Vol}(\mathcal{H})}\right)^R$. Suppose the hyperparameters offering good predictions lie in a hyper-rectangle occupying 5% of the search space [26], i.e., $\frac{\mathrm{Vol}(\mathcal{T})}{\mathrm{Vol}(\mathcal{H})}=0.05$, the probability that at least one draw from 50 trials positioned inside the target hyper-rectangle is more than 90%. The accumulated computational cost of random search is

$$O\Big(\frac{T}{n}\cdot R\cdot \texttt{cost}(\theta^*(\lambda))\Big). \tag{4.2}$$

4.1.3 Gradient-based hyperparameter optimization

Gradient-based hyperparameter optimization methods for offline problems were studied in [3–7, 79–81]. In the offline setting, using a training set S and a hold-out validation set V, one may apply gradient-based algorithm with (4.17) by alternatively fitting $\theta^*(\lambda)$ on S and computing the update direction of hyperparameters on V. When the hyper-gradient is available, [4, 81] demonstrate the superior prediction performance of gradient-based tuning. The complexity is $O\Big(I\cdot\big(\texttt{cost}(\nabla_\lambda f)+\texttt{cost}\,(\theta^*(\lambda))\big)\Big)$, where I is the number of iterations taken. In general, $I=\Omega\left(\frac{1}{\epsilon}\right)$ for non-strongly convex functions [88], where ϵ is the convergence threshold. When this approach is deployed online via the rolling protocol (Algorithm 2), the accumulated cost of hyperparameter tuning becomes

$$O\Big(\frac{T}{n}\cdot\frac{1}{\epsilon}\cdot\big(\texttt{cost}(\nabla_\lambda f)+\texttt{cost}\,(\theta^*(\lambda))\big)\Big). \tag{4.3}$$

4.1.4 Bayesian optimization methods

Bayesian optimization [89] is also a popular hyperparameter tuning paradigm. It has been shown that Bayesian optimization can produce state-of-art results for tuning deep learning models. Ironically, Bayesian optimizer itself uses kernels and involves (hyper)-hyperparameters. Therefore, we exclude these approaches in the Experiments section due to the complications in applying the methods.

Note that there is a common factor of $\frac{T}{n}$ in equation (4.1), (4.2), and (4.3) due to the periodic nature in rolling hyperparameter tuning scheduled in every n steps. In section 4.3.1, we propose an online hyperparameter optimization algorithm which removes this factor. The theoretical performance guarantees of the algorithm will be analyzed in section 4.4.

4.2 Multiple-Kernel Ridge Regression

Traffic flow time series is dynamic and hard to predict for a number of reasons. Despite having an approximately AM/PM and weekday/weekend periodic pattern, the short term traffic variation from the mean can be significant. This can be due to traffic accidents, weather, nearby events, and other factors. In addition, traffic measurements can be very noisy due to inherent uncertainties and measurement error.

We use Multiple-Kernel Ridge Regression to simultaneously capture the periodicity pattern and short-term distortion of traffic data. Kernel methods provide expressive tools to model the periodicity and the short-term nonlinear deviation. At each model learning step τ, let $\mathbf{y} \in \mathbb{R}^N$ denote a vector collecting past N data points. For each y_t, let $\mathbf{x}_t := [y_s]_{s=t-p}^{t-1} \in \mathbb{R}^p$ be a vector of p past flow observations that are used as predictor variables for y_t. The training set S_τ consists of pairs of past-present observations $\{(\mathbf{x}_t, y_t)\}_{t=\tau-N}^{\tau}$. In Kernel Ridge Regression, $\phi_{\lambda_K} : \mathbb{R}^p \to \mathbb{R}^q$ is a feature mapping from the raw observations to another feature space, indexed by hyperparameters λ_K. The Kernel Ridge Regression problem [9, 10, 65] finds a weight vector $\mathbf{w} \in \mathbb{R}^q$ that solves

$$\min_{\mathbf{w}} \sum_{t=\tau-N}^{\tau} \Big(y_t - \phi_{\lambda_K}(\mathbf{x}_t)^T \mathbf{w}\Big)^2 + \lambda_R \|\mathbf{w}\|^2, \tag{4.4}$$

where $\lambda_R > 0$ is a regularization hyperparameter to be selected, which controls the variance of estimation. By the Representer Theorem [9, 10], there is $\theta := [\theta_j]_{j=1}^N \in \mathbb{R}^N$, such that the optimal solution $\mathbf{w}^*$ can be written as $\mathbf{w}^* = \sum_{j=1}^N \theta_j \phi_{\lambda_K}(\mathbf{x}_{\tau+1-j})$. Hence, instead of optimizing over $\mathbf{w}$, Eqn. (4.4) can be equivalently solved by

$$\operatorname*{argmin}_{\theta} \Big(\mathbf{y} - K_{\lambda_K}\theta\Big)^T \Big(\mathbf{y} - K_{\lambda_K}\theta\Big) + \lambda_R \theta^T K_{\lambda_K} \theta, \tag{4.5}$$

where $K_{\lambda_K} \in \mathbb{R}^{N\times N}$, $[K_{\lambda_K}]_{ij} = [\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)], i, j = 1, \cdots, N$. Therefore, instead of explicitly constructing the feature mapping $\phi_{\lambda_K}(\cdot)$, one may work directly with suitable kernels $K_{\lambda_K}(\cdot, \cdot) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$. Roughly speaking, kernel methods express the similarity between the N training samples with a positive semi-definite kernel matrix $K_{\lambda_K} \in \mathbb{R}^{N\times N}$, where λ_K is a vector of hyperparameters that determine the kernel.

The hyperparameters of the Kernel Ridge Regression model are denoted by $\lambda := [\lambda_K, \lambda_R]$. Throughout the paper, we use $\theta^*(\lambda)$ to denote the optimal solution of (4.5), highlighting its dependence on λ. The optimal solution $\theta^*(\lambda)$ can be written in closed-form:

$$\theta^*(\lambda) = \Big(K_{\lambda_K} + \lambda_R I\Big)^{-1}\mathbf{y}. \tag{4.6}$$

Different choices of kernels capture different aspects of the data. We model the periodicity of traffic flows as a function of time, and model the short-term deviation from strict periodicity by considering the memory effect from recent traffic. With slight abuse of notation, we also use $K_{\lambda_K}(\cdot,\cdot) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ to denote a pairwise kernel function on two data points. Let y_t, $y_{t'}$ be the traffic volume at time-stamp t, t' respectively. The periodic kernel proposed by Mackay [90] determines the periodicity pattern by the time difference $|t-t'|$ between two observations,

$$K^{\text{prd}}_{\nu,\omega}(t,t') := \exp\Big(-\nu\sin^2\Big(\frac{\pi|t-t'|}{\omega}\Big)\Big). \tag{4.7}$$

In $K^{\text{prd}}_{\nu,\omega}$, $\omega > 0$ is a hyperparameter controlling the period of recurrence, $\nu > 0$ is another hyperparameter deciding the scale[1] of "wiggles" in traffic flow. The short-term nonlinear effect is modelled by a squared exponential kernel using autoregressive feature $\mathbf{x}_t$,

$$K^{\text{se}}_{\nu}(\mathbf{x}_t,\mathbf{x}_{t'}) := \exp\Big(-\nu\,\|\mathbf{x}_{t'} - \mathbf{x}_t\|_2^2\Big). \tag{4.8}$$

The kernel scale hyperparameter ν in K^{se}_{ν} has similar qualitative effects as the one in $K^{\text{prd}}_{\nu,\omega}$, but their values can be different and remain to be chosen. The automatic relevance determination (ARD) kernel is a generalization of K^{se}_{ν} allowing each component of the feature to have a different length scale,

$$K^{\text{ard}}_{\nu}(\mathbf{x}_t,\mathbf{x}_{t'}) := \exp\Big(-\sum_{i=1}^{p}\nu_i\Big(y_{t-i} - y_{t'-i}\Big)\Big). \tag{4.9}$$

---

[1] most literature refer $l = \nu^{-1}$ as the length scale, we use the reciprocal for ease of differentiation later.

The number of hyperparameters in the ARD kernel increases with the number of features, hence it is usually infeasible to optimize them with grid search. A valid kernel function gives rise to a positive semi-definite kernel matrix, where each entry is computed from the kernel function on two data points. Any linear combination between kernels produce a new one. Using this property, given M different kernels, Multiple-Kernel Ridge Regression uses an composite kernel function:

$$K_{\lambda_K} = \beta_1 K^{(1)}_{\lambda_{K_1}} + \beta_2 K^{(2)}_{\lambda_{K_2}} + \cdots + \beta_M K^{(M)}_{\lambda_{K_M}}, \tag{4.10}$$

where $\sum_{i=1}^{M} \beta_i = 1, \beta_i \geq 0$. We consider the coefficients $\{\beta_m\}_{m=1}^{M}$ as hyperparameters, since they determine the final kernel matrix used in equation (4.5). (4.10) can be viewed as an ensemble learning model from different kernels [63, 68, 69]. The composite kernel is a function of time and the autoregressive feature:

$$K_{\lambda_K}\Big((t, \mathbf{x}_t), (t', \mathbf{x}_{t'})\Big) = \beta_1 K^{\mathrm{prd}}_{\nu,\omega}(t, t') + \beta_2 K^{\mathrm{ard}}_{\nu}(\mathbf{x}_t, \mathbf{x}_{t'}) \tag{4.11}$$

After computing $\theta^*(\lambda)$ through equation (4.6), to make a prediction for time t, let the vector of pairwise kernel mappings between $(t, \mathbf{x}_t)$ and $(t', \mathbf{x}_{t'})$ in the training set S_τ be $k_{\lambda_K} := \Big[K_{\lambda_K}\Big((t, \mathbf{x}_t), (t', \mathbf{x}_{t'})\Big)\Big]_{(t', \mathbf{x}_{t'}) \in S_\tau} \in \mathbb{R}^N$. The prediction is given by

$$\hat{y}_t(\theta^*(\lambda)) = k^T_{\lambda_K} \theta^*(\lambda) \tag{4.12}$$

To summarize, λ_K in Multiple-Kernel Ridge Regression includes the hyperparameters for each kernel and the kernel combination coefficients $\{\beta_m\}_{m=1}^{M}$. λ_K and the regularization constant λ_R must be set properly to balance the effects of periodicity in traffic flow, near-term nonlinear distortion due to unusual events, and estimation variance ˘ resulting in a hyperparameter optimization problem.

4.3 Hyperparameter Learning

4.3.1 Hyper-Gradient Computation for Kernels

The dimension of λ can range from tens to hundreds when an automatic relevance kernel is used with high dimensional features. Periodic hyperparameter re-selection brings heavy computational burden for an online operations. This motivates the development of online methods to adaptively learn the hyperparameters. We apply the ℓ_2 loss function to obtain the prediction error for time-step t:

$$\ell\left(y_t, \hat{y}_t(\theta^*(\lambda))\right) = \left(y_t - k_{\lambda_K}^T \theta^*(\lambda)\right)^2 := f_t(\lambda). \tag{4.13}$$

Notice that given y_t and the training set, the prediction error is a *non-convex* function of λ. Even though the loss function is convex, the nested nature of λ in $\theta^*(\lambda)$ and the kernel k_{λ_K} creates non-convexity. Using the chain rule, the partial derivative of $f_t(\lambda)$ with respect to the kernel hyperparameters λ_K is:

$$\begin{aligned} \frac{\partial f_t(\lambda_K)}{\partial \lambda_K} =& -2\Big(y_t - \hat{y}_t(\theta^*(\lambda))\Big)\Big(\frac{\partial k_{\lambda_K}}{\partial \lambda_K}\theta^*(\lambda)\Big) \\ & -2\Big(y_t - \hat{y}_t(\theta^*(\lambda))\Big)\Big(k_{\lambda_K}^T \frac{\partial \theta^*(\lambda)}{\partial \lambda_K}\Big), \end{aligned} \tag{4.14}$$

and the partial derivative with respect to the regularization constant λ_R is:

$$\frac{\partial f_t(\lambda_K)}{\partial \lambda_R} = -2\Big(y_t - \hat{y}_t(\theta^*(\lambda))\Big)\Big(k_{\lambda_K}^T \frac{\partial \theta^*(\lambda)}{\partial \lambda_R}\Big). \tag{4.15}$$

Let $\lambda_R \in [L, U] \subset \mathbb{R}^+$, since K_{λ_K} is positive semi-definite, $A(\lambda) := \Big(K_{\lambda_K} + \lambda_R I\Big)$ is non-singular. Therefore, $A(\lambda)^{-1}$ is differentiable. Let $\lambda(i)$ denote the i-th hyperparameter. Then

$$\frac{\partial A^{-1}(\lambda)}{\partial \lambda(i)} = -A^{-1}(\lambda)\frac{\partial A(\lambda)}{\partial \lambda(i)}A^{-1}(\lambda) \tag{4.16}$$

Consequently,

$$\frac{\partial \theta^*(\lambda)}{\partial \lambda(i)} = \frac{\partial A^{-1}(\lambda)}{\partial \lambda(i)}\mathbf{y} = -A^{-1}(\lambda)\frac{\partial A(\lambda)}{\partial \lambda(i)}\theta^*(\lambda) \tag{4.17}$$

Equation (4.17) along with (4.14) and (4.15) provide the gradient w.r.t hyperparameters (hyper-gradient) given the loss at y_t. In the next subsection, we described a state-of-art gradient-based hyperparameter optimization method [4], and our rational for improving the method.

4.3.2 Our Method: Online Hyperparameter Learning

We propose an online projected hyper-gradient descent algorithm to address the computational burden of applying gradient-based hyperparameter tuning algorithms. The idea is to compute the hyperparameter gradients $\nabla_\lambda f_t(\lambda)$ on-the-fly when a new datum y_t is observed, then average the historical hyper-gradients to make a smoothed update on λ before fitting $\theta^*(\lambda)$ (every m steps). The entire rolling hyperparameter re-selection cycle is removed and replaced by incremental learning procedure. In addition, to speed up the computation of hyperparameter gradients, the terms in equation (4.14) and (4.15) shared with subsequent hyper-gradients are pre-computed and stored after an update.

The projected gradient update to the hyperparameters in every m steps is:

$$\lambda^{\text{new}} = \Pi_C\Big(\lambda^{\text{old}} - \frac{\eta}{m}\sum_{t=\tau-m}^{\tau-1} \nabla_\lambda f_t(\lambda)\Big) \tag{4.18}$$

where $\Pi_C(\cdot)$ is the orthogonal projection operator defined by $\Pi_C(u) = \underset{v\in C}{\operatorname{argmin}} \|u-v\|_2^2$. The hyperparameter space for Multiple-Kernel Ridge Regression is $C = [U, L] \cup \Delta$, such that $\lambda(i) \in [U_i, L_i]$ if $\lambda(i)$ is not $\{\beta_j\}_{j=1}^M$, and $\{\beta_j\}_{j=1}^M \in \Delta := \{\boldsymbol{\beta}^T\mathbf{1} = 1, \boldsymbol{\beta} \geq \mathbf{0}\}$ enforces the simplex constraints on the kernel weights. The Online Hyperparameter Learning (OHL) algorithm for Multiple-Kernel Ridge Regression is presented in Algorithm 3.

4.3.3 Complexity

Lines 4, 7, 14, and 15 in Algorithm 3 are used for adjusting hyperparameters. Line 6 for computing $\theta^*(\lambda)$ is a common step for all rollingly-trained Kernel Ridge methods, and thus not additionally introduced by Algorithm 3. The cost of computing the Jacobian matrix for a hyperparameter in Line 7 via eqn (4.17) is $O(N^2)$ due to matrix-vector multiplications, since the inverse kernel design matrix $A^{-1}(\lambda)$ has been obtained in computing $\theta^*(\lambda)$. Also, this cost only occurs in every m steps. The cost of computing the partial derivative for each hyperparameter in Line 14 via eqn (4.14) and (4.14) reduces to a $O(N)$ inner product operation with a column in the pre-computed Jacobian matrix. Thus, Algorithm 3 efficiently computes the hyperparameter gradients online. Projection onto simplex $\Pi_\Delta(\cdot)$ for the kernel

Algorithm 3 Online Hyperparameter Learning (OHL) and Prediction with Multiple Kernels.

Input: Update window m, learning rate η, convex feasible set C, initial $\lambda_0 \in C$, number of training samples N, total prediction time-steps T.
Output: Predictions $\hat{y}_t, t = 0, \dots, T-1$.

```
1:  for $t = 0 : T-1$ do
2:      if $((t \mod m) = 0)$ then
3:          if $t > 0$ then
4:              $\lambda_t = \Pi_C\Big(\lambda_{t-1} - \eta\ m^{-1} g_t\Big)$        ▷ update $\lambda$
5:          end if
6:          $S_t$ = Historical Data for Model Training($t$)
7:          $\theta^*(\lambda_t) = \Big(K_{\lambda_{t,K}} + \lambda_{t,R} I\Big)^{-1}\mathbf{y}$        ▷ fit model
8:          $J$ = compute Jacobian matrix using Eqn. (4.17)
9:          $g_t = \mathbf{0}$
10:     else
11:         $\lambda_t = \lambda_{t-1}$
12:     end if
13:     $\hat{y}_t = k_{\lambda_{t,K}}(t, \mathbf{x}_t)^T \theta^*(\lambda_t)$        ▷ prediction
14:     Observe $y_t$
15:     $\nabla_\lambda f_t(\lambda_t)$ = compute hyperparameter gradient using Eqn. (4.14), Eqn. (4.15) and pre-computed $J$
16:     $g_{t+1} = g_t + \nabla_\lambda f_t(\lambda_t)$        ▷ hyper-gradient
17: end for
```

coefficients $\{\beta\}_{i=1}^{M}$ can be computed in $O(M \log M)$ time [91], and projection onto box constraints is a linear time operation on the number of hyperparameters. Hence the update step in line 4 can also be done efficiently. Therefore, the complexity of OHL applied on Multiple-Kernel Ridge Regression is

$$O\Big(\frac{T}{m} \cdot \Big(N^2 d + M \log M\Big) + TNd\Big) \tag{4.19}$$

4.4 Theoretical Analysis with Local Regret

We now present the theoretical analysis under online learning framework for non-convex functions. Algorithm 4 is an abstraction of Algorithm 3 for general non-convex function f_t, where the subscript t represents the the time-varying nature of the hyperparameter optimization problem due to dependence on the rolling training

Algorithm 4 Online Projected Gradient Descent with Lazy Updates.

Input: Update window m, learning rate η, convex feasible set C, initial $z_0 \in C$, timesteps T.

Output: Iterates $z_t, t = 0, \ldots, T-1$.

```
for $t = 0 : T - 1$ do
    if $\mod(t, m) = 0$ then
        if $t > 0$ then
            $z_t = \Pi_C\Big(z_{t-1} - \eta\, m^{-1} g_t\Big)$
        end if
        $g_t = \mathbf{0}$
    else
        $z_t = z_{t-1}$
    end if
    Submit $z_t$
    Observe cost function $f_t : C \to \mathbb{R}$
    Compute $\nabla_z f_t(z_t)$
    $g_{t+1} = g_t + \nabla_z f_t(z_t)$
end for
```

set S_t. Note that since Algorithm 3 is a specific implementation of Algorithm 4, the result extends to our hyperparameter learning problem. Online learning models the iterates $\{z_t\}_{t=0}^{T-1}$ and the functions $\{f_t\}_{t=0}^{T-1}$ as a repeated game of T rounds. At each time t, the learner selects an iterate $z_t \in C$, where $C \subset \mathbb{R}^n$ is a compact convex set. After z_t has been chosen, a cost function $f_t : C \to \mathbb{R}$ is revealed to the learner and the learner suffers a loss $f_t(z_t)$. We make the following assumptions on the cost function f_t. These assumptions are satisfied for kernel method hyperparameter learning problem, i.e., when $f_t(\cdot) = \ell(y_t, \hat{y}_t(\theta^*(\cdot)))$. Let $\|\cdot\|$ to denote the Euclidean norm throughout the rest of the paper.

A1. $\sup_{z \in C} |f_t(z)| \le M$ for all t.

A2. f_t is L-Lipschitz: $|f_t(z) - f_t(v)| \le L\,\|z - v\|$.

A3. f_t has Q-Lipschitz gradient:

$$\|\nabla f_t(z) - \nabla f_t(v)\|_2 \le Q\,\|z - v\|\,.$$

The performance of $\{z_t\}_{t=0}^{T-1}$ with respect to $\{f_t\}_{t=0}^{T-1}$ is studied by the measure of regret. For *convex* cost functions, the regret is typically defined by

$$\sum_{t=0}^{T-1} f_t(z_t) - \min_{z \in C} \sum_{t=0}^{T-1} f_t(z),$$

which is the difference between the choices $\{z\}_{t=0}^{T-1}$ and the best fixed decision in hindsight [27, 29]. However, when the cost functions are non-convex, searching for global minimum is NP-hard in general even in the offline case where a static $f : C \to \mathbb{R}$ is known in advance. Furthermore, due to the convex constraint $z \in C$, a large number of gradient evaluations are required to discover a stationary point [30]. Thus, for offline problems, a relaxed criterion is to minimize the (C, η)-*projected gradient* [30, 92]:

$$P(z, \nabla f(z), \eta) := \frac{1}{\eta}\Big(z - \Pi_C\big(z - \eta \nabla f(z)\big)\Big). \tag{4.20}$$

It is easy to see that $P(z, \nabla f(z), \eta)$ mimics the role of gradient in a projected gradient update:

$$z_{t+1} := \Pi_C\big(z_t - \eta \nabla f(z_t)\big) = z_t - \eta P(z_t, \nabla f(z_t), \eta). \tag{4.21}$$

Since $C \subset \mathbb{R}^n$ is compact and convex, and assuming f satisfies **A1-3**, then there exists a point $z^* \in C$ such that $P(z^*, \nabla f(z^*), \eta) = 0$ [30]. Therefore, as a natural extension from the offline criterion of vanishing projected gradients, the *local regret* for non-convex online learning is defined as follows.

Definition 1. *The local regret [30] for loss functions $\{f_t\}_{t=0}^{T-1}$ and sequence of iterates $\{z_t\}_{t=0}^{T-1}$ is*

$$\mathcal{R}_T = \sum_{t=0}^{T-1} \left\| P(z_t, \nabla f_t(z_t), \eta) \right\|^2. \tag{4.22}$$

Definition 1 was first used by Hazan et al. [30]. The following theorem shows the optimal local regret is lower bounded by $\Omega(T)$.

Theorem 1. *Define $C = [-1, 1]$. For any $T \geq 1$ and $\eta \leq 1$, there exists a distribution $\mathcal{D}$ of loss functions $\{f_t\}_{t=0}^{T-1}$ satisfying assumption **A1-3**, such that for* *any online algorthms, the local regret satisfies*

$$\mathbb{E}_{\mathcal{D}}\Big(\mathcal{R}_T\Big) \geq \Omega(T). \tag{4.23}$$

Proof. See Theorem 2.7 in Hazan et al. [30]. □

A time-smoothed follow-the-leader (FTL) algorithm was proposed in [30], achieving the optimal local regret bound $O(T)$ for non-convex functions. This algorithm computes the gradients $\{\nabla f_{t-i}(z_t)\}_{i=1}^m$ and updates the iterate z_t in every step. For the hyperparameter learning problem considered in this paper, when z represents hyperparameter λ, a change from λ_t to λ_{t+1} will require model refitting to update $\theta^*(\cdot)$ from $\theta^*(\lambda_t)$ to $\theta^*(\lambda_{t+1})$ in every step. Besides, the historical gradients are not reused in the time-smoothed FTL algorithm [30], since $\nabla f_{t-i}(\cdot)$ is re-evaluated at latest z_t in every step. Hence, the method in [30] becomes impractical for online hyperparameter learning given a computational budget. In contrast, Algorithm 4 accumulates the gradients and produces an update every m-steps, which dramatically reduces the amount of gradient computation and model-refitting on $\theta^*(\lambda)$. As a price paid for the speed-up, we need the following additional assumption characterizing the variation of cost functions to achieve optimal regret bounds. Let $[r]$ denote $[0, r] \cap \mathbb{Z}$.

A4. Assume there is a constant $w \in \mathbb{Z}$ **independent of** T, for all $m \in [w]\backslash\{0\}$, there is a constant $V_m \in \mathbb{R}^+$, such that for any t, the variation of gradients from the average within m steps is bounded:

$$\begin{aligned} &\sup_{z\in C} \sum_{i=0}^{m-1} \left\|\nabla f_{t+i}(z) - \nabla F_{t,m}(z)\right\|^2 \leq V_m, \\ &\text{where} \quad F_{t,m}(z) := \frac{1}{m}\sum_{i=0}^{m} f_{t+i}(z). \end{aligned} \tag{4.24}$$

In the convex setting, variations defined similar to (4.24) have also been studied [93–95]. However, the variation used in [93] defines $m = T$, whereas in (4.24) m is a constant independent of T. Note that the quadratic variation in (4.24) also implies

$$\sup_{z\in C} \sum_{i=0}^{m-1} \|\nabla f_{t+i}(z) - \nabla F_{t,m}(z)\| \leq \sqrt{mV_m}. \tag{4.25}$$

Corollary 1. *There exists a distribution $\mathcal{D}$ of loss functions satisfying assumption **A1-3**, and **A4**, such that for any iterates $\{z_t\}_{t=1}^{T}$, the lower bound $\mathbb{E}_{\mathcal{D}}\Big(\mathcal{R}_T\Big) \geq \Omega(T)$ still applies.*

Proof. See Theorem 2.7 in Hazan et al. [30], construction of $\mathcal{D}$ in Theorem 1 also satisfies assumption **A4**. ☐

We need a few properties of projected gradients before establishing the regret bound for algorithm 4.

Lemma 1. *For any $z \in C$, $\nabla f(z)$, and $\nabla g(z)$,*

$$\left\| P(z, \nabla f(z), \eta) - P(z, \nabla g(z), \eta) \right\|_2 \leq \left\| \nabla f(z) - \nabla g(z) \right\|_2 \tag{4.26}$$

Proof. An application of Lemma 2 in Ghadimi et al. [92]. ☐

Lemma 2. *For any $z \in C$ and $\nabla f(z)$,*

$$\langle \nabla f(z), P(z, \nabla f(z), \eta) \rangle \geq \left\| P(z, \nabla f(z), \eta) \right\|_2^2 . \tag{4.27}$$

Proof. See Lemma 1 in [92] and Lemma 3.2 in [30]. ☐

We now bound the local regret $\mathcal{R}_T$ of the whole sequence by the projected gradients of its subsequence. Recall that Algorithm 4 updates the iterate z_t every m steps. Without loss of generality, assume $s = T/m \in \mathbb{Z}$. Let $\tau_j, j = 1, \cdots, s$ denote the steps at which an increment will occur, i.e., $0, \cdots, T-1$ can be represented as

$$\tau_0, \tau_0 + 1, \cdots, \tau_0 + m - 1, \tau_1, \cdots, \tau_s, \tau_s + 1, \cdots, \tau_s + m - 1.$$

Moreover, $z_{\tau_j} = z_{\tau_j + i}$ for any $j \in [s]$ and $i \in [m-1]$.

Proposition 1. *Let $\{\tau_j\}_{j=0}^{s}$ denote the steps at which an increment to the iterates will occur in Algorithm 4. Suppose m in Algorithm 4 is chosen such that $m \leq w$ in* *assumption **A4**, the local regret satisfies*

$$\begin{aligned}\mathcal{R}_T \leq & \sum_{j=0}^{s} \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|^2 + \\ & + 2\sqrt{mV_m} \sum_{j=0}^{s} \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\| + (s+1)V_m.\end{aligned} \tag{4.28}$$

Proof. Recall $F_{t,m}(z) = m^{-1}\sum_{i=0}^{m-1} f_{t+i}(z)$ from equation (4.24),

$$\begin{aligned}\mathcal{R}_T &= \sum_{j=0}^{s}\sum_{i=0}^{m-1} \left\| P(z_{\tau_j+i}, \nabla f_{\tau_j+i}(z_{\tau_j+i}), \eta) \right\|^2 \\ &= \sum_{j=0}^{s}\sum_{i=0}^{m-1} \left(\left\| P(z_{\tau_j+i}, \nabla f_{\tau_j+i}(z_{\tau_j+i}), \eta) - P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) + P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\| \right)^2 \\ &\leq \sum_{j=0}^{s}\sum_{i=0}^{m-1} \left(\left\| P(z_{\tau_j}, \nabla f_{\tau_j+i}(z_{\tau_j}), \eta) - P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\| + \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\| \right)^2\end{aligned} \tag{4.29}$$

where the last line follows from $z_{\tau_j+i} = z_{\tau_j}$ for $i \in [m-1]$ and triangle inequality. From Lemma 1,

$$\begin{aligned}\mathcal{R}_T &\leq \sum_{j=0}^{s}\sum_{i=0}^{m-1} \left(\left\| \nabla f_{\tau_j+i}(z_{\tau_j}) - \nabla F_{\tau_j,m}(z_{\tau_j}) \right\| + \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\| \right)^2 \\ &\leq \sum_{j=0}^{s}\sum_{i=0}^{m-1} \left(\left\| \nabla f_{\tau_j+i}(z_{\tau_j}) - \nabla F_{\tau_j,m}(z_{\tau_j}) \right\|^2 + \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|^2 \right. \\ &\qquad \left. + 2\left\| \nabla f_{\tau_j+i}(z_{\tau_j}) - \nabla F_{\tau_j,m}(z_{\tau_j}) \right\| \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\| \right)\end{aligned} \tag{4.30}$$

Applying assumption **A4** and its implication (4.25) yields the claim. □

Proposition 1 has a very intuitive meaning: when the variation of gradients of the loss functions are bounded, the local regret is bounded by the projected gradients of loss at the updating steps $\tau_j, j \in [s]$. We now state the main theorem on the asymptotic growth of local regret in Algorithm 4.

Theorem 2. *Let w be the constant in assumption **A4**, choosing the update period $m \leq w$, learning rate $\eta \in (0, \frac{2}{Q})$ in Algorithm 4, the local regret satisfies*

$$\mathcal{R}_T \leq O(T). \tag{4.31}$$

Proof. From assumption **A3**, $\nabla f_t(z)$ is Q-Lipschitz, therefore $\nabla F_{t,m}(z)$ is also Q-Lipschitz for all t, m. For any τ_j,

$$\begin{aligned} F_{\tau_j,m}(z_{\tau_{j+1}}) \leq F_{\tau_j,m}(z_{\tau_j}) - \eta \left\langle \nabla F_{\tau_j,m}(z_{\tau_j}), P\Big(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta\Big) \right\rangle \\ + \frac{Q\eta^2}{2} \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|_2^2 \end{aligned} \tag{4.32}$$

Applying Lemma 2,

$$F_{\tau_j,m}(z_{\tau_{j+1}}) \leq F_{\tau_j,m}(z_{\tau_j}) - \eta \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|_2^2 + \frac{Q\eta^2}{2} \left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|_2^2 \tag{4.33}$$

Rearrange the terms,

$$\left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|_2^2 \leq \frac{\Big[F_{\tau_j,m}(z_{\tau_j}) - F_{\tau_j,m}(z_{\tau_{j+1}})\Big]}{(\eta - \frac{Q\eta^2}{2})} \tag{4.34}$$

From assumption **A2**, $F_{\tau_j,m}(z)$ is L-Lipschitz. Since $C \subset \mathbb{R}^n$ is compact, let $D := \max_{u,v \in C} \|u - v\|_2$ denote the diameter of C. We have

$$\left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|_2^2 \leq LD(\eta - \frac{Q\eta^2}{2})^{-1} \tag{4.35}$$

$$\left\| P(z_{\tau_j}, \nabla F_{\tau_j,m}(z_{\tau_j}), \eta) \right\|_2 \leq L^{1/2} D^{1/2} (\eta - \frac{Q\eta^2}{2})^{-1/2} \tag{4.36}$$

Summing up all $j = 0$ to s, plugging the bounds (4.35) and (4.36) into Proposition 1, and from assumption **A4**,

$$\begin{aligned} \mathcal{R}_T &\leq (T/m) LD(\eta - \frac{Q\eta^2}{2})^{-1} + (2T\sqrt{V_m}/\sqrt{m}) L^{1/2} D^{1/2} (\eta - \frac{Q\eta^2}{2})^{-1/2} + TV_m/m \\ &\leq O(T). \end{aligned} \tag{4.37}$$

Hence the regret bound is proved as claimed. □

4.5 Experiments

We conduct experiments to evaluate the proposed online hyperparameter learning method on synthetic and real data. On the real data, we perform 15-minutes-ahead traffic flow prediction on 13 randomly-selected sensors of different types, distributed along the I-210 [96] highway in California. The primary goals of the experiments are to

1. Test whether our proposed Online Hyperparameter Learning (OHL) method can *adaptively learn* the hyperparameters, given a misspecified starting point.
2. Examine the *computational efficiency* of OHL against other model tuning methods.
3. Compare the traffic flow *prediction accuracy* of OHL with other hyperparameter tuning strategies with multiple kernel models.

4.5.1 Synthetic data

A stochastic process with periodicity and linear trends is generated using the following scheme:

$$y(t) = 1 + c_1 \mathrm{AR}(20) + c_2 \sin(t/\omega) + \epsilon \tag{4.38}$$

Here, $c_1 = c_2 = 0.5$, $\omega = 5$, $\epsilon \sim \mathcal{N}(0, 0.3)$. The autoregressive coefficients are $\boldsymbol{\alpha}_i / 2 \|\boldsymbol{\alpha}\|$ with $\boldsymbol{\alpha}_i = i$. One-step-ahead predictions are produced by kernel methods under fixed hyperparameters and under the proposed OHL algorithm. The predictions are compared against the ground-truth. In both cases, the initial hyperparameters are the same. $m = 10$ and $\eta = 0.001$ in Algorithm 3.

Two kernels are used in the tests, a squared exponential kernel and the combination of a linear kernel and a periodic kernel. The initial kernel scale is $\nu = 0.1$ for the square exponential kernel. When this choice of kernel scale is fixed and used for the rest of learning, the prediction produces a zigzagging line, indicating the kernel scale is misspecified (green line in Figure 4.1). The predictions produced by squared exponential With OHL overlap with the fixed hyperparameter case initially, given the same starting hyperparameters, but aligns much closer to ground truth after time step 400. OHL is tested for multi-kernel learning using periodicity and

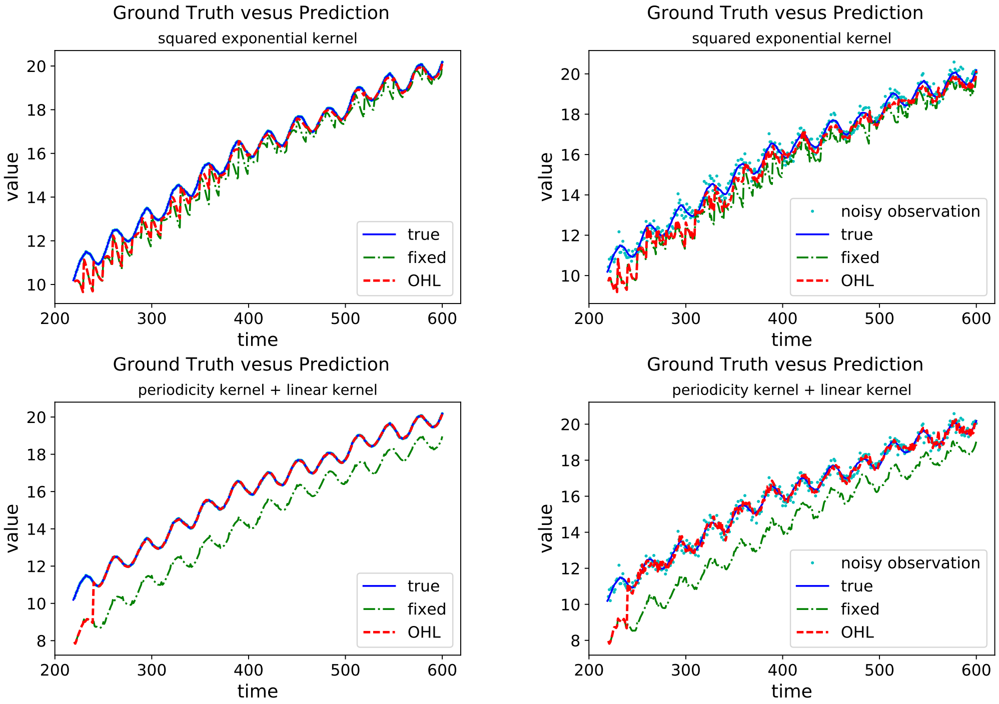


Figure 4.1. Comparing OHL and FIXED on synthetic data. Left: squared exponential kernel, Right: Combination of periodicity and linear kernel. OHL (red line) has a self-correcting behavior towards ground truth, even when the initial hyperparameters are mis-specified.

linear kernel. The initial hyperparameters are $\beta_{\text{prd}} = 1$, period $\omega = 5$ and scale $\nu = 10$ for the periodicity kernel, and $\beta_{\text{lin}} = 0$ for the linear kernel. Hence, it is expected in the fixed hyperparameter case, periodicity can be reproduced but linear trend will be hard to capture. As seen in the bottom panels of Figure 4.1, there is an equidistant gap between the ground truth and the predictions under fixed hyperparameters. With OHL, the learner soon discovers the autoregressive drift term and the predictions are perfectly aligned with ground truth (bottom panels of Figure 4.1). The synthetic experiments demonstrate that OHL can perform adaptive learning, which allows the users to initiate the system without the costly tuning process.

4.5.2 I-210 Traffic Data

4.5.2.1 Data and Setup

The I-210 highway is a vital route in the San Gabriel Valley region of the Los Angeles metropolitan area [96]. We use sensor data from thirteen randomly selected locations covering both mainline detectors and ramp detectors. Measurements from January 1 to May 16 of 2017 are used. The raw data were binned using a 15 minute time window. Hence, there are 96 observations per day. 15-minutes-ahead predictions are tested.

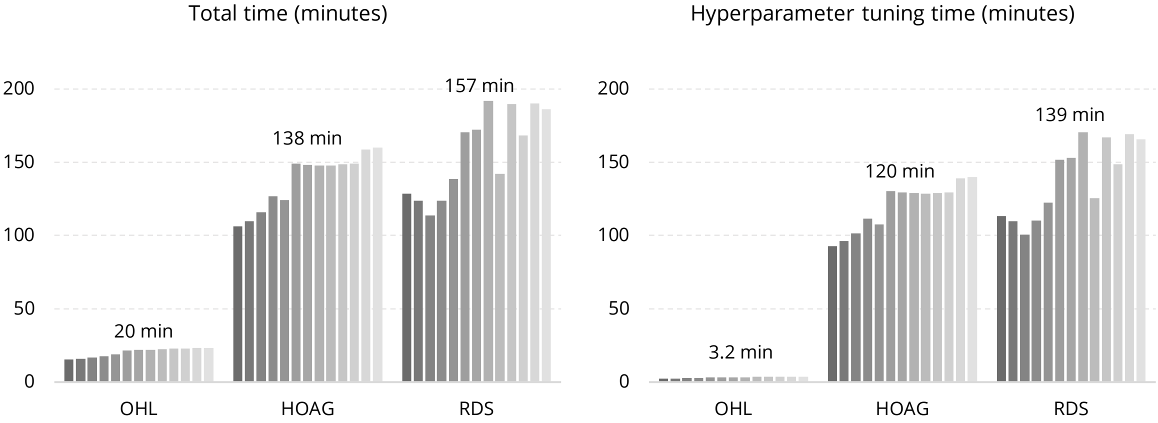


Figure 4.2. Comparing total time and hyperparameter tuning time for OHL, HOAG, and RDS. Each bar indicates the time taken for a detector, and the labels above bars indicate average time for 13 detectors. Detectors are ordered by total time using OHL.

We compare the computational efficiency and prediction accuracy of hyperparameter configurations tuned by OHL versus other algorithms on multiple kernel regression. A linear combination of the ARD kernel and periodic kernel is used. Flow data from past 20 time-steps are used as autoregressive features. The feasibility sets for the hyperparameters are $[1.5 * 10^{-2}, 1.5 * 10^{-6}]$ for kernel scales, $[96/2, 96 * 7]$ for periodicity in the periodic kernel, $[0.03, 3]$ for regularization constant. The following hyperparameter tuning algorithms are tested:

- The Online Hyperparameter Learning (OHL) proposed in this paper.
- Hyperparameter Optimization with Approximate Gradient (HOAG) [4], a state-of-art gradient-based hyperparameter optimization method.
- Random Search (RDS) [26].

- Grid Search and fixed hyperparameters (FIXED), a baseline.

Our experimental procedure aims to mimic the application scenario of a traffic prediction engine. Note that both HOAG and RDS are offline tuning strategies. Therefore, to apply HOAG and RDS in a running environment, the traffic operators need to re-run these tuning strategies periodically, as described in the rolling protocol (Algorithm 2) in the introduction. Hence, we set the hyperparameter optimization interval for HOAG and RDS as $n = 96 * 7$, which corresponds to weekly model tuning. At each hyperparameter tuning step, the validation set V_t consists of observations in the past one month, and these are given to the tuning algorithm. The tuning algorithms HOAG and RDS then perform backtesting on the validation data V_t. HOAG uses hyperparameter gradient information to guide the search for optimal hyperparameters on V_t [4]. RDS experiments with previously selected configuration and 50 additional random configurations on the validation dataset V_t; the one offering the best backtesting accuracy is used for the next period.

RDS is simple to implement and often produces good hyperparameter tuning results. It is thus widely used in practice [26]. After the model hyperparameters are selected for the weekly interval, the model is trained with a training set of $|S_t| = 2880$ time-steps.

OHL is an online method, taking the streaming data and adaptively updating the hyperparameters. Therefore, it does not require backtesting with the validation data V_t. The hyperparameter learning rate is set to $\eta = 10^{-4}$ in OHL, and m is set to 96. Therefore, OHL computes the hyperparameter gradient online and makes an adaptive update every 96 steps. We keep the training frequency and amount of training data the same for all methods.

4.5.2.2 Computational Efficiency Comparisons

Figure 4.2 gives the total time and the hyperparameter tuning time for the methods OHL, HOAG, and RDS, and for the 13 detectors. Overall, OHL is nearly **7**× faster than HOAG and RDS. The speedup is mainly due to the faster hyperparameter tuning in OHL compared to HOAG and RDS. The average hyperparameter tuning times for OHL and HOAG are 3.2 and 120 minutes, respectively, indicating a tuning speedup of 37.5×. The time spent on hyperparameter selection in OHL is a small percentage of the total running time, and hence additional computational overheads are not introduced compared to predictions under FIXED hyperparameters. The dramatic speedup in OHL over the slow execution of HOAG and RDS is expected: although HOAG and RDS are both good hyperparameter tuning algorithms for I.I.D. setting, the rolling procedure for time-series prediction applies the tuning algorithms periodically according to the operation schedule (weekly in our experiments). Each run of the hyperparameter optimization algorithms requires backtesting on the validation set, which also in turn involves multiple parameter fitting steps corresponding to different hyperparameters. Even though HOAG uses gradient-based optimization, the algorithm searches for the optimal solution of hyperparameters on V_t during each tuning stage. In comparison, OHL extracts the hyper-gradient knowledge adaptively from the streaming data, and making a single projected-gradient update to the hyperparameters before re-training the model. Further, the overall times (average across 13 detectors) for OHL and HOAG are 20 and 138 minutes, indicating a 6.9× speedup.

4.5.2.3 Prediction Accuracy

We use Root Mean Squared Error (RMSE) to measure the traffic flow prediction accuracy with different hyperparameter optimization algorithms. In order to examine the variation of RMSE over time, we also report the RMSE as a function of time-step t:

$$\mathrm{RMSE}(t) := \sqrt{\frac{1}{t}\sum_{\tau=0}^{t}(y_\tau - \hat{y}_\tau)^2},$$

Thus, RMSE(t) summarizes the average prediction error from the start to time-step t. Due to space limit, we only show the evolution of RMSE(t) over the testing period for six detectors in Figure 4.3. The X axis in Figure 4.3 is the prediction time-step and the Y axis corresponds to the RMSE up to that time-step. OHL, HOAG, and RDS have lower RMSE compared to the result with FIXED hyperparameters. This is an indication of suboptimal hyperparameters selected and then fixed by grid search. The prediction accuracy of OHL is similar to HOAG in most cases, and sometimes better. For example, on detector 1 and detector 2 (first two charts in the top row of Figure 4.3), RDS has the lowest RMSE in the beginning phrase of testing, but OHL gradually improves and outperforms

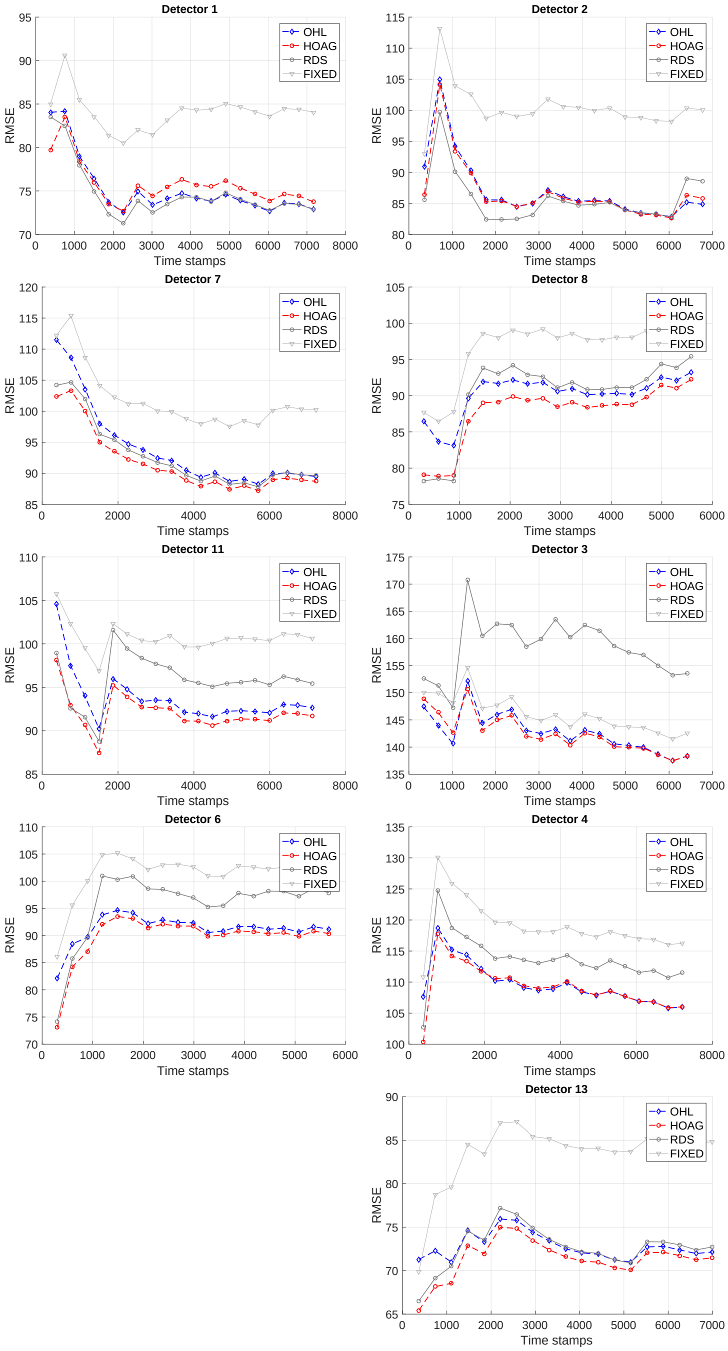


Figure 4.3. Prediction RMSE comparison for traffic flow data, OHL vs other hyperparameter tuning algorithms.

others over time. Meanwhile, OHL is also computationally the cheapest among the three hyperparameter tuning strategies. The similar prediction accuracy between HOAG and OHL in most cases are expected (Table 4.1), since both use gradient-based optimization on hyperparameters. However, HOAG periodically applies the gradient-based iterations until convergence on the validation dataset, which makes the overall computation costly. In contrast, OHL achieves the same result with adaptive updates. On some detectors, RMSE of the model tuned by OHL algorithm is lower than HOAG - an offline gradient-based counter part. There are two reasons that can explain why an online HO algorithm performs better than an offline one. The optimal hyperparameters on validation set underperform in future data, suggesting that either there is *overfitting by hyperparameters* or the data distribution is not stationary. On the contrary, OHL enables timely hyperparameter updates adapted to the latest observations.

Table 4.1. RMSE percentage improvement relative to FIXED hyperparameters. Larger values are better. OHL achieves *similar accuracy* to HOAG, and nearly 7× faster (see Figure 4.2).

	RMSE Improvements					
	4000 time-steps			final		
ID	OHL	HOAG	RDS	OHL	HOAG	RDS
1	12%	11%	12%	13%	12%	13%
2	15%	15%	15%	15%	14%	11%
3	2%	2%	-11%	3%	3%	-8%
4	8%	7%	4%	9%	9%	4%
5	8%	6%	3%	9%	8%	1%
6	11%	11%	5%	11%	11%	4%
7	9%	10%	9%	11%	12%	11%
8	8%	9%	7%	8%	9%	6%
9	8%	9%	2%	9%	10%	3%
10	7%	9%	9%	8%	10%	9%
11	8%	8%	4%	8%	9%	5%
12	7%	7%	6%	7%	8%	5%
13	14%	15%	14%	15%	16%	14%

4.6 Conclusions

Motivated by the need for hyperparameter optimization in traffic time series prediction, we proposed the OHL algorithm and applied it on Multiple-Kernel Ridge Regression. The proposed OHL algorithm achieves optimal local regret. In the traffic flow prediction experiments, OHL is nearly 7× faster than other rolling hyperparameter tuning methods, while achieving similar prediction accuracy. In addition, we observed a consistent improvement in accuracy compared to predictions produced with static hyperparameters.

There are possible extensions to this work: efficient online hyper-gradient approximation methods for a general class of models can expand the application scope of OHL. One direction of improvement is combining our OHL algorithm with the reverse-mode and forward-mode computation of hyper-gradients [6].

Chapter 5
Stochastic Gradient Optimization for Mixed Logit Model

The probability distribution of multinomial response conditioned on the features can be modeled by logistic (logit) models. The two most common applications of logit models in transportation are traffic accident analysis [97–101] and transportation demand modeling [16, 102–104]. In traffic safety analysis, multinomial logit is used for modeling the probability of different injury outcomes in an accident, given a vector of factors or features describing the accident characteristics. For example, researchers have applied multinomial logit model to understand the significance of gender and age effects of the victim, effect of speed and road condition on the accident. A fixed vector of parameters are estimated in multinomial logistic regression. The estimation procedure is often carried out via (penalized) maximum likelihood [105] approach. The resulting negative log-likelihood functions is convex. In multinomial logit, a shared vector of parameters across the population are fitted in the model. The model implies that different victims, given the same observed accident factors, will always suffer the same degree of injury. In reality, the nature of accident severity cannot be fully specified. There are many unobserved factors that influence the outcome of an accident. For example, effect of age may depend health conditions of individuals. However, information such as prior health conditions are typically not collected in the accident report, creating unobserved heterogeneity. The unobserved heterogeneity can lead to diverse accident outcomes. Therefore, accounting for the effect of unobserved heterogeneity on severity of crashes is an important issue in analyzing accident data [106]. Incorporating randomness in the parameters is a way to account for the unobserved heterogeneity.

Instead of fitting fixed parameters across the population, the mixed logit model assumes the parameters also follow a distribution. Therefore, given a same vector of observed factors, the randomness in the parameters in different samples give rise to a diverse range of potential severity levels. Analyzing the fitted distributions of the parameters in mixed effect logistic model assists in the discovery of population heterogeneity with respect to different accident factors. For instance, when Gaussian assumptions is placed on the parameters, a small variance implies the effect of a factor does not vary much in the population.

Fitting fixed effect or mixed effect logit model is usually performed by optimizing the log-likelihood or marginal log-likelihood function. In this paper, we focus on scalable algorithms to solve the underlying optimization problem from logit model parameter estimation on big datasets. The optimization problem is often overlooked in traffic safety domain, despite of the popularity in applying this models for analyzing accident data. Although many nonlinear optimization algorithms can be adapted to solve the problem arisen from parameter estimation for logit models, not all of them are scalable to large datasets. Stata [107], NLOGIT [108], and LIMDEP [108] are arguably the most commonly chosen software packages for estimating logit models for traffic safety analytic usages. Newton's method or Quasi-Newton method with BFGS or BHHH algorithms [33–36] are used as the optimization routines in these packages. While Newton's method and Quasi-Newton methods enjoys superlinear convergence rate, the computational complexity per iteration for these algorithms grows with the sample size in the dataset due to the need for computing gradient directions from each sample. Therefore not surprisingly, overall running time for fitting logit models with these algorithm will be dependent on the size of datasets. Large accident samples are available today due to advances in data collection technologies. Gaining insights into big data may confirm or invalidate findings from small samples. Supporting scalable model estimation is a fundamental step for studying traffic crash factors on big datasets.

5.1 Related Work

Research in mixed logit estimation roughly fall into two broad categories, studying different estimators and improving the optimization efficiency given a type of estimators as the objective function. This paper belongs to the second category.

There is a large body work in different statistical estimators for mixed logit model, see [12–18, 25, 31, 32, 101, 109–111] and the references therein. Mixed logit models are typically fitted with simulation-assisted estimation under either the frequentist or the Bayesian paradigm. Due to the presence of unobserved random effects in the model, marginalization of the random effects by simulation are one of the most popular approaches. The Maximum Simulated Likelihood (MSL) estimator is based on this idea [15, 25, 31, 32]. Solving the optimization problem arisen from the estimator is equally important as the estimator itself. In this work, we focus on the optimization aspect for the Maximum Simulated Likelihood estimation. MSL uses simulation to approximate the high dimensional integrals resulting from marginalization of the random effects. In general, the approximation accuracy increases with the number of samples generated by the simulator. At the same time, the computational cost for solving the optimization problem grows with both the simulation size and the sample size of dataset. With the same number of simulation draws, controlling the approximation error for the expectation operator requires trade-offs between bias and variance of the simulator. There is a large body of work on Quasi-Monte Carlo methods [112–115] (and the references therein), which aims to reduce the variance of simulation and makes less number of draws required in practice. In addition to producing better simulation procedures, several authors considered optimization algorithms tailored to the Maximum Simulated Likelihood objective, and compared the optimization efficiency of different algorithms. These papers are close to ours in spirit. Bastin et al. proposed an adaptive Trust-Region Monte Carlo algorithm for estimating mixed logit from the MSL objective, where each trust-region subproblem is constructed with varying number of simulations [116]. The optimization performance of variants of trust-region method and Quasi-Newton algorithms are compared in [117] and [118]. In [116–118], the algorithms were tested on datasets with a few thousands observations. We aim to scale mixed logit estimation on much larger datasets. In addition, the models in [116–118] are specified for choice analysis from survey data. Therefore, the covariates are alternative-specific but parameters are universal for all choices. On the contrary, there are no alternative-specific covariates in accident records typically, and alternative-specific parameters need to be specified. Therefore, the dimension of optimization variables are larger in the latter case. Multiple local maxima (or minima) may exists, due to the non-convexity of Maximum Simulated Likelihood. Hole and Yoo studied the impact of initialization by the use of heuristic algorithms [119].

5.2 Background

5.2.1 Logit Model

We describe the fixed effect and mixed effect multinomial logit models in this section. Given N observations of vehicle crashes, let the set $\{1, \cdots, I\}$ represent different injury classes of the accident outcome. Let $P(n_c = k)$ denote the probability of accident n resulting in severity category k. For each observation, $\mathbf{x}_n \in \mathbb{R}^p$ is a p-dimensional vector of variables describing the accident characteristics. For example, $\mathbf{x}_n$ can be gender and age of victims, alcohol influence, weather conditions, and other information about the accident. In the multinomial logit models, the probability of suffering injury severity class k in an accident given $\mathbf{x}_n$ is

$$P(n_c = k | V_{nj}, \mathbf{x}_n) = \frac{\exp(V_{nk})}{\sum\limits_{j=1}^{I} \exp(V_{nj})} \tag{5.1}$$

where $\{V_{nj}\}_{j=1}^{I}$ are linear predictors for each injury class. In the fixed effect case,

$$V_{nj} = \alpha^{(j)} + \boldsymbol{\beta}^{(j)^T} \mathbf{x}_n, \tag{5.2}$$

where $\alpha^{(j)}, j = 1, \cdots, I$ are unknown class-specific intercepts, $\boldsymbol{\beta}^{(j)} \in \mathbb{R}^p$ are class-specific parameters to be estimated. Since $\{\boldsymbol{\beta}^{(j)}\}_{j=1}^{I}$ are shared across different observations, accidents with the same observed vector $\mathbf{x}$ will result in identical distribution of injury probability, despite that distinct consequences from accidents may be suffered even under the same observed crash situation. To remove this restriction, the mixed logit model allows randomness in the parameters to capture the individual heterogeneity,

$$\begin{aligned} V_{nj} &= \alpha_n^{(j)} + \boldsymbol{\beta}_n^{(j)^T} \mathbf{x}_n, \\ \text{where } \boldsymbol{\beta}_n &:= [\boldsymbol{\beta}_n^{(1)}, \cdots, \boldsymbol{\beta}_n^{(I)}] \in \mathbb{R}^{pI}, \\ \boldsymbol{\alpha}_n &:= [\alpha_n^{(1)}, \cdots, \alpha_n^{(I)}] \in \mathbb{R}^p, \\ (\boldsymbol{\alpha}_n, \boldsymbol{\beta}_n) &\sim f(\boldsymbol{\alpha}, \boldsymbol{\beta} | \boldsymbol{\Psi}) \end{aligned} \tag{5.3}$$

$\boldsymbol{\alpha}_n$ and $\boldsymbol{\beta}_n \in \mathbb{R}^{pI}$ are random intercepts and random parameters for each observation n. The joint distribution of $\boldsymbol{\alpha}_n$ and $\boldsymbol{\beta}_n$ is specified by the mixing distribution $f(\boldsymbol{\alpha}, \boldsymbol{\beta}|\boldsymbol{\Psi})$, where $\boldsymbol{\Psi}$ is an unknown parameter for the mixing distribution f. The randomness in $\boldsymbol{\beta}_n$ captures the unobserved heterogeneity in an accident, such that two observations are allowed to have different outcomes even if $\mathbf{x}_n = \mathbf{x}_{n'}$. The differences in crash severity outcome can be due to unique factors in each accident that are not observed by the analyst. For example, the health condition for people of the same age may vary, but these information are typically not collected in the accident report. In this paper, we make the following blockwise Gaussian assumptions in the mixing distribution:

$$\begin{aligned} &\boldsymbol{\beta}_n^{(i)} \perp \boldsymbol{\beta}_n^{(j)}, \text{ if } i \neq j \\ &\boldsymbol{\beta}_n^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}^{(i)}) \end{aligned} \tag{5.4}$$

We further assume the intercept $\boldsymbol{\alpha}$ are non-random variables. Gaussian, truncated Gaussian, log-normal, triangular, uniform distribution are among the popular choices of mixing distributions used in traffic safety research. We focus on blockwise Gaussian mixing distribution in this paper, though the method can be applied to any distributions listed above.

5.2.2 Parameter Estimation

Estimation for multinomial logit model is typically carried out via the Maximum Likelihood Estimator.

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_{n=1}^{N} \prod_{j=1}^{I} P(n_c = j|\boldsymbol{\alpha}, \boldsymbol{\beta})^{t_{nj}}, \tag{5.5}$$

where $t_{nj} = 1$ if severity outcome j is observed on accident n, and $t_{nj} = 0$ otherwise. The estimator $(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\beta}}^*)$ for parameters in fixed effect logit model is obtained from

$$(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\beta}}^*) \in \operatorname{argmax} \log \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{5.6}$$

Note that in the mixed effect case, the conditional probability $P(n_c|\boldsymbol{\alpha}, \boldsymbol{\beta}_n, \boldsymbol{\Psi})$ depends on unobserved random parameters $\boldsymbol{\beta}_n$. In traffic safety analysis, researchers typically care about the distribution of $\boldsymbol{\beta}$ in order to understand the degree of heterogeneity in the population with respect to different accident factors. Therefore, $\boldsymbol{\beta}_n$ is integrated out to yield the marginal probability:

$$P(n_c = j|\boldsymbol{\alpha}, \boldsymbol{\Psi}) = \mathbb{E}[P(n_c = j|\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\Psi})] = \int P(n_c = j|\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\Psi}) f(\boldsymbol{\beta}|\boldsymbol{\Psi}) d\boldsymbol{\beta}. \quad (5.7)$$

Accordingly, the marginal likelihood for mixed logit model based on (5.7) is

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\Psi}) = \prod_{n=1}^{N} \prod_{j=1}^{I} P(n_c = j|\boldsymbol{\alpha}, \boldsymbol{\Psi})^{t_{nj}}. \quad (5.8)$$

The estimator for the mixing distribution parameters are obtained by

$$(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\Psi}}^*) \in \operatorname{argmax} \log \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\Psi}) \quad (5.9)$$

Equation (5.7) contains an high dimensional integral without closed-form solutions, which requires approximation. In the next section, we review the popular Maximum Simulated Likelihood as an approximation to (5.9).

5.2.3 Optimization

In this section, we briefly describe popular optimization methods used by traffic safety modeler for optimizing the objective function arisen from fitting fixed effect logit model (5.6) and mixed logit model (5.9). Note that the maximization problem can be written as minimizing $-\mathcal{L}$. For subsequent presentation, we adapt the equivalent minimization problem. In the fixed effect logit model, let

$$(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\beta}}^*) \in \operatorname{argmin} \sum_{n=1}^{N} \ell_n(\boldsymbol{\alpha}, \boldsymbol{\beta}), \quad (5.10)$$

where $\ell_n(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\log P(n_c = t_n|\boldsymbol{\alpha}, \boldsymbol{\beta})$, t_n is the severity result for observation n. The optimization objective function (5.10) is a finite sum of N partial objective functions contributed from each accident sample $(\mathbf{x}_n, t_n)$. Let $\boldsymbol{\Theta} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$, the Newton-Raphson iteration for solving (5.10) is

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \Big(\sum_{n=1}^{N} \mathbf{H}_{\ell_n}(\boldsymbol{\Theta}_t)\Big)^{-1} \Big(\sum_{n=1}^{N} \nabla \ell_n(\boldsymbol{\Theta}_t)\Big). \quad (5.11)$$

Here, $\mathbf{H}_{\ell_n}(\mathbf{\Theta}_t)$ is the Hessian matrix of ℓ_n with respect to $\mathbf{\Theta}_t$. In Quasi-Newton methods, the Hessian matrices $\mathbf{H}_{\ell_n}(\mathbf{\Theta}_t)$ are approximated by the BFGS formula or the BHHH formula [33–36] instead of being computed exactly to reduce the computational cost. Both the BFGS and the BHHH updates are supported in Stata [107] and NLogit [108] and widely used. The optimization procedure discussed above is not limited to fixed effect logit model. These methods are also applied on fitting mixed logit model with the help of simulation, as we will show in the following. Recall the mixed logit model defined in equation (5.3) and (5.4). $\mathbf{\Theta}$ is specified by a distribution instead of a fixed value. Therefore, the goal is to fit the parameter $\mathbf{\Psi}$ governing the distribution of $\boldsymbol{\beta}$. Since $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$, $\boldsymbol{\beta}$ can be decomposed as

$$\boldsymbol{\beta} = \boldsymbol{\mu} + \mathbf{\Gamma}\boldsymbol{\eta},\ \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),\ \mathbf{\Gamma}\mathbf{\Gamma}^T = \mathbf{\Sigma}, \tag{5.12}$$

where $\mathbf{\Gamma} \in \mathbb{R}^{pI \times pI}$ is the Choleskey factor of the covariance matrix. Therefore, the high-dimensional intergral in the marginal probability (5.7) can be approximated via Sample Average Approximation (SAA):

$$\widetilde{P}(n_c = k|\boldsymbol{\alpha}, \boldsymbol{\mu}, \mathbf{\Gamma}) = \frac{1}{R}\sum_{r=1}^{R} \frac{\exp\left(\alpha^{(k)} + \left(\boldsymbol{\mu}^{(k)} + \mathbf{\Gamma}^{(k)}\boldsymbol{\eta}_{n,r}^{(k)}\right)^T \mathbf{x}_n\right)}{\sum\limits_{j=1}^{I} \exp\left(\alpha^{(j)} + \left(\boldsymbol{\mu}^{(j)} + \mathbf{\Gamma}^{(j)}\boldsymbol{\eta}_{n,r}^{(j)}\right)^T \mathbf{x}_n\right)} \tag{5.13}$$

where $\{\boldsymbol{\eta}_{n,r}^{(j)}\}_{r=1}^{R}$, $j = 1, \cdots, I$ are random draws from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{p\times p})$. Quasi-random simulations [112, 113] may also be used, without affecting the representation in (5.13). Hence, instead of optimizing the marginal log-likelihood (5.9), the Maximum Simulated Likelihood (MSL) is used:

$$(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\mu}}^*, \hat{\mathbf{\Gamma}}^*) \in \operatorname{argmin} \sum_{n=1}^{N} -\log \widetilde{P}(n_c = t_n|\boldsymbol{\alpha}, \boldsymbol{\mu}, \mathbf{\Gamma}) := \sum_{n=1}^{N} \tilde{\ell}_n(\boldsymbol{\alpha}, \boldsymbol{\mu}, \mathbf{\Gamma}). \tag{5.14}$$

Newton's method or Quasi-Newton method with BFGS approximation discussed earlier in this section can be now applied on (5.14) to obtain the estimated parameters for mixed logit model.

5.2.4 Limitation

Newton's method and Quasi-Newton methods enjoy superlinear convergence rate. However, the complexity per iteration grows with the number of samples in the dataset due to the computation cost of the gradients:

$$\text{cost of computing } \left(\sum_{n=1}^{N} \nabla \ell_n \right) \approx N \times \text{ cost of computing } \Big(\nabla \ell_n \Big). \tag{5.15}$$

Therefore, despite of the potential superlinear convergence rate from using Newton or Quasi-Newton methods, the time per iteration will be excessively long for large N. Hence, the total complexity of Newton's method and Quasi-Newton methods are dependent on the sample size and difficult to scale up on large datasets. Finite-sum optimization, similar to (5.10), frequently ocurrs in machine learning problems. Stochastic Gradient Method (SGM) gains extensive study in machine learning literature recently, due to its low complexity per iteration for minimizing finite-sum of objective functions [19–24]. In the next section, we explain SGM on fixed effect logit model. We then propose a combination of Sample Average Approximation and Stochastic Gradient for mixed logit model estimation.

5.3 Stochastic Optimization for Logit Models

5.3.1 Stochastic Gradient Method

Stochastic Gradient Method was originally proposed by Robbins and Monro [120] in stochastic approximation literature. Despite of the large body of work on SGM for machine learning applications [19–24] (and references therein), this approach is still largely ignored in traffic safety logit modeling and discrete choice literature. To the best of our knowledge, stochastic optimization algorithms for estimating logit/discrete choice models in transportation was studied only in [121]. In addition, unlike most machine learning models, the objective function arisen from mixed logit estimation depends on Monte Carlo simulation (5.14). Hence, it is important to perform empirically examination of the performance of SGM applied on simulation-based problems, such as Maximum Simulated Likelihood optimization. We describe SGM in this section.

In stochastic optimization, the goal is to solve

$$\min\{F(\boldsymbol{\Theta}) := \mathbb{E}[f(\boldsymbol{\Theta}, \xi)]\} \tag{5.16}$$

where ξ is a random vector from an unknown distribution. Even though the distribution which generates ξ is unknown, we assume that:

(A1) It is possible to generate i.i.d samples $\xi_1, \xi_2, ...$, of the realization of ξ.

(A2) For every given ξ and $\boldsymbol{\Theta}$, $f(\boldsymbol{\Theta}, \xi)$ can be computed. Moreover, for each ξ and $\boldsymbol{\Theta}$, the *stochastic gradient* $g(\boldsymbol{\Theta}, \xi) := \nabla f(\boldsymbol{\Theta}, \xi)$ is well defined and unbiased, i.e., $\mathbb{E}[g(\boldsymbol{\Theta}, \xi)] = \nabla F(\boldsymbol{\Theta})$.

These are standard assumptions in stochastic optimization literature [19, 22, 122]. Stochastic Gradient Method (SGM) performs the following iteration:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \gamma_t g(\boldsymbol{\Theta}_t, \xi_t),\ t = 1, 2, ... \tag{5.17}$$

where ξ_t's are i.i.d. observations of the random variable ξ sampled at iteration t, $\{\gamma_t\}_{t=1}^{\infty}$ is the step-size sequence. In addition to update $\boldsymbol{\Theta}_t$ from the stochastic gradient computed from a single observation of ξ, a mini-batch gradient from multiple samples $\{\xi_{t,i}\}_{i=1}^{\mathcal{B}}$ is also valid since the average of $|\mathcal{B}|$ stochastic gradients is also unbiased:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \frac{\gamma_t}{|\mathcal{B}|} \sum_{i=1}^{\mathcal{B}} g(\boldsymbol{\Theta}_t, \xi_{t,i}),\ t = 1, 2, ... \tag{5.18}$$

Problem in (5.10) and (5.14) are can be casted as stochastic optimization problems solvable by SGM due to the finite-sum nature of the objective functions. To see that,

$$\operatorname{argmin} \sum_{n=1}^{N} \ell_n = \operatorname{argmin} N^{-1} \sum_{n=1}^{N} \ell_n = \operatorname{argmin} \mathbb{E}[\ell_n] \tag{5.19}$$

where ℓ_n are sampled from $\{\ell_n\}_{n=1}^{N}$ uniformly at random. Moreover,

$$\nabla \mathbb{E}[\ell_n] = \nabla\Big(N^{-1} \sum_{n=1}^{N} \ell_n\Big) = N^{-1} \sum_{n=1}^{N} \nabla \ell_n = \mathbb{E}[\nabla \ell_n] \tag{5.20}$$

Therefore, assumption (A1) and (A2) are both satisfied. SGM for solving the fixed

Algorithm 5 Stochastic Gradient Method

```
Input: number of iterations $T$, step-size sequence $\{\gamma\}_{t=0}^{T}$
Output: an approximate solution for (5.10).
for $t = 0$ to $T$ do
    subsample a data $(t_{i_t}, \mathbf{x}_{i_t})$ from the dataset
    $\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \gamma_t \nabla \ell_{i_t}(\mathbf{\Theta}_t)$
end for
```

Algorithm 6 Stochastic Gradient with Sample Average Approximation for Mixed Logit Parameter Estimation

```
Input: number of iterations $T$, step-size sequence $\{\gamma\}_{t=0}^{T}$, simulation size $R$
Output: an approximate solution for (5.14).
for $t = 0$ to $T$ do
    subsample a data $(t_{i_t}, \mathbf{x}_{i_t})$ from the dataset
    simulate random draws $\{\boldsymbol{\eta}_{i_t,r}^{(j)}\}_{j=1}^{I} \sim \mathcal{N}\Big(\mathbf{0}, \mathbf{I}\Big)$, $r = 1, ..., R$
    $\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \gamma_t \nabla \widetilde{\ell}_{i_t}(\mathbf{\Theta}_t)$
end for
```

effect logit parameter estimation problem is given in Algorithm (5).

For strongly-convex functions f, SGM with step-sizes $\gamma_t = O\Big(\frac{1}{t}\Big)$ requires $O\Big(\frac{1}{\epsilon}\Big)$ iterations to reach ϵ accuracy, i.e., $\mathbb{E}[f(\mathbf{\Theta}, \xi) - f(\mathbf{\Theta}^*, \xi)] \leq \epsilon$ [122]. In comparison, (deterministic) gradient descent requires $\log\Big(\frac{1}{\epsilon}\Big)$ iterations and Newton's method only requires only $\log\Big(\log\Big(\frac{1}{\epsilon}\Big)\Big)$ iterations [37]. However, gradient descent, Newton's method and Quasi-Newton methods (5.11) require $O(N)$ computations of the gradients $\sum_{n=1}^{N} \nabla \ell_n$ and additional computational cost on the Hessian or approximate Hessian matrix for Newton-variants. In contrast, the number of gradient computations per iteration with SGM is independent of the size of the dataset (Algorithm 5). Therefore, SGM brings advantage over Newton's method or Quasi-Newton methods when $N \geq \Omega\Big(\Big(\epsilon \log\log\Big(\frac{1}{\epsilon}\Big)\Big)^{-1}\Big)$, i.e., on large datasets. Note that estimation of the multinomial logit model by maximum likelihood gives rise to a convex objective (5.10), and estimation via Ridge estimator (ℓ_2-penalized likelihood) [105, 123] results in a strongly-convex formulation.

5.3.2 Stochastic Gradient with Sample Average Approximation

We now extend the Stochastic Gradient Method introduced in the last section to mixed logit model, where the objective function involves high-dimensional integrals. We propose a combination of stochastic gradient and Sample Average Approximation (SAA). The pseudo-code is provided in Algorithm (6). There are two different options in the implementation of the simulation step in line 3 of Algorithm (6). The first option is to keep the same random sequence $\{\boldsymbol{\eta}_{n,r}^{(j)}\}_{j=1}^{I}$ for each $n \in \{1, ..., N\}$ (but different sequences if $n_1 \neq n_2$). Another options is to generate a new independent sequence in each iteration, and independent of the data $(t_{i_t}, \mathbf{x}_{i_t})$ at which the stochastic gradient is computed on. The first option is essentially first transforming (5.9) into the Maximum Simulated Likelihood problem (5.14), then solving the sample average approximated problem with SGM. The second option is equivalent to applying SGM and produce Monte Carlo simulation dynamically to approximate the sub-sampled objectives on-the-fly independent of previous draws. We left the convergence analysis of SGM with dynamic Monte Carlo approximation to future work.

5.3.3 Choice of Step-size Sequence and Practical Implementation

For convex but non-strongly-convex functions $f(\cdot, \xi)$, SGM with step-size sequences $\gamma_t = O\Big(\frac{1}{\sqrt{t}}\Big)$ finds a solution $\boldsymbol{\Theta}$ such that $\mathbb{E}[f(\boldsymbol{\Theta}, \xi) - f(\boldsymbol{\Theta}^*, \xi)] \leq \epsilon$ in $O(\frac{1}{\epsilon^2})$ iterations [122, 124]. For non-convex functions, [22] show that SGM achieves the first order optimality condition $\mathbb{E}[\|\nabla f(\boldsymbol{\Theta}, \xi)\|^2] \leq \epsilon$ in $O\Big(\frac{1}{\epsilon^2}\Big)$ iterations, by applying either the constant step-size policy or decreasing step-size policy. Although the theoretical convergence rate is established for any step-sizes meeting the aforementioned requirements, in practice the convergence speed can be significantly impacted by the specific step-size scheme being used. In the implementation, we use without replacement sampling to obtain the stochastic gradients. Choices of step-size sequence is discussed below.

Decreasing step-size per iteration: a popular choice of step-sizes is setting $\gamma_t = c/(1+t)^z$ for $0 < z \leq 1$ and $c > 0$, where t denotes the current iteration. Although the $O\Big(\frac{1}{\epsilon^2}\Big)$ iterations required for convergence mentioned above is obtained for the diminishing step-size rules, the iterates can be stalled to reach the optimality conditions in practice. Suppose $\boldsymbol{\Theta}^* \in C$, such that $\|\nabla \ell_n(\boldsymbol{\Theta})\| \leq M$ for all $\boldsymbol{\Theta} \in C$, for all $n \in \{1, ..., N\}$. When a decreasing step-size sequence $\{\gamma_t\}_{t=1}^{\infty}$ is applied on

SGM, after the t-th iteration such that

$$\|\gamma_t \nabla \ell_{i_t}(\mathbf{\Theta}_t)\| \leq \gamma_t \|\nabla \ell_{i_t}(\mathbf{\Theta}_t)\| \leq \epsilon_{mach} \tag{5.21}$$

where ϵ_{mach} is the machine precision, the iterate $\mathbf{\Theta}_{t+1}$ is indistinguishable from $\mathbf{\Theta}_t$ and the iterations are stalled. This does not necessary certifies the first order optimality. The update can become arbitrarily small simply due to the shrinking step-size γ_t.

Decreasing step-size per epoch: In the implementation of Algorithm (5) and Algorithm (6), we can group the iterations into *epochs* such that each epoch consists of N iterations. Therefore, each epoch performs N gradient computations chosen in stochastic manner, which is equivalent to the number of gradient computations performed in an iteration of a deterministic algorithm. The step-size is reduced after each epoch instead of each iteration. An epoch-wise decreasing schedule preserves the asymptotic of step-sizes used in the theoretical analysis, but delays the iterates from reaching the stalling stage.

Phase transition and function value adaptive step-sizes: In many problems, practitioners have observed fast reduction in the objective value in the early iterations of Stochastic Gradient Method, followed by a phase when the convergence slows down and the objective value moves in non-monotone directions [19, 125]. This is due to the stochastic nature of the algorithm. Formally, suppose a constant step-size γ for SGM is used, and suppose the objective function is convex (satisfied by eqn. (5.10)) and meets some smoothness assumptions, then for each iterate $\mathbf{\Theta}_t$, it holds that

$$\mathbb{E}[\|\mathbf{\Theta}_t - \mathbf{\Theta}^*\|^2] \leq \mathbb{E}[\|\mathbf{\Theta}_0 - \mathbf{\Theta}^*\|^2] \exp\Big(- A_\gamma t\Big) + B\gamma \tag{5.22}$$

where $\mathbf{\Theta}^*$ is the global minimum of objective function, A_γ is a positive constant depending on the step-size γ and the objective function, B is a positive constant independent of γ. Proof for equation (5.22) has been given by several authors, see [20, 126, 127]. When constant step-size is used with SGM, (5.22) implies the *phase-transition* phenomenon during the optimization process. Initially, the iterate $\mathbf{\Theta}_t$ converges to $\mathbf{\Theta}^*$ exponentially fast, suggested by the term $\exp\Big(- A_\gamma t\Big)$. The error term $B\gamma$ does not decrease with t and dominates the right hand side of (5.22) when $\|\mathbf{\Theta}_t - \mathbf{\Theta}^*\|^2$ is sufficiently small, suggesting the iteration enters a stationary phase. Equation (5.22) also suggests the following adaptive step-size strategy: keeping a constant step-size within each epoch and monitor the objective function values at the end of an epoch. Halving the step-size if the function value increases compared to the previous epoch, otherwise the step-size is unchanged. The intuition behind the adaptive strategy is to gain the benefit of exponentially fast convergence of SGM before the stationary phase, but reducing the step-size in order to make progress once the iterates are stalled. More theoretical analysis on this adaptive strategy is given by [128].

5.4 Case Studies

In this section, we compare different optimization algorithms for estimating multinomial logit model (5.2) and mixed logit model (5.4) on large traffic accident datasets across multiple years. The following algorithms are tested.

For multinomial logit model estimation:

- **Stochastic Gradient Method (SGM)** with different step-size schemes:

  - constant step-size scheme.
  - decreasing step-size after each iteration, i.e., $\gamma_t = \frac{1}{1+t}$.
  - decreasing step-size after each epoch, i.e., $\gamma_t = \frac{1}{1+epoch}$.
  - adaptive step-size according to the objective function value at the end of each epoch (section 5.3.3).

  We implement SGM in C++ and MATLAB.

- **Nesterov Accelerated Gradient Descent (AGD)**, an optimal first-order method for convex optimization [129]. For general convex problems, AGD requires $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ iterations to find a solution no greater than ϵ from the optimum. We implement AGD in C++. Constant step-size is used.

- **BFGS Quasi-Newton with line search (BFGS)**: this method is commonly used in statistical packages, for example, Stata [107], NLOGIT [108], and LIMDEP [108], and open-source MATLAB package supplied in [18]. To obtain various experimental measurements during the optimization process, we use the `fminunc` function in MATLAB with `algorithm` set to `bfgs` [130].

We tested the following algorithms for estimating mixed logit model:

- **Stochastic Gradient Method + Sample Average Approximation (SGM-SAA)**. Different step-size schemes are tested.
- **Non-convex Accelerated Gradient + Sample Average Approximation (AGD-SAA)**. Nesterov Accelerated Gradient Descent was proposed for convex functions. Here we tested a modification of AGD designed for non-convex situations [131].
- **BFGS Quasi-Newton + Sample Average Approximation (BFGS-SAA)**: `fminunc` solver in MATLAB is used [130]. Pseudo-random standard normal distributions are generated by `mvnrnd` function in MATLAB [130].

The `boost` library is used for pseudo-random number generation [132] in the C++ code.

5.4.1 Data

Traffic accident records from The Statewide Integrated Traffic Records System (SWITRS) [133] in California are used in the experiment. It has been reported that single-vehicle crashes account for nearly 30% of all vehicle crashes in the United States in 2015 [134]. We estimated multinomial logit and mixed logit models on single-vehicle crash accident cases from 2003 to 2013 extracted from SWITRS. The set of injury categories are {`no injury or only complaint of pain, visible injury, severe injury, fatal`}. The models are estimated with seventeen variables listed in Table 5.1. All variables are processed into binary vector $\mathbf{x}_n$'s using dummy coding, in which variable x_{ni} is 0 if it belongs to the reference level in Table 5.1, and 1 otherwise. The number of observations in each two-year period are shown in Table 5.2.

5.4.2 Convergence Behavior

The empirical convergence behavior during the optimization process in different algorithms is studied in this section. Note that in stochastic gradient-based algorithms, each iteration requires computation of gradients on a single observation chosen at random, i.e., $\nabla \ell_{i_t}$, where as an iteration of deterministic gradient-based

feature ID	0	1	2	3	4	5	6	7
Description	victim gender	victim age	seat-belt used	alcohol used	cell-phone used	wrong side of road	improper tuning	over speed
reference level	female	25-64	false	false	false	false	false	false

feature ID	8	9	10	11	12	13	14	15	16
Description	weather	light conditions	wet road	drug used	vehicle age	AM peak	PM peak	week -end	season
reference level	clear	daylight	false	false	0-10	false	false	false	spring/ summer

Table 5.1. Accident-related features used in the experiment

years	03-04	04-05	05-06	06-07	07-08
N	90307	138363	134729	129187	122142
years	08-09	09-10	10-11	11-12	12-13
N	114579	109121	104040	100783	95111

Table 5.2. Number of observations in each two-year period between 2003 and 2013.

algorithm consists of gradient computation over the entire dataset, i.e., $\sum_{n=1}^{N} \nabla \ell_n$. Therefore, we measure the progress of different algorithms based on number of passes over the dataset:

$$\text{A pass over dataset} := N \text{ computations of } \nabla \ell_n, \text{ for arbitrary } n \in \{1, 2, ..., N\}.$$

We aim to compare the convergence behavior under a computational budget constraint, where the budget is measured by the number of times an algorithm processes the dataset or computation time. As described in section 5.3.3, the stochastic iterations are combined into unit of epochs. Each epoch consumes the computational budget of one pass over the dataset. The objective value $\frac{1}{N}\sum_{n=1} \ell_n(\boldsymbol{\Theta})$ is recorded after each full pass. The objective value represents the negative log-likelihood averaged over the number of observations for multinomial logit model, and is the averaged negative simulated log-likelihood for mixed logit model. Note that the $\frac{1}{N}$ factor does not change the estimator or optimal parameters

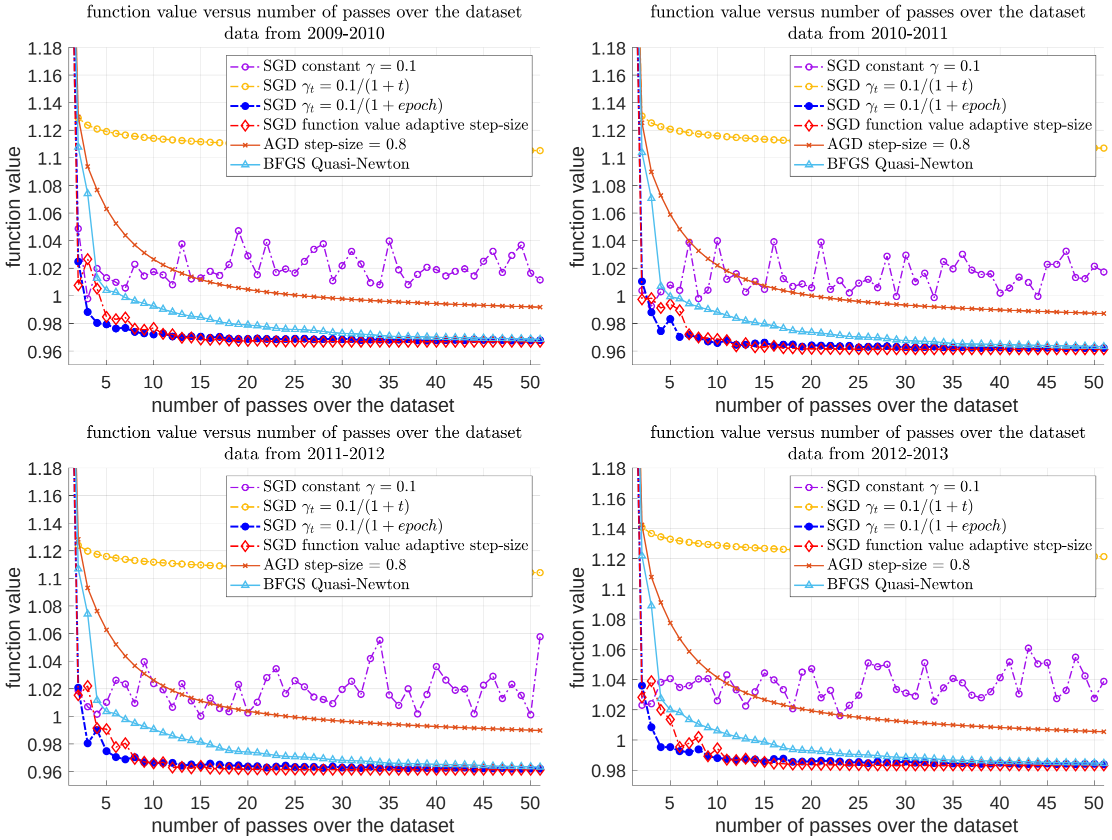


Figure 5.1. Objective value after each dataset pass for multinomial logit model.

since N is a constant for each dataset, but standardized the range of function values on different subsets of data. The initial step-size γ_0 for all variants of SGM, and the step-size for AGD requires tuning. We trial different initial step-sizes among $\{0.1, 0.5, 0.8, 1\}$, and use the one with lowest objective values after five dataset passes to continue the experiments. Each algorithm is run for ten times. Figure 5.1 shows the mean of objective values for multinomial logit at the end of each dataset pass for accidents datasets between 2009 and 2013. The mean of objective values versus number of passes over the dataset for mixed logit model is shown in Figure 5.2.

For the convex optimization problem arisen from multinomial logit model estimation, BFGS with line-search, SGM with epoch-wise decreasing step-sizes and SGM with adaptive step-sizes converge to the same function value within 50 dataset passes. The objective value descents in the first few epochs for SGM with decreasing step-size per iteration, but gradually flattens above the optimum. This

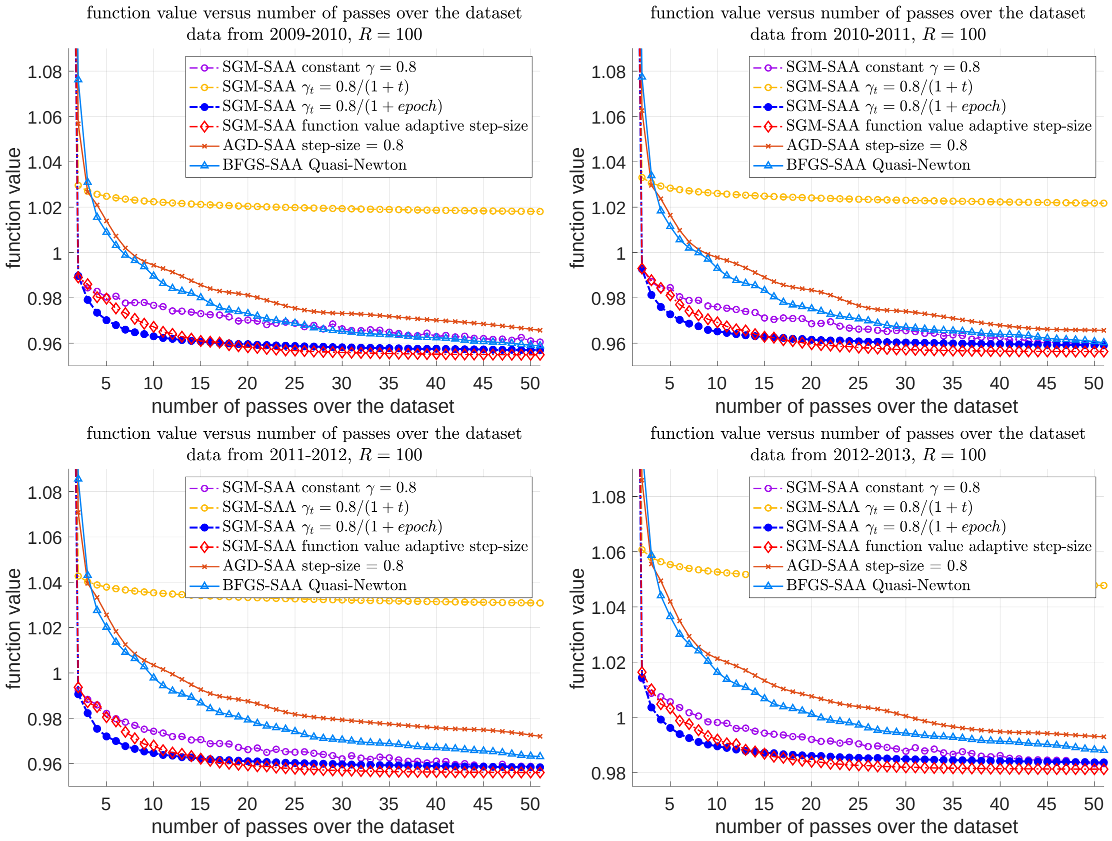


Figure 5.2. Objective value of mixed logit model after each pass over the dataset.

is not surprising, due to the shrinking step-sizes as suggested in equation (5.21). SGM with constant step-sizes exhibits noisy fluctuations above the optimum, albeit offering an initial descent. This implies the iterates have entered a stationary phase, corresponds to equation (5.22). AGD also fails to converge within the first 50 passes over the datast, and the convergence slows down after reaching function values at a similar range to which SGM with constant step-sizes becomes noisy. SGM with adaptive step-sizes experiences large initial descent, followed by a brief phase when the function value moves in noisy directions compared to epoch-wise decreasing step-sizes, but eventually converges smoothly. Although BFGS gradually reaches the same objective value as the two converging SGM variants, the reduction in function value is smaller than the SGM variants within the first 40 dataset passes. The comparison between BFGS and SGM *under the same computational budget* demonstrates the trade-off between complexity per iteration and convergence rate. The complexity per iteration in SGM is independent of N. Each pass of the dataset in SGM produces N iterations, whereas only one BFGS iteration can be computed in a full pass of dataset. Therefore, when N is large and the computational budget is restricted, the optimization error of SGM variants is smaller than BFGS.

The optimization problem for mixed logit estimation is non-convex and the objective function depends on Monte Carlo simulation. When the experiments were run on the datasets in Table 5.2, a single test run of the `bfgs` algorithm in MATLAB with maximum 50 iterations and $R = 500$ Monte Carlo draws takes more than 20 hours to complete. Therefore, in order to keep the overall computational budget reasonable, we sub-sampled 20000 observations from each dataset to perform the experiments for optimizing mixed logit model. The objective value in mixed logit model versus number of dataset passes are displayed in Figure 5.2. For this non-convex problem, SGM with constant step-sizes yields diminishing function values after each epoch in the first 50 dataset passes, but at a slower rate compared to SGM with epoch-wise decreasing step-size or adaptive step-size. Unlike in the convex case, SGM with adaptive step-size descents smoothly. Note that if the step-size adaptation criterion is not triggered at all, adaptive step-sizes rule will produce the same iterates as constant step-sizes in expectation. The gaps between SGM with adaptive step-sizes and with constant step-sizes in Figure 5.2 indicate that adjusting the step-sizes according to the function value criterion indeed helps the empirical convergence speed for mixed logit estimation. In addition, adaptive step-size SGM found the lowest objective value after 50 passes over the dataset among the methods, but the differences are small. This is a sign that different optimizers converge to flat regions where the gradient magnitude is small, as demonstrated in Figure 5.3, and further descent becomes slow. Similar to the multinomial logit case, SGM-variants reduce the function value faster than BFGS (except for dataset 2009-2010 after 25 passes), given the same number of passes allowed to process the dataset. The objective values against computation time budgets are shown in Figure 5.4. Due to the light computation complexity per iteration, Stochastic Gradient Method achieves lower objective values given a fixed amount of time compared to BFGS Quasi-Newton method and Accelerated Gradient Descent. Stochastic gradient-based algorithms brings additional stochasticity into the estimation, in addition to the randomness due to Monte Carlo simulation in the objective function. Therefore, it is not surprise to find higher variance in the estimates obtained by SGM. To quantify the stochasticity introduced by SGM, the standard deviation of estimated parameter from ten independent runs of SGM-SAA and AGD-SAA are computed. The average standard deviation of estimated parameters in the intercept $\boldsymbol{\alpha}$, mean $\boldsymbol{\mu}$, and covariance matrix $\boldsymbol{\Sigma}$ are given in Table 5.3. The Likelihood Ratio index (McFadden's pseudo-R^2), i.e., $1 - \log \tilde{\mathcal{L}}(\hat{\boldsymbol{\Theta}}) / \log \tilde{\mathcal{L}}(\mathbf{0})$ [16, 135, 136], from different algorithms after 50 passes over the dataset are also reported in Table 5.3. SGM-SAA variants suffer higher variance of estimates compared to deterministic Accelerated Gradient Descent, although the SGM-based optimizers exited at solutions with better goodness-of-fit on average. Therefore, the improved optimization efficiency of SGM for large N are gained at the cost of less stable estimates.

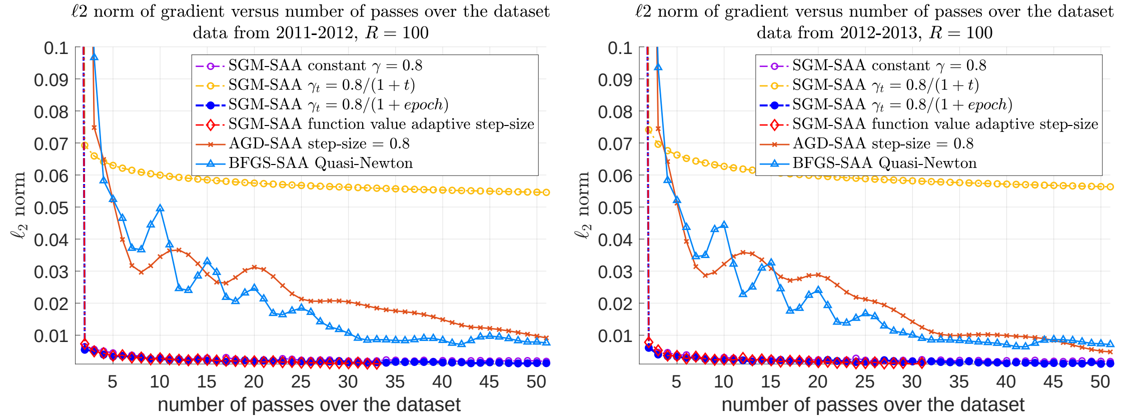


Figure 5.3. ℓ_2 norm of gradient after each pass of the dataset.

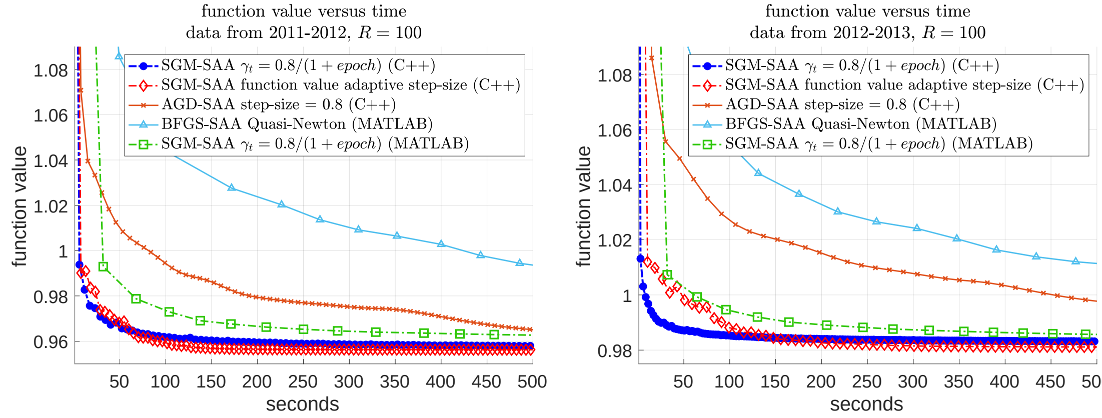


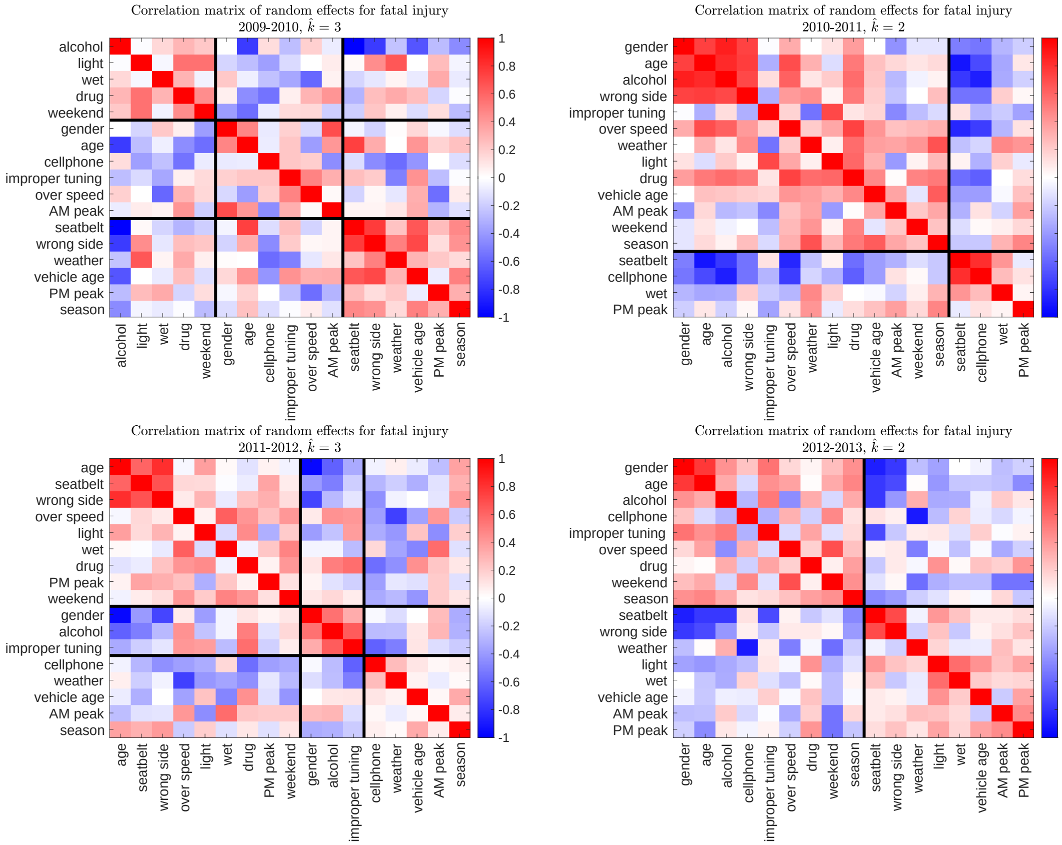
Figure 5.4. Objective values of mixed logit model against time budgets given to the optimization algorithms.

		SGM-SAA adaptive γ_t	SGM-SAA epoch-wise γ_t	AGD-SAA constant γ_t
$R = 100$	$\boldsymbol{\alpha}$ std.	0.9672	0.4209	0.0144
	$\boldsymbol{\mu}$ std.	0.8053	0.4808	0.0088
	$\boldsymbol{\Gamma}$ std.	2.6007	1.6379	0.0436
	LR index mean	0.2921	0.2902	0.2834
	LR index std.	9.4×10^{-4}	6.4×10^{-4}	2.9×10^{-4}
$R = 500$	$\boldsymbol{\alpha}$ std.	0.9295	0.2704	0.0252
	$\boldsymbol{\mu}$ std.	0.4769	0.3204	0.0067
	$\boldsymbol{\Gamma}$ std.	1.6878	1.1187	0.0234
	LR index mean	0.2898	0.2889	0.2815
	LR index std.	8.3×10^{-4}	5.0×10^{-4}	2.6×10^{-4}
$R = 1000$	$\boldsymbol{\alpha}$ std.	0.9418	0.2396	0.0101
	$\boldsymbol{\mu}$ std.	0.4551	0.3156	0.0062
	$\boldsymbol{\Gamma}$ std.	1.4240	1.0321	0.0166
	LR index mean	0.2883	0.2876	0.2813
	LR index std.	8.5×10^{-4}	4.5×10^{-4}	6.4×10^{-5}

Table 5.3. Average standard deviation of individual parameters in the intercept $\boldsymbol{\alpha}$, mean $\boldsymbol{\mu}$, and covariance $\boldsymbol{\Gamma}$, and Likelihood Ratio index (McFadden's pseudo-R^2) achieved after 50 pass of 2012-2013 data. Higher LR index corresponds to better goodness-of-fit.

5.4.3 Correlation Clustering Analysis

In this section, we present analysis based on the fitted covariance matrix of random effects for different accident factors in mixed logit model. The covariance matrix fitted by Stochastic Gradient Method with adaptive step-sizes and Monte Carlo simulation are studied, since this algorithm obtained the best likelihood value among all the tested methods. The covariance matrix $\hat{\boldsymbol{\Sigma}}$ is estimated by the average of 10 independent runs of SGM. For easier interpretation, we transformed the covariance matrix $\hat{\boldsymbol{\Sigma}}$ into correlation matrix $\hat{\boldsymbol{\rho}}$, $\hat{\boldsymbol{\rho}}_{ij} = \hat{\boldsymbol{\Sigma}}_{ij}/\sqrt{\hat{\boldsymbol{\Sigma}}_{ii}\hat{\boldsymbol{\Sigma}}_{jj}}$. We aim to understand the correlation between different accident factors. In most applications of mixed logit model for traffic accident analysis, the random effects are assumed to be independent. However, accident factors are known to be correlated. For example, young males are the most likely to speed for drivers involved in fatal crashes [137]. Therefore, under independence assumption, clustering structure does not exist in the correlation matrix of random effects. Each factor forms a singleton cluster. In comparison, correlation between random effects within each outcome


Correlation matrix of random effects for fatal injury 2009-2010, $\hat{k}=3$

Correlation matrix of random effects for fatal injury 2010-2011, $\hat{k}=2$

Correlation matrix of random effects for fatal injury 2011-2012, $\hat{k}=3$

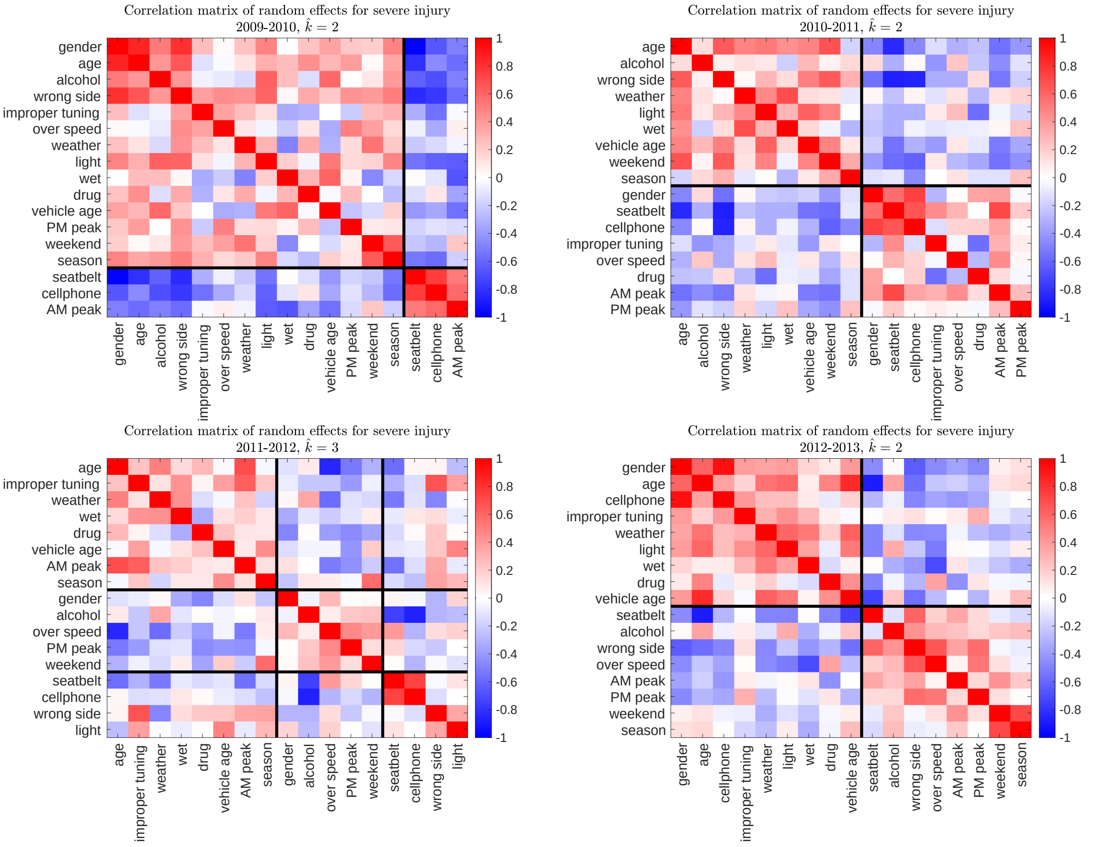
Correlation matrix of random effects for fatal injury 2012-2013, $\hat{k}=2$

Figure 5.5. Correlation matrices in mixed logit model for random effects of accident factors in fatal crashes. Rows and columns are reordered according to the cluster membership output by best spectral clustering results.

category are possible in the model definition (5.3) and (5.4). We analyzed whether there are clustering structure for the random effects of different accident factors emerged from possible correlation. We formulate this objective as a **correlation clustering** problem on p variables.

$$\left(k^*, \{C_i^*\}_{i=1}^k\right) \in \arg\max_{1\le k\le p} \max_{\substack{C_i\cap C_j=\emptyset\\ \cup_{i=1}^k |C_i|=p}} \sum_{l=1}^k \sum_{i,j\in C_l} \hat{\boldsymbol{\rho}}_{ij} - \sum_{\substack{l,r=1\\ l\neq r}}^k \sum_{\substack{i\in C_l,\\ j\in C_r}} \hat{\boldsymbol{\rho}}_{ij}. \tag{5.23}$$

The optimization objective in (5.23) seeks for the partitions of factors, such that positive correlation among factors within each cluster and negative correlation cross different clusters are encouraged. Vice versa, negative correlation within the clusters and positive correlation across different clusters are penalized. Note that both the optimal number of clusters k^* and the partition $\{C_i^*\}_{i=1}^k$ are optimization


Correlation matrix of random effects for severe injury 2009-2010, $\hat{k} = 2$

Correlation matrix of random effects for severe injury 2010-2011, $\hat{k} = 2$

Correlation matrix of random effects for severe injury 2011-2012, $\hat{k} = 3$

Correlation matrix of random effects for severe injury 2012-2013, $\hat{k} = 2$

Figure 5.6. Correlation matrices in mixed logit model for random effects of accident factors in severe injury crashes. Rows and columns are reordered according to the cluster membership output by best spectral clustering results.

variables. We adapt the following definition of clusterability.

Definition 2. *A correlation matrix or correlation graph of p factors is clusterable if the optimal k^* in (5.23) is less than p.*

Problem (5.23) is NP-Hard and computationally infeasible to find the exact solution. Therefore, we apply graph spectral clustering technique for each value of k, and select the best solution among the them. Intuitively, the factors and correlation between them can be represented by a weighted graph, where each node represents a factor and the edges are weighted by the correlation between nodes. Spectral clustering is a popular clustering method on graphs with non-negative edge weights. To apply spectral clustering, we shift the center of the correlation matrix and rescale it. Define the weighted adjacency matrix

$$\boldsymbol{\omega} = \frac{1}{2}(\boldsymbol{\rho} + \mathbf{1}). \tag{5.24}$$

Elements of the resulting matrix $\boldsymbol{\omega}$ is bounded between 0 and 1, but encodes the same information as the original matrix $\boldsymbol{\rho}$. Further, the Laplacian matrix of a graph is defined by

$$L = D - \boldsymbol{\omega}, \tag{5.25}$$

where L is a diagonal matrix and $L_{ii} := \sum_{ij} \boldsymbol{\omega}_{ij}$. The algorithm is briefly described here, please see [138] for additional details. The spectral clustering algorithm requires the users to select the number of clusters k to be found.

- First, the smallest k-th eigenvectors $\{\mathbf{v}_i\}_{i=1}^k$ of the Laplacian matrix L are computed, and let $V \in \mathbb{R}^{p \times k}$ denote the matrix whose columns are the eigenvectors.

- Let V_i be the i-th row of V, V_i is used as a representation of factor i on the graph.

- run k-means clusering algorithm on $V \in \mathbb{R}^{p \times k}$ with k clusters, and output the cluster membership for each factor.

Note that the k-means algorithm used in spectral clustering requires the number k of clusters as input, and the algorithm solves a non-convex optimization problem. We apply spectral clustering for each value of $1 \leq k \leq p$. For each k, the objective value of the original correlation clustering problem (5.23) is computed according to the cluster membership output from spectral clustering. To mitigate the variation introduced by the non-convexity of k-means, we repeat k-means 100 times for each value of k and compute the average score of the objective function (5.23). Finally, the $\hat{k}$ with highest score for (5.23) and the partitioning with $\hat{k}$ clusters is used as an approximate solution. Moreover, spectral clustering always returns p singleton clusters if the k is set to the maximal number of clusters p. Let v^* denote the optimal objective value of (5.23), and v_a be the objective value associated with clustering result a. We have $v_{\text{singleton}} \leq v_{\hat{k}} \leq v^*$. Therefore, according to the Definition 2 for clusterability, the decision of whether the factors are clusterable is unaffected by the spectral clustering approximation once the certificate $v_{\text{singleton}} < v_{\hat{k}}$

is found. The correlation matrix between random effects of accident factors in fatal category and severe injury category are displayed in Figure 5.5 and Figure 5.6 respectively. To assist the visualization, we reorder the rows and columns of each correlation matrix by grouping together factors belonging to the same cluster. For both fatal injury and severe injury, the cluster pattern of the correlation matrix can be immediately informed by visual inspection. The best spectral clustering results are much less than p in all cases. However, the clustering structure is not temporally stable, as seen by the changing correlation clustering pattern across years. Temporal instability in highway accidents data are also studied recently by Mannering. [139]. For fatal injuries, the random effects of gender and age have strong positive correlation when the model is estimated on accidents occurred between 2009 and 2010, between 2010 and 2011, and between 2012 and 2013, but negatively correlated on the 2011-2012 dataset.

5.5 Conclusion

Deterministic gradient-based methods, such as BFGS Quasi-Newton algorithm, are usually used for logit model estimation in transportation research. However, deterministic gradient-based methods suffer scalability issue on large datasets, due to the growing cost of gradient computation proportional to the number of samples in the dataset. In order to perform model estimation on large datasets under realistic computational budget constraints, one has to choose algorithms that offer light weight computation per iteration and optimize the estimators as good as possible within the computational budget limit. Stochastic Gradient Method is a candidate solution meeting the aforementioned requirements. To apply Stochastic Gradient Method on mixed logit estimation where the objective function involves high dimensional integrals, we described a combination of Monte Carlo simulation and Stochastic Gradient Method. The optimization performance of deterministic versus stochastic gradient-based algorithms are evaluated on estimating multinomial logit and mixed logit model with a correlated Gaussian mixing distribution on traffic accident data collected in SWITRS from multiple years. Under the same computational budget, variants of Stochastic Gradient Method are able to reach solutions with better goodness-of-fit on average compared to Accelerated Gradient Descent and Quasi-Newton BFGS method. However, SGM variants suffer higher variance of estimated parameters due to the stochasticity from both the iteration and the simulation of objective functions. Users of SGM for estimating mixed logit model should be aware of this downside. In our case study using a mixed logit model with block-wise correlated Gaussian mixing distribution, we found that the random effects of accident factors can be grouped into clusters and the number of clusters is much less than the number of factors. However the clustering structure may not be stable due to non-convexity of the estimator, stochasticity of the algorithm, and the intrinsic temporal variation of high-way data.

Chapter 6

Convex Latent Effect Logistic Regression via Low-Rank and Sparse Decomposition

The mixed logit model described in Chapter 5 gains its popularity in transportation research due to its flexibility to incorporate individual heterogeneity. In the mixed logit model, the heterogeneity is modeled by randomness in the parameters, i.e., for each observation $n \in \{1, 2, \cdots, N\}$, the model parameter

$$\boldsymbol{\beta}_n \sim f(\boldsymbol{\beta}|\boldsymbol{\Psi}) \tag{6.1}$$

for some mixing distribution f. For example, when f is the Gaussian distribution, $\boldsymbol{\beta}_n \in \mathbb{R}^{pI}$ can be decomposed as

$$\boldsymbol{\beta}_n = \boldsymbol{\mu} + \boldsymbol{\Gamma}\boldsymbol{\eta}_n, \tag{6.2}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{pI}$ and $\boldsymbol{\Gamma} \in \mathbb{R}^{pI \times pI}$ is the mean and the Choleskey factor of covariance matrix respectively. Chapter 5 described an efficient stochastic optimization algorithm based on a combination of Monte Carlo simulation and stochastic gradient to estimate $\boldsymbol{\mu}$ and $\boldsymbol{\Gamma}$. Although the scalability issue for model estimation is addressed, the Maximum Simulated Likelihood objective function is inherently non-convex and leads to unstable estimates. The present chapter aims to develop a convex alternative to mixed logit model, allowing parameter heterogeneity while preserving convexity. We first revisit the regularized multinomial logistic regression with only fixed effects,

$$(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}) \in \operatorname{argmin} \frac{1}{N} \sum_{n=1}^{N} \ell\Big(\mathbf{x_n}, t_n; \boldsymbol{\alpha}, \boldsymbol{\beta}\Big) + \lambda \mathcal{R}(\boldsymbol{\beta}), \tag{6.3}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^I, \boldsymbol{\beta} \in \mathbb{R}^{pI}$, $\mathcal{R}(\boldsymbol{\beta})$ is a penalty term with non-negative regularization constant λ, ℓ is the multinomial loss function

$$\ell\Big(\mathbf{x_n}, t_n; \boldsymbol{\alpha}, \boldsymbol{\beta}\Big) = -\log\left(\frac{\exp\left(\alpha^{(t_n)} + {\boldsymbol{\beta}^{(t_n)}}^T \mathbf{x}_n\right)}{\sum_{j=1}^{I} \exp\left(\alpha^{(j)} + {\boldsymbol{\beta}^{(j)}}^T \mathbf{x}_n\right)}\right). \tag{6.4}$$

The penalty $\mathcal{R}(\boldsymbol{\beta})$ is useful for incorporating prior knowledge into the estimation process, and restricting the model complexity to aid interpretation and prevent overfitting. The degrees of freedom in the model can be limited by the support constraint, by solving

$$\min \frac{1}{N} \sum_{n=1}^{N} \ell\Big(\mathbf{x_n}, t_n; \boldsymbol{\alpha}, \boldsymbol{\beta}\Big) \qquad \text{subject to} \qquad \sum_{i=1}^{pI} \mathbb{I}(\beta_i \neq 0) \tag{6.5}$$

where $\mathbb{I}$ is the indicator function. However, optimization with cardinality constraint in (6.5) becomes a NP-hard discrete problem. The ℓ_1-norm penalty $\|\boldsymbol{\beta}\|_1$ is a well-known convex surrogate to the cardinality function. In the Lagrangian form, the ℓ_1-regularized logit model is

$$(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}) \in \operatorname{argmin} \frac{1}{N} \sum_{n=1}^{N} \ell\Big(\mathbf{x_n}, t_n; \boldsymbol{\alpha}, \boldsymbol{\beta}\Big) + \lambda \|\boldsymbol{\beta}\|_1 . \tag{6.6}$$

When the penalty constant λ is small, the optimal solutions to problem (6.6) occur at the vertices of the ℓ_1 ball, and force some components of $\hat{\boldsymbol{\beta}}$ into zeros. Therefore, problem (6.6) produces sparse solutions. Although the solution from the ℓ_1-penalized problem is sparse, it is possible that $\hat{\boldsymbol{\beta}}_i^{(j)} = 0$ and $\hat{\boldsymbol{\beta}}_i^{(k)} \neq 0$ for some $j \neq k$. In these cases, variable x_i affects the linear predictor function for category j but not for category k. Interpreting sparsity as variable selection is often desired, such that $\hat{\boldsymbol{\beta}}_i^{(j)} = 0$ for all $j \in \{1, 2, \cdots, I\}$ if x_i is dropped from the model. Let

$$\mathbf{B} := [\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, \cdots, \boldsymbol{\beta}^{(I)}] \in \mathbb{R}^{p \times I}$$

be a matrix appending $\boldsymbol{\beta}^{(j)}$s as columns, and let $\{\mathbf{b}_i\}_{i=1}^{p} \in \mathbb{R}^I$ be the rows in $\mathbf{B}$. The *group* ℓ_1 regularized problem

$$(\hat{\boldsymbol{\alpha}}, \hat{\mathbf{B}}) \in \operatorname{argmin} \frac{1}{N} \sum_{n=1}^{N} \ell\Big(\mathbf{x_n}, t_n; \boldsymbol{\alpha}, \mathbf{B}\Big) + \lambda \sum_{i=1}^{p} \|\mathbf{B}_i\|_2 \tag{6.7}$$

promotes the entire rows $\mathbf{B}_i$ to be zero vectors, and yields *group sparsity.* Thus $\mathbf{B}_i = \mathbf{0}$ implies variable i is not selected in the model. Since the multinomial logistic loss function (6.4) is convex, problem (6.3) is convex when convex penalties $\mathcal{R}(\boldsymbol{\beta})$ are used, as in the case of (6.6) and (6.7). So far our discussion focuses on regularized fixed effect models. The parameters $\{\boldsymbol{\beta}^{(j)}\}_{j=1}^{I}$ are shared across all individuals. Compared to the mixed logit model (5.9), problem (6.6) and (6.7) have the advantages of being convex with efficient algorithms to solve them, and can be interpreted as built-in variable selection methods. In this chapter, our goal is to develop a convex formulation with possible variable selection, and consider individual heterogeneity at the same time. A naive approach is to parameterize each observation separately, fitting heterogeneous $\{\boldsymbol{\beta}_n\}_{n=1}^{N}$ for each individual:

$$\min \frac{1}{N} \sum_{n=1}^{N} -\log\left(\frac{\exp\left(\alpha^{(t_n)} + {\boldsymbol{\beta}_n^{(t_n)}}^T \mathbf{x}_n\right)}{\sum_{j=1}^{I} \exp\left(\alpha^{(j)} + {\boldsymbol{\beta}_n^{(j)}}^T \mathbf{x}_n\right)}\right). \tag{6.8}$$

(6.8) is a saturated model with $N \times p \times I + I$ free parameters on N observations. The over-parametrization can overfit the data with little statistical power.

6.1 Low-rank and sparse decomposition

To avoid over-parametrization and extract the common effect from the observations, we propose the following modification

$$\boldsymbol{\beta}_n = \boldsymbol{\mu} + \boldsymbol{v}_n. \tag{6.9}$$

We separate the individual parameter $\boldsymbol{\beta}_n$ into $\boldsymbol{\mu}$ for the homogeneous effect and $\boldsymbol{v}_n$ for the heterogeneous effect of observation n. Without other constraints, the representation in (6.9) fits $(N+1) \times p \times I$ parameters on N data points and identifying the decomposition from data is in general impossible without other conditions. Therefore, we further impose that

$$\begin{aligned} &\sum_{i=1}^{p} \|\mathbf{U}_i\|_2 \le \tau_1 \\ &\operatorname{rank}(\boldsymbol{\Upsilon}) \le \tau_2 \end{aligned} \tag{6.10}$$

where

$$\boldsymbol{\Upsilon} := [\boldsymbol{\upsilon}_1, \boldsymbol{\upsilon}_2, \cdots, \boldsymbol{\upsilon}_N] \in \mathbb{R}^{pI \times N}$$

is a matrix with the heterogeneous parameters $\boldsymbol{\upsilon}_n$ as columns,

$$\mathbf{U} = [\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \cdots, \boldsymbol{\mu}^{(I)}] \in \mathbb{R}^{p \times I}$$

is a matrix collecting the common effects, and the i-th row of $\mathbf{U}$ is denoted by $\mathbf{U}_i \in \mathbb{R}^I$. The following problem is considered.

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \mathbf{U}, \boldsymbol{\Upsilon}} \ & \frac{1}{N} \sum_{n=1}^{N} -\log \left(\frac{\exp\left(\alpha^{(t_n)} + \boldsymbol{\mu}^{(t_n)^T} \mathbf{x}_n + \boldsymbol{\upsilon}_n^{(t_n)^T} \mathbf{x}_n\right)}{\sum_{j=1}^{I} \exp\left(\alpha^{(j)} + \boldsymbol{\mu}^{(j)^T} \mathbf{x}_n + \boldsymbol{\upsilon}_n^{(j)^T} \mathbf{x}_n\right)} \right) \\ \text{subject to } & \sum_{i=1}^{p} \|\mathbf{U}_i\|_2 \le \tau_1 \\ & \operatorname{rank}(\boldsymbol{\Upsilon}) \le \tau_2. \end{aligned} \tag{6.11}$$

In (6.11), the group ℓ_1 norm produces group sparse $\mathbf{U}$ and effectively reduces the degrees of freedom of the homogeneous effect. When $\tau_2 \ll \min(pI, N)$, the heterogeneity effects $[\boldsymbol{\upsilon}_n]_{n=1}^N$ are constrained in a low-rank space. In the next section, we explain why imposing low-rank structural assumptions helps uncovering the separation between homogeneous and heterogeneous effect.

6.2 Why Low-Rankness?

In this section, we explain situations where low-rank heterogeneous effect arises.

6.2.1 Gaussian mixing with low-rank covariance

The homogeneous-heterogeneous effect decomposition in (6.9) resembles the linear representation for Gaussian mixing variables in (6.2), where the mean of the

Gaussian distributions corresponds to the homogeneous effect, and

$$\boldsymbol{\Upsilon} = \boldsymbol{\Gamma}\mathbf{H} \tag{6.12}$$

where $\mathbf{H} := [\boldsymbol{\eta}_1, \boldsymbol{\eta}_1, \cdots, \boldsymbol{\eta}_N] \in \mathbb{R}^{pI \times N}, \boldsymbol{\eta}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Low-rank mixing effect $\boldsymbol{\Upsilon}$ occurs when the covariance matrix is (approximately) low-rank. This happens when the covariance matrix is a superposition of a low-rank component and a sparse components [140–144], or when the covariance matrix has a block diagonal structure where the variables within each block are highly correlated [145, 146].

6.2.2 Latent clustered heterogeneity

Suppose the population forms clusters, such the accident factors have identical or similar influence on individuals belonging to the same cluster. Let $\{\mathcal{C}_l\}_{p=1}^{\tau}$ be a partition of the data, i.e., $\mathcal{C}_l \cap \mathcal{C} = \emptyset$ if $l \neq l'$, $\cup_{l=1}^{\tau} |\mathcal{C}_l| = N$. If the individual heterogeneities are identical on each cluster, then

$$\boldsymbol{v}_i = \boldsymbol{v}_j \quad \forall\ i, j \in C_l,\ l = 1, \cdots, \tau. \tag{6.13}$$

Therefore, observations in the same cluster have identical columns in $\boldsymbol{\Upsilon}$. As a result, $\text{rank}(\boldsymbol{\Upsilon}) \leq \tau$. (6.13) can be relaxed slightly, consider

$$\|\boldsymbol{v}_i - \boldsymbol{v}_j\|_{\ell_p} \leq \epsilon \quad \forall\ i, j \in C_l,\ l = 1, \cdots, \tau. \tag{6.14}$$

(6.14) leads to grouping of the heterogeneous effect, and forces the latent effects in the same group are stays close. $\boldsymbol{\Upsilon}$ can be approximated by a low-rank matrix in this case.

6.2.3 Latent matrix factorization

Suppose $\text{rank}(\boldsymbol{\Upsilon}) = \tau$ with $\tau \leq \min(pI, N)$. $\boldsymbol{\Upsilon}$ can be equivalently expressed as

$$\boldsymbol{\Upsilon} = \mathbf{W}\mathbf{V}^T, \tag{6.15}$$

where $\mathbf{W} \in \mathbb{R}^{pI \times \tau}$, $\mathbf{V} \in \mathbb{R}^{N \times \tau}$ and both $\mathbf{W}$ and $\mathbf{V}$ has full column-rank. The matrix $\mathbf{V}$ is usually interpreted as latent loadings, and $\mathbf{W}$ are scores for each of the basis in $\mathbf{V}$. This is sometimes referred as a matrix factorization representation [147–152]. Under this interpretation, individual heterogeneities are created through different linear combinations of the same latent sources.

6.3 Convex Optimization

6.3.1 Convex relaxation

Optimization with matrix rank constraints is NP-Hard. Finding the exact solutions to problem (6.11) is difficult and can be unpractical. Therefore, we use a convex program to approximate the original formulation. Let $\|A\|_*$ be the *nuclear norm* for matrix A define as following.

$$\|A\|_* = \sum_{i=1}^{\tau} \sigma_i, \tag{6.16}$$

where τ is the rank of A and σ_i is the i-th largest singular value of A. The nuclear norm $\|A\|_*$ is a convex relaxation for the rank function $\text{rank}(A)$ [153–155] and has been successfully applied on many problems involving the rank function as objective or constraints [38–42, 140–144] (and references therein). Therefore, we solve the following convex program

$$\min_{\boldsymbol{\alpha},\mathbf{U},\boldsymbol{\Upsilon}} \frac{1}{N}\sum_{n=1}^{N} -\log\left(\frac{\exp\left(\alpha^{(t_n)} + \boldsymbol{\mu}^{(t_n)T}\mathbf{x}_n + \boldsymbol{\upsilon}_n^{(t_n)T}\mathbf{x}_n\right)}{\sum_{j=1}^{I}\exp\left(\alpha^{(j)} + \boldsymbol{\mu}^{(j)T}\mathbf{x}_n + \boldsymbol{\upsilon}_n^{(j)T}\mathbf{x}_n\right)}\right) + \lambda_1\sum_{i=1}^{p}\|\mathbf{U}_i\|_2 + \lambda_2\|\boldsymbol{\Upsilon}\|_*. \tag{6.17}$$

(6.17) reformulates (6.11) by first transforming the original problem into the Lagrangian form, then applying the nuclear norm relaxation to $\text{rank}(\boldsymbol{\Upsilon})$. (6.17) is the addition of three convex functions and therefore the convexity is preserved. $\lambda_1, \lambda_2 \in \mathbb{R}_+$ are non-negative hyperparameters correspond to the upper bound τ_1 and τ_2 in the hard-constraint version.

6.3.2 Proximal Gradient algorithm

Algorithm to solve (6.17) is described in this section. The main steps in the algorithm are to deal with the non-smooth terms $\sum_{i=1}^{p}\|\mathbf{U}_i\|_2$ and $\|\boldsymbol{\Upsilon}\|_*$. Problem

(6.17) has the form

$$\min_{\boldsymbol{\theta}}\{F(\boldsymbol{\theta}) \equiv f(\boldsymbol{\theta}) + h(\boldsymbol{\theta})\},$$

where f is a smooth convex part corresponds to the multinomial loss, h is convex but non-smooth part consists of $\Big(\lambda_1 \sum_{i=1}^{p} \|\mathbf{U}_i\|_2 + \lambda_2 \|\boldsymbol{\Upsilon}\|_*\Big)$. Proximal Gradient method (PG) [43–47] is an efficient first-order algorithm to handle the non-smoothness in h. It performs the following simple iteration

$$\boldsymbol{\theta}_{t+1} = \mathrm{Prox}_{s_t,h}\Big(\boldsymbol{\theta}_t - s_t \nabla f(\boldsymbol{\theta}_t)\Big) \tag{6.18}$$

where s_t is the step-size and $\mathrm{Prox}(\cdot)$ is the proximal operator associated with s_t and $h(\cdot)$ defined below.

$$\mathrm{Prox}_{s_t,h}(\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\omega}} \left\{\frac{1}{2}\|\boldsymbol{\omega} - \boldsymbol{\theta}\|_2^2 + s_t h(\boldsymbol{\theta})\right\}. \tag{6.19}$$

The Proximal Gradient method closely resembles the iterations in Gradient Descent, especially when the proximal operator admits close-form solutions. Let $(\boldsymbol{\alpha}_t, \mathbf{U}_t, \boldsymbol{\Upsilon}_t)$ denote the iterates after the t-th PG iteration (6.18). Applying (6.18) on (6.17) leads to the following update steps using intermediate variables $(\widehat{\boldsymbol{\alpha}}_t, \widehat{\mathbf{U}}_t, \widehat{\boldsymbol{\Upsilon}}_t)$

$$(\widehat{\boldsymbol{\alpha}}_t, \widehat{\mathbf{U}}_t, \widehat{\boldsymbol{\Upsilon}}_t) = (\boldsymbol{\alpha}_t, \mathbf{U}_t, \boldsymbol{\Upsilon}_t) - \frac{s_t}{N}\sum_{n=1}^{N} \nabla \ell\Big(\mathbf{x}_n, t_n; \boldsymbol{\alpha}_t, \mathbf{U}_t, \boldsymbol{\Upsilon}_t\Big) \tag{6.20}$$

$$\boldsymbol{\alpha}_{t+1} = \widehat{\boldsymbol{\alpha}}_t \tag{6.21}$$

$$(\mathbf{U}_{t+1}, \boldsymbol{\Upsilon}_{t+1}) = \operatorname*{argmin}_{\mathbf{U},\boldsymbol{\Upsilon}} \left\{\frac{1}{2}\Big(\Big\|\widehat{\mathbf{U}}_t - \mathbf{U}\Big\|_{\mathrm{F}}^2 + \Big\|\widehat{\boldsymbol{\Upsilon}}_t - \boldsymbol{\Upsilon}\Big\|_{\mathrm{F}}^2\Big) + s_t\Big(\lambda_1 \sum_{i=1}^{p} \|\mathbf{U}_i\|_2 + \lambda_2 \|\boldsymbol{\Upsilon}\|_*\Big)\right\} \tag{6.22}$$

The sub-problem from the proximal operator (6.22) can be decoupled into two separate optimization problem without changing the results.

$$\mathbf{U}_{t+1} = \operatorname*{argmin}_{\mathbf{U}} \left\{\frac{1}{2}\Big\|\widehat{\mathbf{U}}_t - \mathbf{U}\Big\|_{\mathrm{F}}^2 + s_t\lambda_1 \sum_{i=1}^{p} \|\mathbf{U}_i\|_2\right\} \tag{6.23a}$$

$$\boldsymbol{\Upsilon}_{t+1} = \operatorname*{argmin}_{\boldsymbol{\Upsilon}} \left\{\frac{1}{2}\Big\|\widehat{\boldsymbol{\Upsilon}}_t - \boldsymbol{\Upsilon}\Big\|_{\mathrm{F}}^2 + s_t\lambda_2 \|\boldsymbol{\Upsilon}\|_*\right\} \tag{6.23b}$$

Moreover, both (6.23a) and (6.23b) has closed-form solutions. The optimal solution to (6.23a) can be obtained by solving the proximal operator independently over each row $\{\mathbf{U}_i\}_{i=1}^{p}$. The sub-problems with respect to $\mathbf{U}_i$ are essentially the proximal operation arisen from the group-Lasso penalty [47, 156]. For each $i = 1, 2, \cdots, p$,

$$\operatorname{Prox}_{s_t, \lambda_1 \|\cdot\|_2}\Big(\mathbf{U}_i\Big) = \Big(1 - \frac{s_t \lambda_1}{\|\mathbf{U}_i\|}\Big)_+ \mathbf{U}_i, \tag{6.24}$$

where the operator $(\cdot)_+$ denotes $a_+ = \max(0, a)$. The closed-form solution for the proximal operator associated with nuclear norm penalty $\lambda_2 \|\cdot\|_*$ and step-size s_t can be obtained via a generalization of the soft-thresholding procedure in (6.24). Suppose the Singular Value Decomposition (SVD) for $\boldsymbol{\Upsilon}$ is

$$\boldsymbol{\Upsilon} = \mathbf{P}\boldsymbol{\Sigma}\mathbf{V}^T, \tag{6.25}$$

where $\boldsymbol{\Sigma} = \operatorname{diag}(\{\sigma_i\}_{i=1}^{\tau})$ is the diagonal matrix of singular values of $\boldsymbol{\Upsilon}$ and $\operatorname{rank}(\boldsymbol{\Upsilon}) = \tau$. The closed-form solution to (6.23b) is given by the Singular-Value Thresholding operator [44–46],

$$\operatorname{Prox}_{s_t, \lambda_2 \|\cdot\|_*}(\boldsymbol{\Upsilon}) = \mathbf{P}\mathcal{T}_{s_t\lambda_2}(\boldsymbol{\Sigma})\mathbf{V}^T, \quad \mathcal{T}_{s_t\lambda_2}(\boldsymbol{\Sigma}) = \operatorname{diag}\{(\sigma_i - s_t\lambda_2)_+\}_{i=1}^{\tau}. \tag{6.26}$$

(6.24) and (6.26) together provide the solutions to (6.22).

6.3.3 Acceleration and practical implementation

We apply several simple techniques to improve the convergence speed of Proximal Gradient algorithm described in the previous section. Nesterov's momentum method is a well-known technique to derive optimal convergence rate for first-order optimization methods on smooth convex problems [129]. The extension to accelerate the convergence on non-smooth objectives via proximal operators are developed by Beck and Teboulle [43], known as the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA). The acceleration technique is applied in our implementation.

To ensure, the convergence of Proximal Gradient and FISTA, it is required that the smooth part $f(\boldsymbol{\theta})$ has Lipschitz continuous gradient, i.e.,

$$\|\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\| \leq L \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|$$

for all $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \operatorname{dom}(f)$. The Proximal Gradient algorithm has $O\Big(\frac{1}{t}\Big)$ convergence rate and FISTA achieves improved rate of $O\left(\frac{1}{t^2}\right)$. The convergence can be established by using a fixed step-size $s_t = 1/L$, when the Lipschitz constant L is known. In practice, the Lipschitz constant might be unknown or hard to compute. Backtracking line search is a common strategy to decide the step-size s_t when the Lipschitz constant is unknown. Backtracking line search requires computing the proximal operator over a range of different step-sizes. Since the intermediate variable $\widehat{\mathbf{\Upsilon}}_t$ changes due to different trial step-size s_t during the line search process, the Singular Value Decomposition of $\widehat{\mathbf{\Upsilon}}_t$ needs to be re-computed in each trial step in order to obtain $\mathrm{Prox}_{s_t, \lambda_2 \|\cdot\|_*}(\widehat{\mathbf{\Upsilon}}_t)$. Therefore backtracking line search becomes costly due to multiple computations of SVD in each iteration. Hence we use an adaptive step-size scheme as a surrogate to line search. After each iteration, the objective value of (6.17) is measured, the step-size is halved if the objective value increases compared to the previous iteration. Note that Accelerated Proximal Gradient (APG) is not a strictly descent method, as oppose to Gradient Descent, therefore temporally increments of the function value do occur. This phenomenon is referred as the Nesterov ripples in the literature [129, 157]. Further, we employ a function value-based restarting criterion introduced in [157]. The algorithm is restarted with the current iterates as re-initialization when ripples are detected, and the step-size is halved. The acceleration momentums are reset at each restart. The Fast Accelerated Proximal Gradient with Adaptive Restart (FAPGAR) algorithm combining these features is listed in Algorithm 7. In addition, to scale SVD on large datasets, FAPGAR uses the randomized SVD algorithm [158] to compute the Singular-Value Thresholding operator.

6.4 Greedy Local Continuation for Pathwise Solutions

Algorithm 7 solves one instance of problem (6.17), given λ_1 and λ_2. Chapter 4 introduced a gradient-based method for hyperparameter optimization, viewing hyperparameter tuning as a bi-level optimization problem. However, the objective function in (6.17) is non-smooth, therefore it is difficult to compute the hyperparameter gradient. Hence, we solve (6.17) over a range of λ_1 and λ_2, then choose the regularization constants by some predictability criteria. Suppose the search space in each λ is discretized into c points. Grid search requires calling Algorithm 7 c^2 times in order to obtain the solution over the two dimensional grids. This can be

Algorithm 7 Fast Accelerated Proximal Gradient with Adaptive Restart

Input: number of iterations T, initial step-size s_0, initialization $\boldsymbol{\alpha}_0, \mathbf{U}_0, \boldsymbol{\Upsilon}_0$, tolerance ϵ_{tol}, tuning parameters $\lambda_1, \lambda_2 \in \mathbb{R}_+$

Output: solution to (6.17).

```
$q_1 = 1$
$\widetilde{\boldsymbol{\alpha}}_1 = \boldsymbol{\alpha}_0$
$\widetilde{\mathbf{U}}_1 = \mathbf{U}_0$
$\widetilde{\boldsymbol{\Upsilon}}_1 = \boldsymbol{\Upsilon}_0$
for $t = 1$ to $T$ do
    $(\boldsymbol{\alpha}_t, \widehat{\mathbf{U}}_t, \widehat{\boldsymbol{\Upsilon}}_t) = (\boldsymbol{\alpha}_t, \mathbf{U}_t, \boldsymbol{\Upsilon}_t) - \frac{s_t}{N}\sum_{n=1}^{N} \nabla\ell\Big(\mathbf{x}_n, t_n; \widetilde{\boldsymbol{\alpha}}_t, \widetilde{\mathbf{U}}_t, \widetilde{\boldsymbol{\Upsilon}}_t\Big)$
    for $i = 1$ to $p$ do                          ▷ proximal operator for group $\ell_1$ penalty
        $(\mathbf{U}_i)_t = \Big(1 - \frac{s_t\lambda_1}{\|(\widehat{\mathbf{U}}_i)_t\|}\Big)_+ (\widehat{\mathbf{U}}_i)_t$
    end for
    $\boldsymbol{\Upsilon}_t = \texttt{RandomizedSVT}(\widehat{\mathbf{U}}_t, s_t\lambda_2)$       ▷ proximal operator for nuclear norm
    if $F(\boldsymbol{\alpha}_t, \mathbf{U}_t, \boldsymbol{\Upsilon}_t) > F(\boldsymbol{\alpha}_{t-1}, \mathbf{U}_{t-1}, \boldsymbol{\Upsilon}_{t-1})$ then
        $q_t = 1$                                  ▷ restart
        $\widetilde{\boldsymbol{\alpha}}_{t+1} = \boldsymbol{\alpha}_t$
        $\widetilde{\mathbf{U}}_{t+1} = \mathbf{U}_t$
        $\widetilde{\boldsymbol{\Upsilon}}_{t+1} = \boldsymbol{\Upsilon}_t$
        $s_{t+1} = s_t/2$                          ▷ adjust step-size
    else
        $q_{t+1} = \frac{1+\sqrt{1+4q_t^2}}{2}$    ▷ acceleration
        $\widetilde{\boldsymbol{\alpha}}_{t+1} = \boldsymbol{\alpha}_t + \Big(\frac{q_t - 1}{q_{t+1}}\Big)\Big(\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}\Big)$
        $\widetilde{\mathbf{U}}_{t+1} = \mathbf{U}_t + \Big(\frac{q_t - 1}{q_{t+1}}\Big)\Big(\mathbf{U}_t - \mathbf{U}_{t-1}\Big)$
        $\widetilde{\boldsymbol{\Upsilon}}_{t+1} = \boldsymbol{\Upsilon}_t + \Big(\frac{q_t - 1}{q_{t+1}}\Big)\Big(\boldsymbol{\Upsilon}_t - \boldsymbol{\Upsilon}_{t-1}\Big)$
        $s_{t+1} = s_t$
    end if
    if $\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}\|_2 + \|\mathbf{U}_t - \mathbf{U}_{t-1}\|_{\mathrm{F}} + \|\boldsymbol{\Upsilon}_t - \boldsymbol{\Upsilon}_{t-1}\|_{\mathrm{F}} < \epsilon_{tol}$ then
        break
    end if
end for
```

prohibitive. In this section, we describe a fast strategy via greedy local search and continuation method to establish the solution over a range of tuning constants.

6.4.1 Prediction for new observations

Given a new unseen sample $\mathbf{x}_n \in \mathbb{R}^p$, the latent heterogeneous effect $\boldsymbol{\upsilon}_n$ needs to be decided. As show in section 6.2, clustering is one of the situations causing low-rank latent effect. Therefore, we query the k-nearest neighbors of $\mathbf{x}_n$, where the distance is calculated by one minus the cosine similarity $w_{ij} = \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$. The heterogeneous effect for the new test sample is set to the weighted average of heterogeneous effect of observations belonged to the neighborhood $\mathcal{N}$ of the new test sample, i.e., $\boldsymbol{v}_n = \sum_{j \in \mathcal{N}} w_j \boldsymbol{v}_j / \sum_{j \in \mathcal{N}} w_j$.

6.4.2 Greedy local continuation

For Lasso or other ℓ_1 type penalized fixed effect generalized linear models, Friedman et al. [159, 160]. showed that $\|\beta(\lambda)\|_1$ forms a continuous path of the regularization constant λ. Therefore, the model parameters stay close when change in the penalty constant is small enough. Based on this observation, Friedman et al. proposed warm start strategy to compute the solution over the entire path of regularization hyperparameters, where the optimal solution from a neighboring λ is used as initialization [159, 160]. This idea is extended to the nuclear norm penalized problem in [46].

This warm start continuation strategy can also be applied on top of FAPGAR (Algorithm 7), when either λ_1 or λ_2 is fixed. Although it is possible to produce the solution path over the entire two dimensional grids generated by the Cartesian product $\lambda_1 \bigotimes \lambda_2$ with warm-starting strategy, the computation cost becomes large since $\text{card}\,(\lambda_1) \times \text{card}\,(\lambda_2)$ calls of Algorithm 7 is required. Here $\text{card}\,(\lambda)$ is the cardinality of grids in λ. Hence, we develop a greedy strategy with continuation method to avoid the computation over the entire search space. We propose a coordinate-wise search strategy. λ_1 and λ_2 is optimized alternatively using the continuation scheme while the other is being fixed. After a solution path over λ_1 is computed for a fixed λ_2, we pick the value of λ_1 yielding the best prediction on a separate validation set. After λ_1 is selected, the solution path for λ_2 is computed with the warm-start continuation strategy. The process continuous and search for λ_1 and λ_2 alternatively. Let $(\lambda_1^{(t)}, \lambda_2^{(t)})$ denote the constants selected after t coordinate-wise outer iterations. A cycle is a configuration $(\lambda_1^{(t)}, \lambda_2^{(t)})$ such that $(\lambda_1^{(t)}, \lambda_2^{(t)}) = (\lambda_1^{(t')}, \lambda_2^{(t')})$ for some $t' < t$. The search process stops when a cycle is detected. Figure 6.1 provides an illustration of the Greedy Local Continuation search.

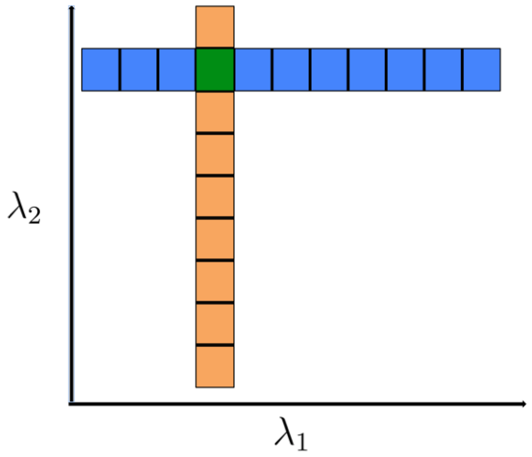


Figure 6.1. Graphical illustration of Greedy Local Continuation search. λ_1 and λ_2 are optimized alternatively with warm-starting strategy.

6.5 Experiments

Experimental evaluations on the proposed low-rank and sparse decomposition-based latent effect model (6.17) and the solution algorithm FAPGAR are described in this section.

6.5.1 Computation efficiency

Computational efficiency of Algorithm 7 is examined. The running time of Algorithm 7 is measured on the traffic accident dataset described in section 5.4.1. We implement Algorithm 7 in MATLAB and perform the experiments on a MacOS system with 2.4 GHz Intel Core i5 processor and 4 GB 1600 MHz DDR3 memory. Data from 2012-2013 are sub-sampled to create training sets with different sizes (N) in order to study the scalability of Algorithm 7. The average running time per iteration are shown in Figure 6.2. The computation time per iteration in FAPGAR scales linearly with the number of samples in the dataset. The FAPGAR algorithm is compared with the Proximal Gradient method with constant step-sizes. The objective value of (6.17) after each iteration is shown in Figure 6.3. Figure 6.3 clearly demonstrates the advantage of employing the adaptive acceleration techniques.

To start the Local Greedy Continuation search for λ_1 and λ_2, we fist run group-ℓ_1

regularized logistic regression over a range of λ_1s using software GLMNET [159,160]. Note that the result of group-ℓ_1 regularized logistic regression is equivalent to our proposed model (6.17) with λ_2 set to a large value, which yields a matrix of zeros for the latent heterogeneity $\boldsymbol{\Upsilon}$. For each λ_1 on the solution path of group-ℓ_1 regularized logistic regression, we classify the samples on a separate validation set by choosing the class with largest estimated probability as the output category for each validation sample. For each hyperparameter configuration, the **F-1 score** is computed:

$$\text{F-1 score} \overset{\text{def}}{=} \Big(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2}\Big)^{-1}. \tag{6.27}$$

The λ_1 value on the group-ℓ_1 regularized logistic regression path with highest F-1 score is taken as initialization for the Local Greedy Continuation search. The progress of the greedy search process is visualized in Figure 6.4. The search process terminated after three iterations due to the occurrence of a cycle. The selected $\lambda_1 = 0.0028, \lambda_2 = 0.01$.

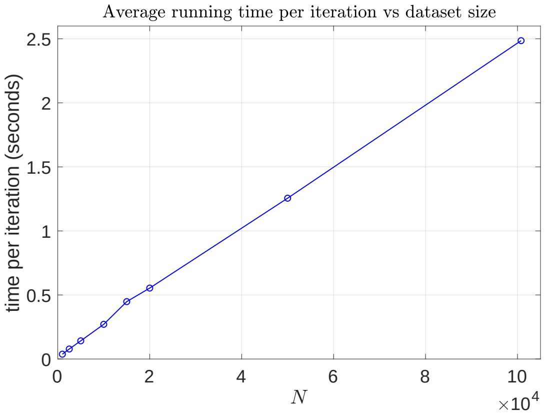


Figure 6.2. Time per iteration (seconds) of FAPGAR implemented in MATLAB.

6.5.2 Accident factor analysis

The latent effect model (6.17) is applied for analyzing traffic accident data in this section. The analysis is performed on $N = 10000$ randomly sampled accidents from the 2012-2013 dataset described in section 5.4.1. The average direct pseudo-elasticity is a metrics for quantifying the influence of a factor [97–99, 106, 161–166] (and references therein). This metric is frequently utilized in transportation research. Given a binary feature vector $\mathbf{x}$, the average direct pseudo-elasticity measures the

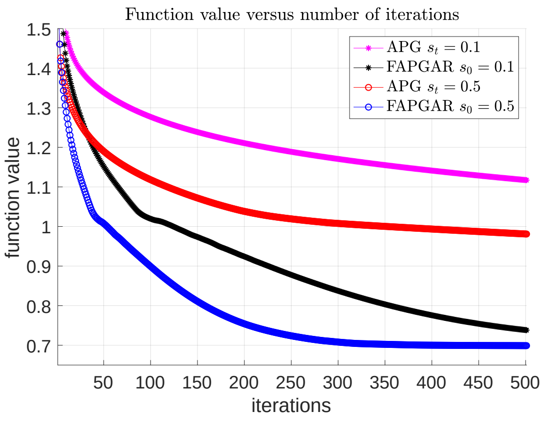


Figure 6.3. Objective value after each iteration, FAPGAR vs Proximal Gradient.

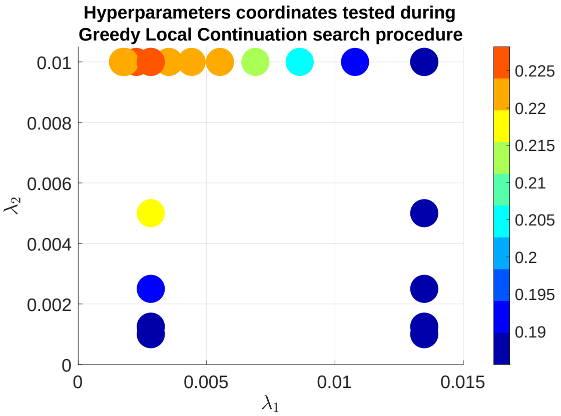
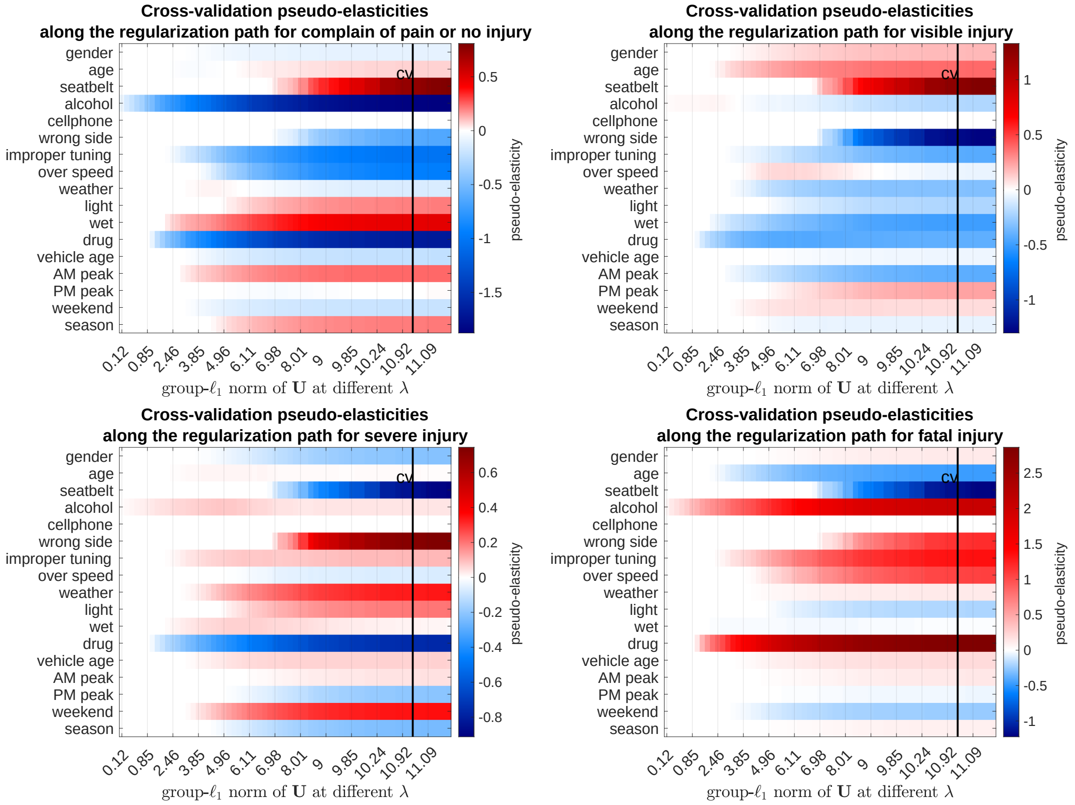


Figure 6.4. Greedy Local Continuation search on the traffic accident dataset, 2012-2013. A cycle is detected after three iterations.

change in the probability of suffering class k injury outcome, when a feature x_i switches from zero to one. The direct pseudo-elasticity for observation n on class k due to the change of x_i can be computed by

$$\mathcal{E}_{ni}^{(k)} \overset{\text{def}}{=} \frac{P(t_n = k|\hat{\boldsymbol{\theta}}, \mathbf{x}_{n\backslash i}, x_i = 1) - P(t_n = k|\hat{\boldsymbol{\theta}}, \mathbf{x}_{n\backslash i}, x_i = 0)}{P(t_n = k|\hat{\boldsymbol{\theta}}, \mathbf{x}_{n\backslash i}, x_i = 0)}, \tag{6.28}$$

where $\hat{\boldsymbol{\theta}}$ denotes the estimated parameters for a model, $\mathbf{x}_{n\backslash i} \in \mathbb{R}^{p-1}$ is a sub-vector of features from $\mathbf{x}_n \in \mathbb{R}^p$ excluding the i-th variable. In transportation literature, the average of $\mathcal{E}_{ni}^{(k)}$ from the training set is calculated. Note that since the parameter $\hat{\boldsymbol{\theta}}$ is fitted from the training set, $\mathcal{E}_{ni}^{(k)}$ is in fact conditioned on the training set. The purpose of computing the average of $\mathcal{E}_{ni}^{(k)}$ is to achieve better estimation of the


Cross-validation pseudo-elasticities along the regularization path for complain of pain or no injury

Cross-validation pseudo-elasticities along the regularization path for visible injury

Cross-validation pseudo-elasticities along the regularization path for severe injury

Cross-validation pseudo-elasticities along the regularization path for fatal injury

Figure 6.5. Cross-validation direct pseudo-elasticity over a path of λ_1. Vertical line marked with 'CV' correspond to the λ_1 with best F-1 score from cross-validation.

direct pseudo-elasticity in the population, i.e.,

$$\mathbb{E}_{n\sim\mathcal{D}}\Big(\mathbb{E}_{\mathcal{S}\sim\mathcal{D}}[\mathcal{E}_{ni}^{(k)}|\mathcal{S}]\Big), \tag{6.29}$$

where $\mathcal{D}$ denotes the population, and $\mathcal{S}$ denotes the training data randomly sampled from $\mathcal{D}$. Therefore, we propose an extension to the average pseudo-elasticity by incorporating the cross-validation procedure:

1. Shuffle the dataset randomly and separate it into s folds. Let $\mathcal{S}_{\setminus i}$ denote the subset of data with the i-th fold removed, and $\mathcal{S}_i$ be the remaining data.

2. estimate a model on $\mathcal{S}_{\setminus i}$, and compute the average pseudo-elasticity on $\mathcal{S}_i$.

3. compute the mean of average pseudo-elasticity estimated from $\mathcal{S}_i, i = 1, \cdots, s$.

We name the estimated direct pseudo-elasticity from this procedure as the cross-validation direct pseudo-elasticity (CV-DPE). We compute CV-DPE of (6.17) over a path of λ_1, and fix $\lambda_2 = 0.01$ obtained from the Greedy Local Continuation search. The results for each injury outcome category are displayed in Figure 6.5. Since the model parameters change when λ_1 varies, Figure 6.5 provides a visual inspection of CV-DPE with different degrees of freedom in the model. From the CV-DPE analysis, alcohol increases the probability of suffering severe and fatal injuries. Meanwhile, the probability of experiencing only visible injuries and the chance of free from injuries in an accident is greatly reduce. In addition, alcohol is the first factor selected into the model, when λ_1 decreases. This is an indication about the importance of alcohol factor in predicting the accident results. On the contrary, wearing seatbelt improves the chance of no injury or suffering only visible injuries from accidents by more than 50%, reducing the odds of suffer severe injuries or fatality by more than 50% according to the CV-DPE result from the estimated model with best F-1 score. Consumption of drugs increases the likeliness of fatality by more than 200% according to the model selected by the cross-validation, while reducing the chance of all other injury categories. Factors related to violation of traffic laws, i.e., over-speeding, improper tuning, wrong side of road, all lead to greater chance of fatality, but relatively less impactful compared to the influence of drug and alcohol.

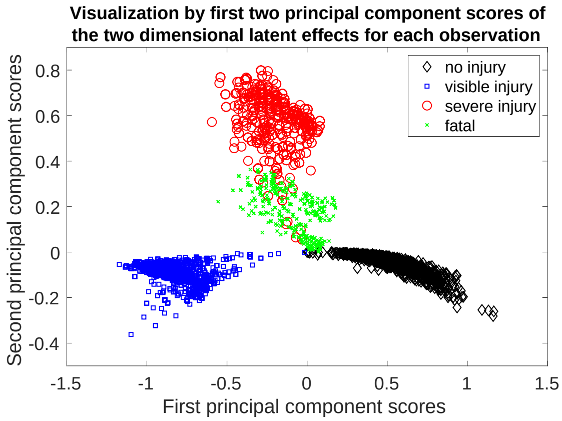

Visualization by first two principal component scores of the two dimensional latent effects for each observation

Figure 6.6. Latent heterogeneity $\boldsymbol{v}_n$ inferred from the train data. rank($\boldsymbol{\Upsilon}$) $= 2$. Each point is represented by the first two principal component scores.

The latent effect model (6.17) also has the advantages of providing insights into the individual heterogeneity. The rank of $\boldsymbol{\Upsilon}$ deceases as λ_2 increases. Hence, (6.17) can be used as a dimensional reduction tool. The resulting $\boldsymbol{\Upsilon}$ learned from model (6.17) has rank two at the λ_2 selected from the Local Greedy Continuation search process. Hence, each $\boldsymbol{v}$ can be visualized by a two dimensional vector, represented by their principal component scores. The first two principal component scores of the latent heterogeneity $\{\boldsymbol{v}_n\}_{n=1}^N$ inferred from each training point are shown in Figure 6.6. There are four well identified clusters, each cluster is dominated by samples from one injury category. This is an indication of the clustering effect in the individual heterogeneity. Moreover, the cluster with majority of severe injury accidents and the cluster comprises mostly fatality cases are overlapped with each other.

6.6 Conclusion

We present a latent effect logistic model based on sparse and low-rank decomposition between the homogeneous effect and heterogeneous effect of observations. The formulation has the advantages of preserving model convexity while capturing the latent individual heterogeneity. The optimization problem from the model is solved by a Fast Accelerated Proximal Gradient with Adaptive Restarting algorithm. A Greedy Local Continuation search process is developed to enable efficient exploration of model hyperparameters. We demonstrate that low-rankness is a result of different data-generating process, and validate through experiments clustering gave rise to the low-rankness of latent the heterogeneity in accident observations. The usefulness of the model is demonstrated on analyzing traffic accident factors. From the analysis, drug and alcohol are found to be the factors with largest impact on the probability of suffering fatality in an accident, whereas the usage of seatbelt greatly improves the chance of avoiding injuries.

Chapter 7

Matrix Factorization for Network Analysis

This chapter describes a technique for analyzing network structure based on matrix factorization. The problem of ranking vertices in large-scale networks is a well-studied, interdisciplinary topic. There are several characterizations of vertex and edge centrality, and these can be used to assess the global importance of individual entities in a network. PageRank [167], for instance, is a popular centrality measure that was initially motivated by web search, but is now widely used for network analysis [168]. Automated community identification [169] is another popular computation in network science. Ranking or centrality analysis of communities [170] is relatively less-studied. Since communities are now frequently used to characterize networks, methods that analyze or rank the importance of communities aid in the component-level centrality analysis of a network.

In this work, a new unsupervised method to analyze the community or clustering structure in a weighted network is presented. Assume that edge weights denote the strength of interactions between vertices. Our method generates weights corresponding to each community, that aim to explain how well the the currently-available community information explain the strength of interactions between vertices. The sorted ordering of weights can also be used for ranking communities. For example, in an online social network data with friendship links, weights of links, and user-community information, this method can be used to identify the most active communities, or the global contribution of the communities in explaining friendship links. Given a co-authorship network and community structure information for individual authors, we can apply our proposed method to identify important author communities. Networks can be constructed on traffic accident observational datasets. Given the crash observations, the geographical regions and accident factors can be represented by nodes on a bipartite graph, where the edges between locations and factors are weighted by the number of accidents occurred in a region with the factors recorded.

Our method is based on a factorization, in an approximate sense, of the adjacency matrix corresponding to the weighted network. An advantage of this factorization is that it has a clear combinatorial interpretation. We assume that community structure is the driving factor for forming links between vertices. The factorization and the weights obtained determine important communities. This chapter is based on the work in [171].

7.1 Mathematical Formulation

Consider an undirected, weighted network $G(V, E, W)$ with n vertices (set V), m edges (edge set E), p communities, and positive real-valued edge weights (an $n \times n$ matrix W). Assume that the community membership information is encoded in a binary association matrix $A \in \mathbf{R}^{n\times p}$:

$$A_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to community } k \\ 0 & \text{otherwise} \end{cases}$$

Let W denote the adjacency matrix of the weighted network. Observe that the $n \times n$ matrix $C = AA^T$ gives valuable structural information about the network G: specifically, C_{ij} gives the number of communities common to both i and j. Therefore, the network corresponding to matrix C could be thought of as a weighted graph, where the edge weights denote the number of common communities that two vertices belong to. C also has self loops giving the number of communities that a vertex belongs to. We will call the weighted network represented by C the *community structure network*.

Let $\mathbf{a}_k$ denote the k^{th} column of A.

$$C = AA^T = \sum_{k=1}^{p} \mathbf{a}_k \mathbf{a}_k^T.$$

Each of these rank-one matrices $\mathbf{a}_k\mathbf{a}_k^T$ also have a useful combinatorial interpretation:

$$(\mathbf{a}_k\mathbf{a}_k^T)_{ij} = \begin{cases} 1 & \text{if } \textit{both} \text{ node } i \text{ and node } j \\ & \text{belong to community } k \\ 0 & \text{otherwise} \end{cases}$$

Therefore, the rank-one matrix $\mathbf{a}_k\mathbf{a}_k^T$ provides information about the community structure for the k^{th} community. This is illustrated via the following example on a graph with 4 vertices and 3 communities:

Example.

$$\begin{aligned} C = AA^T &= \sum_{k=1}^{3} \mathbf{a}_k\mathbf{a}_k^T \\ &= \underbrace{\begin{pmatrix}1\\0\\1\\0\end{pmatrix}\begin{pmatrix}1&0&1&0\end{pmatrix}}_{\text{community 1}} + \underbrace{\begin{pmatrix}0\\1\\1\\0\end{pmatrix}\begin{pmatrix}0&1&1&0\end{pmatrix}}_{\text{community 2}} + \underbrace{\begin{pmatrix}1\\0\\0\\1\end{pmatrix}\begin{pmatrix}1&0&0&1\end{pmatrix}}_{\text{community 3}}. \end{aligned}$$

Now suppose each community has an associated nonnegative weight d_{kk}, such that more important communities receive higher weights. $\sum_{k=1}^{p} d_{kk}\mathbf{a}_k\mathbf{a}_k^T$ generates a *reweighted community structure matrix*, in which the ij^{th} entry takes into account both the number of common communities connecting vertices i and j, and the importance of these communities in the global sense. Since the common notion of a cluster or a community is that there are stronger intra-cluster edge links than inter-cluster links, it is reasonable to assume that the global structure and connectivity of the network $G(V, E, W)$ can be largely explained by the community structure and intra-cluster links. In addition, assume that edge weights W_{ij} in the network reflect both the number of common community labels between i and j, and the weights of those communities. We could decompose the weighted adjacency matrix W as

$$W = ADA^T + R = \sum_{k=1}^{p} d_{kk}\mathbf{a}_k\mathbf{a}_k^T + R, \tag{7.1}$$

where D is a diagonal matrix with nonnegative diagonal elements d_{kk}. ADA^T explains the contribution of community structure to W (with the *ideal* community being a clique), and R is the *residual portion.* The community weights D are computed by approximating the weighted adjacency matrix W by the reweighted community structure matrix and minimizing R, via the following optimization problem:

$$\min_{D \in \mathbf{R}^{p \times p}} \|W - ADA^T\|_F^2, \quad \text{subject to } D \text{ being a nonnegative diagonal matrix.} \tag{P1}$$

We comment on the combinatorial interpretation of the values in the optimal solution matrix D^* in Section 7.3.2. In particular, for the case when A encodes non-overlapping communities, the solution to the optimization problem P1 corresponds to the average internal interaction strength between nodes in each community.

7.2 Connections to Prior Work

This chapter addresses a graph analytic problem via matrix factorization. Specifically, given a list of potential communities, we want to analyze which communities are more important in terms of generating the observed links. Our problem is related, but not identical to the community detection problem on networks. The method presented here could be considered as a post-processing step to analyze the outcome of a clustering algorithm. There is a large body of literature on matrix decomposition approach for graph clustering and community detection.

The optimization problem we propose is related to binary biclustering methods and nonnegative matrix factorizations. Li [172] proposes a method for biclustering binary matrices. Given an $m \times f$ binary entity-feature association matrix S, whose rows correspond to entities, columns correspond to features, the binary biclustering problem aims to simultaneously group entities into K clusters, and features into L clusters, by factorizing S as $S = PXQ^T + R$. Here, P, Q, X are unknown matrices to be solved, and R is the reconstruction residual. The $m \times K$ matrix P is binary, designating cluster memberships for the data. The rows of P record data entries and columns represent cluster labels, and so P itself is also an association matrix between data and data clusters. Similarly, the $f \times L$ binary association matrix L designates feature cluster memberships for each feature. $X \in \mathbf{R}^{K \times L}$ is a dense matrix mapping both the cluster memberships of entities and cluster memberships of features to approximate S. Although both this method and our approach factorize a matrix into a binary association matrix, the formulation by Li [172] is created for purpose of clustering, and the matrix X does not yield any combinatorial information. Nonnegative Matrix Factorization (NMF) [173, 174] is a well-known method for principal component analysis, with restrictions that both the data factor and the component factor have to be nonnegative. NMF is widely used in image analysis and computer vision. However, directly applying NMF on adjacency matrices corresponding to networks does not convey information about combinatorial properties of the adjacency matrix, whereas our factorization in problem P1 gives new insights. The Bounded Nonnegative Matrix Tri-Factorization [175] is a method proposed for overlapping community detection. Mathematically, it factorizes the weighted adjacency matrix W into UBU^T, where U_{ij} represents the probability that node i belongs to community j. The matrix B here is not restricted to be a diagonal matrix, and it models the interaction between communities. Wang et al. [176] formulated an optimization problem similar to ours for community detection purpose.

Our work is also related to literature on understanding community structure in networks [177–179]. Yang and Leskovec [180] studied scoring functions to define a good community. We take an optimization approach to produce the goodness scores.

We next show that our optimization problem P1 can be transformed into a nonnegative least squares (NNLS) problem. NNLS is often used as a subroutine in solving matrix approximation problems involving nonnegative constraints [181, 182].

7.3 Solution to Matrix Factorization

7.3.1 Connection to Nonnegative Least Squares

The key idea in reducing optimization problem P1 to an NNLS problem is to decouple ADA^T into matrix-vector products and transform W into a vector accordingly. Minimizing $\|W - ADA^T\|_F^2$ is the same as minimizing the squared l_2-norm of the differences over each column.

Since $ADA^T = \sum_{k=1}^{p} d_{kk}\mathbf{a}_k\mathbf{a}_k^T$, the i^{th} column of ADA^T is equal to the linear combination of the i^{th} columns of the rank-one matrices $\mathbf{a}_k\mathbf{a}_k^T$. Collect the the i^{th} columns of each rank-one matrix $\mathbf{a}_k\mathbf{a}_k^T$ into a matrix, and define the following *matrixize* operator:

$$\mathrm{Mat}_i(A) = \left[(\mathbf{a}_1\mathbf{a}_1^T)_i, (\mathbf{a}_2\mathbf{a}_2^T)_i, (\mathbf{a}_3\mathbf{a}_3^T)_i, \cdots, (\mathbf{a}_p\mathbf{a}_p^T)_i\right]. \tag{7.2}$$

Note that the dimensions of $\mathrm{Mat}_i(A)$ are $n \times p$.

Since $D \in \mathbf{R}^{p\times p}$ is a diagonal matrix, we could also store the p diagonal elements as a column vector. Define the vectorization operator over the diagonal matrix to be

$$\mathbf{x} := \text{vec-diag}(D) = \left[d_{11}, d_{22}, \cdots, d_{pp}\right]^T$$

Then, we have

$$\mathrm{Mat}_i(A)\mathbf{x} = \sum_{k=1}^{n} d_{kk}\left(\mathbf{a}_k\mathbf{a}_k^T\right)_i = \left(ADA^T\right)_i.$$

Hence, $\|\mathbf{w}_i - \mathrm{Mat}_i(A)\mathbf{x}\|_2^2$ gives the l_2-norm difference in the i^{th} column. Using the column-wise vectorization operator, we could write $\mathrm{vec}(W) = \left[\mathbf{w}_1; \mathbf{w}_2; \cdots; \mathbf{w}_m\right] := \mathbf{v}$, and

$$M = \left[\mathrm{vec}\left(\mathbf{a}_1\mathbf{a}_1^T\right), \mathrm{vec}\left(\mathbf{a}_2\mathbf{a}_2^T\right), \cdots, \mathrm{vec}\left(\mathbf{a}_p\mathbf{a}_p^T\right)\right].$$

We obtain

$$\|W - ADA^T\|_F^2 = \|\mathbf{v} - M\mathbf{x}\|_2^2.$$

Therefore, we have transformed problem P1 to an equivalent Nonnegative Least Squares (NNLS) problem:

$$\min \|\mathbf{v} - M\mathbf{x}\|_2^2, \quad \text{subject to } \mathbf{x} \geq 0. \tag{P1-NNLS}$$

NNLS is a well-studied optimization problem. The Active-Set method by Lawson and Hanson [183] is a well-known algorithm. Luo et al. recently improved the Active-Set method with QR update/downdates [184]. Another popular method for solving large-scale sparse NNLS problems is the Coordinate Descent method [185–187]. We describe how Coordinate Descent can be applied to this problem in the next section. Since A is a sparse binary matrix, the transformed matrix M is also sparse and binary. This fact allows us to optimize the Coordinate Descent method and provide an upper bound on the solution. We outline the overall ranking scheme in Algorithm 8.

Algorithm 8 ADA^T ranking.

Input: Weighted adjacency matrix W, binary association matrix A with community information.

Output: Community weights/ranks given by D_{rec}.

```
$\mathbf{v} \leftarrow$ vec-diag($W$)    ▷ vectorized $W$ column by column.
for $k = 1 : p$ do
    Compute vec $\left(\mathbf{a}_k \mathbf{a}_k^T\right)$
end for
$M \leftarrow \left[\text{vec}\left(\mathbf{a}_1 \mathbf{a}_1^T\right), \text{vec}\left(\mathbf{a}_2 \mathbf{a}_2^T\right), \cdots, \text{vec}\left(\mathbf{a}_p \mathbf{a}_p^T\right)\right]$
$\mathbf{x} \leftarrow$ NNLS-Coordinate-Descent($M$, $v$, $threshold$)
Sort $\mathbf{x}$ to produce a ranking.
$D_{rec} \leftarrow \text{diag}(\mathbf{x})$, $W_{rec} = AD_{rec}A^T$
```

7.3.2 Solution Upper Bound and Interpretation

Theorem. *Let D^* be the optimal solution to problem P1. Let n_k denote the number of members in community C_k, and define δ_{ijk} as follows:*

$$\delta_{ijk} = \begin{cases} 1 \text{ if } i \in C_k \text{ and } j \in C_k \\ 0 \text{ otherwise.} \end{cases}$$

We have the following upper bound on each component of D^*:

$$d_{kk}^* \leq \frac{\sum_{j=1}^{n} \sum_{i=1}^{n} \delta_{ijk} w_{ij}}{n_k^2} \tag{7.3}$$

$$= \frac{\sum_{i \in C_k} \sum_{j \in C_k} w_{ij}}{n_k^2} \tag{7.4}$$

Proof. Let $\mathbf{x}^*$ denote the optimal solution to P1-NNLS, i.e., $d_{kk}^* = x_k^*$. The component-wise upper bound to the solution of NNLS problem $\mathbf{x}^* = \arg\min_{\mathbf{x} \geq 0} \|\mathbf{v} - M\mathbf{x}\|_2^2$ (Theorem 7 in [188]) is

$$x_k^* \leq \max\left(0, \frac{\mathbf{m}_k^T \mathbf{v}}{\mathbf{m}_k^T \mathbf{m}_k}\right).$$

Recall $\mathbf{v} = \mathrm{vec}(W)$, and

$$M = \left[\mathrm{vec}\left(\mathbf{a}_1\mathbf{a}_1^T\right), \mathrm{vec}\left(\mathbf{a}_2\mathbf{a}_2^T\right), \cdots, \mathrm{vec}\left(\mathbf{a}_p\mathbf{a}_p^T\right)\right],$$

the k^{th} column of M is the vectorization of $\mathbf{a}_k\mathbf{a}_k^T$ column-by-column. Hence, $\mathbf{m}_k = \left[\left(\mathbf{a}_k\mathbf{a}_k^T\right)_1; \left(\mathbf{a}_k\mathbf{a}_k^T\right)_2; \cdots; \left(\mathbf{a}_k\mathbf{a}_k^T\right)_n\right]$; we have

$$\begin{aligned}\mathbf{m}_k^T\mathbf{v} &= \sum_{j=1}^{n} \left(\mathbf{a}_k\mathbf{a}_k^T\right)_j^T \mathbf{w}_j \\ &= \sum_{j=1}^{n}\sum_{i=1}^{n} \left(\mathbf{a}_k\mathbf{a}_k^T\right)_{ij} w_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} \delta_{ijk} w_{ij}.\end{aligned}$$

Also, using the fact that $\mathbf{a}_k\mathbf{a}_k^T$ is a binary matrix,

$$\begin{aligned}\mathbf{m}_k^T\mathbf{m}_k &= nnz(\mathbf{m}_k) \\ &= \sum_{j=1}^{n} nnz\left(\left(\mathbf{a}_k\mathbf{a}_k^T\right)_j\right) \\ &= \sum_{j=1}^{n}\sum_{i=1}^{n} \left(\mathbf{a}_k\mathbf{a}_k^T\right)_{ij} = n_k^2\end{aligned}$$

Notice that $\frac{\mathbf{m}_k^T\mathbf{v}}{\mathbf{m}_k^T\mathbf{m}_k}$ is nonnegative, and so the result follows. □

The above theorem says that the weight for community k is upper-bounded by the sums of weight of all links $\langle i, j\rangle$ that belong to community k, divided by the total number of possible links (including self loops $\langle i, i\rangle$). In addition, the upper bound is tight when A encodes non-overlapping communities. For non-overlapping communities, we may rearrange the nodes so that in the adjacency matrix W, nodes of the same community are indexed sequentially. We also rearrange the rows of A accordingly. After row rearrangements,

$$ADA^T = \begin{pmatrix} d_{11}\mathbf{1}_{n_1} & \cdots & \cdots & 0 \\ 0 & d_{22}\mathbf{1}_{n_2} & \cdots & 0 \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & \cdots & d_{pp}\mathbf{1}_{n_p} \end{pmatrix},$$

where $\mathbf{1}_{n_k}$ is a block matrix of ones of size $n_k \times n_k$ (recall n_k is the number of nodes belonging to community k), and ADA^T is a block diagonal matrix. Therefore, for non-overlapping communities, the minimization problem P1 can be decomposed into minimization over each block,

$$\arg\min \|W - ADA^T\| = \arg\min \sum_{k=1}^{p} \sum_{i \in C_k} \sum_{j \in C_k} (w_{ij} - d_{kk})^2.$$

The analytical solution in each component d_{kk} is precisely the upper bound in Equation (7.4).

7.3.3 Coordinate Descent solution strategy

NNLS is a quadratic optimization problem and could be reformulated as

$$\arg\min_{\mathbf{x} \geq 0} \|\mathbf{v} - M\mathbf{x}\|_2^2 = \arg\min_{\mathbf{x} \geq 0} \frac{1}{2}\mathbf{x}^T M^T M\mathbf{x} - \mathbf{x}^T M^T \mathbf{v}.$$

When fixing all coordinates of $\mathbf{x}$ but one, the objective function is univariate quadratic, and therefore has an analytical solution [186],

$$\arg\min_{x_k \geq 0} \|\mathbf{v} - M\mathbf{x}\|_2^2 = \max\left(0, x_k - \frac{\left(M^T M\mathbf{x} - M^T\mathbf{v}\right)_k}{(M^T M)_{kk}}\right) \tag{7.5}$$

The Coordinate Descent method iteratively updates one variable at a time until convergence. The univariate quadratic problem has a unique minimum with analytical form in Equation (7.5), and therefore the iterative scheme is guaranteed to converge to a stationary point. In our case, P1-NNLS is a convex problem, and hence the stationary point is also a global minimum. During the $(t+1)^{th}$ iteration, the update to x_{k+1} depends on the quantity $\mathbf{u} := M^T M\mathbf{x} - M^T\mathbf{v}$. After optimizing x_k in the $(t+1)^{th}$ iteration, the only update to $\mathbf{u}$ comes from the change in the k^{th} column of $M^T M\mathbf{x}$. Therefore, as in [186], we could initialize $\mathbf{u}$ to $-M^T\mathbf{v}$, and keep track of this quantity by computing

$$\mathbf{u}_{new} \leftarrow \mathbf{u}_{old} + (x_{k,new} - x_{k,old})M^T\mathbf{m}_k,$$

after optimizing the k^{th} coordinate during the $(t+1)^{\text{th}}$ iteration. The pseudocode for NNLS-Coordinate-Descent method is provided in Algorithm 9.

NNLS-Coordinate-Descent takes an input ϵ for determining the termination criterion. The algorithm consists of an outer while loop until termination, and an

Algorithm 9 NNLS-Coordinate-Descent algorithm.

Input: M, $\mathbf{v}$, ϵ (*threshold*).

Output: $\mathbf{x} = \arg\min_{\mathbf{x}\geq 0} \|v - M\mathbf{x}\|_2^2$.

```
 1: $\mathbf{x} \leftarrow 0$, $\mu \leftarrow -M^T\mathbf{v}$
 2: $MaxChange \leftarrow$ value greater than $\epsilon$
 3: while $MaxChange > \epsilon$ do
 4:     $MaxChange \leftarrow 0$
 5:     for $k = 1 : p$ do
 6:         $x_k^{new} \leftarrow \max\left(0, x_k - \frac{\mu_k}{(M^TM)_{kk}}\right)$
 7:         $\mathbf{u} \leftarrow \mathbf{u} + (x_{k,new} - x_k)M^T\mathbf{m}_k$
 8:         if $|x_{k,new} - x_k| > MaxChange$ then
 9:             $MaxChange \leftarrow |x_{k,new} - x_k|$
10:         end if
11:         $x_k \leftarrow x_{k,new}$
12:     end for
13: end while
```

inner iteration over each coordinate. The maximum change over all coordinates is recorded in each inner loop iteration. Once *MaxChange* is smaller than ϵ, the algorithm terminates. Note that the dimensions of M are $n^2 \times p$, therefore Algorithm 9 requires $\Theta(n^2p)$ memory if there are no optimizations exploiting the problem structure. The arithmetic cost for updating line 7 is $\Theta(n^2p)$, and the per iteration complexity of the while-loop is $\Theta(n^2p^2)$. If this problem fits in memory, we could precompute and store M^TM instead of doing the matrix-vector products every iteration. The precomputation takes $\Theta(n^2p^2)$. We refer to this as the baseline Coordinate Descent method (BaselineCD). In the next section, we describe an optimized implementation to reduce the time complexity and memory footprint.

7.3.4 A faster solution method

We now describe a faster, memory-reducing modification to BaselineCD to speed up the solution to our optimization problem. This method exploits the special matrix structure of M to reduce the work performed in the matrix-vector products.

The inputs to the original matrix approximation problem P1 are the graph adjacency matrix $W \in \mathbf{R}^{n\times n}$ and the community association matrix $A \in \mathbf{R}^{n\times p}$. We created a larger matrix $M \in \mathbf{R}^{n^2\times p}$ after the transformation to an NNLS problem. However, we now show that we can avoid explicitly computing M. The matrix-vector multiplication $M^T\mathbf{m}_k$ in line 7 of Algorithm 9 could instead be done with A. Observe that we need the values $\mathbf{m}_j^T\mathbf{m}_k$ for all columns $(1 \leq j \leq p)$. From the results in Section 7.3.1, we have $\mathbf{m}_k = \mathrm{vec}(\mathbf{a}_k\mathbf{a}_k^T)$. Hence

$$\mathbf{m}_j^T\mathbf{m}_k = \sum_{r,c}\left(\mathbf{a}_j\mathbf{a}_j^T\right)_{rc}\left(\mathbf{a}_k\mathbf{a}_k^T\right)_{rc} := \left(\mathbf{a}_j\mathbf{a}_j^T\right)\cdot\left(\mathbf{a}_k\mathbf{a}_k^T\right),$$

where the operation denoted with $\cdot$ is sometimes called the *Frobenius product* of two matrices.

Again, recall $\left(\mathbf{a}_j\mathbf{a}_j^T\right)_{rc} = 1$ iff both r and c are members of community j. Hence, $\left(\mathbf{a}_j\mathbf{a}_j^T\right)_c$ is a non-zero column iff c is a member of community j, and we have $nnz(\mathbf{a}_j)$ (note that $nnz(L)$ indicates the number of non-zeros in L) duplicate non-zero columns. Moreover, if column c is a non-zero column of the rank-one matrix $\mathbf{a}_j\mathbf{a}_j^T$, then $\left(\mathbf{a}_j\mathbf{a}_j^T\right)_c$ is $\mathbf{a}_j$. Therefore,

$$\sum_{r,c}\left(\mathbf{a}_j\mathbf{a}_j^T\right)_{rc}\left(\mathbf{a}_k\mathbf{a}_k^T\right)_{rc} = \left(\mathbf{a}_j^T\mathbf{a}_k\right)^2. \tag{7.6}$$

We give a small example to illustrate Equation 7.6:
Let $\mathbf{a}_j = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1\end{bmatrix}$, $\mathbf{a}_k = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1\end{bmatrix}$.

$$\left(\mathbf{a}_j\mathbf{a}_j^T\right)\cdot\left(\mathbf{a}_k\mathbf{a}_k^T\right) = \begin{pmatrix} 1&0&1&0&0&1\\ 0&0&0&0&0&0\\ 1&0&1&0&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 1&0&1&0&0&1 \end{pmatrix} \cdot \begin{pmatrix} 1&0&1&1&0&1\\ 0&0&0&0&0&0\\ 1&0&1&1&0&1\\ 1&0&1&1&0&1\\ 0&0&0&0&0&0\\ 1&0&1&1&0&1 \end{pmatrix} = 9.$$

In this example, only the first, third, and sixth columns of the two matrices add a positive value to their *Frobenius product*. For each of these columns, only the first, third, and sixth rows have nonzero elements in the same component. Hence we have

$$\mathbf{m}_j^T\mathbf{m}_k = \left(\mathbf{a}_j^T\mathbf{a}_k\right)^2. \tag{7.7}$$

Therefore, we do not have to store or compute $M \in \mathbf{R}^{n^2\times p}$. Both the memory and computational complexity depend only on $A \in \mathbf{R}^{n\times p}$ and $W \in \mathbf{R}^{n\times n}$.

Moreover, to store a sparse binary vector, one only needs to record the non-zero indices of the vector in sorted order. Suppose $\mathbf{x}, \mathbf{y} \in \mathbf{R}^m$ are two sparse binary vectors, then computing $\mathbf{x}^T\mathbf{y}$ is equivalent to finding the number of common elements in the non-zero index lists for $\mathbf{x}$ and $\mathbf{y}$. Denote $nnz(\mathbf{x})$, $nnz(\mathbf{y})$ to be the number of non-zero values in $\mathbf{x}$, $\mathbf{y}$. A simple implementation of the array intersection operation for computing $\mathbf{x}^T\mathbf{y}$ takes only $\Theta\left(nnz(\mathbf{x}) + nnz(\mathbf{y})\right)$, which is

Algorithm 10 FastInnerProduct algorithm.

Input: Sorted index arrays corresponding to vectors $\mathbf{a}_i$ and $\mathbf{a}_j$, denoted as $\mathbf{a}_i.nzl$ and $\mathbf{a}_j.nzl$.

Output: $\mathbf{m}_i^T \mathbf{m}_j$

```
$p \leftarrow 1$, $q \leftarrow 1$                                ▷ initialize iterators
$product \leftarrow 0$
while $p \leq nnz(\mathbf{a}_i)$ and $q \leq nnz(\mathbf{a}_j)$ do
    if $\mathbf{a}_i.nzl[p] == \mathbf{a}_j.nzl[q]$ then
        Increment $product$
        Increment $p$ and $q$
    else if $\mathbf{a}_i.nzl[p] < \mathbf{a}_j.nzl[q]$ then
        Increment $p$
    else
        Increment $q$
    end if
end while
$\mathbf{m}_i^T \mathbf{m}_j \leftarrow product^2$
```

faster than $\Theta(n)$ if the size of the largest community is significantly smaller than n. We use this special vector multiplication operation and the identity in Equation 7.7 to develop the FastInnerProduct algorithm (see Algorithm 10).

Using FastInnerProduct($\mathbf{a}_j$,$\mathbf{a}_k$), updating line 7 in Algorithm 9 without precomputing $M^T M$ takes $\Theta(pn_c)$ time, where n_c is the size of the largest community. Therefore, the per-iteration computational complexity is much lower than BaselineCD without precomputation. Similarly, the computational cost of precomputing $M^T M$ becomes $\Theta(p^2 n_c)$. In addition, storing $M^T M$ requires $\Theta(p\gamma)$, where γ is the maximum number of overlapping nodes between any pair of communities.

7.4 Empirical Evaluation

7.4.1 Experimental Setup

Communities in real-world networks deviate significantly from the ideal *clique* notion of a community that we assume in our problem formulation. We show in Section 7.1 that the product AA^T gives a matrix whose ij^{th} entry is equal to the number of common communities i and j belong to. However, the fact that i and j belong to the same community need not guarantee the existence of edge $\langle i, j \rangle$ in

W. We term the locations of W where we would expect edges due to community structure, but there are actually no edges in the real data, as *missing edges*. The missing edge count is a measure of *intra-community edge density*. Also, vertices i and j may be linked in W even if they do not share any communities in common. We consider these edges as the second kind of noise and refer to them as *mixing edges*. The cumulative mixing edge count for a community corresponds to the *number of inter-community edges*. Due to these two reasons, the sparsity pattern of W and ADA^T will not match. We evaluate the proposed method by varying the percentage of missing edges and mixing edges. We would like to empirically identify proportions of missing and mixing edges that are significant enough to alter the global network structure and obtained rankings.

To evaluate the proposed method, we conduct three sets of experiments. The first set of experiments are on small synthetic networks where we generate both W and A, and these serve as sanity and correctness checks. We obtain insight into the impact of noisy edges on the overall result quality from these experiments. The next set of experiments is on the graphs generated using the LFR benchmark [189]. LFR benchmark graphs are random graphs with ground-truth community structure. There are several configurable parameters to generate these graphs, such as the number of nodes with multiple community labels, and the community mixing rate. The third set of experiments are performed on subgraphs extracted from a LiveJournal crawl [180], where ground truth information of community membership is available. Since LFR and LiveJournal are unweighted graphs, edge weights in these two sets of graphs are generated synthetically.

We implement the optimized NNLS-Coordinate-Descent method with the FastInnerProduct (Algorithm 10) in C++ (code available at `https://psu.box.com/s/8o0n2ybxol7rri0g7xamvj70vptwtsh6`), and primarily present results using this code. We have also implemented the baseline approach (Algorithm 9) in MATLAB. We compare the running time and solutions of our implementation with the C++ code for Active-Set method with QR updating/downdating by Luo et al. [184]. Because of the significant memory requirements of the unoptimized Coordinate Descent and the Active-Set methods, it is infeasible to run these algorithms on graphs larger than 4000 vertices. We obtain results on a single server of Cyberstar, a Penn State compute cluster. The server we run our programs on is a dual-socket quad-core Intel Nehalem system (Intel Xeon X5550 processor) with 32 GB main

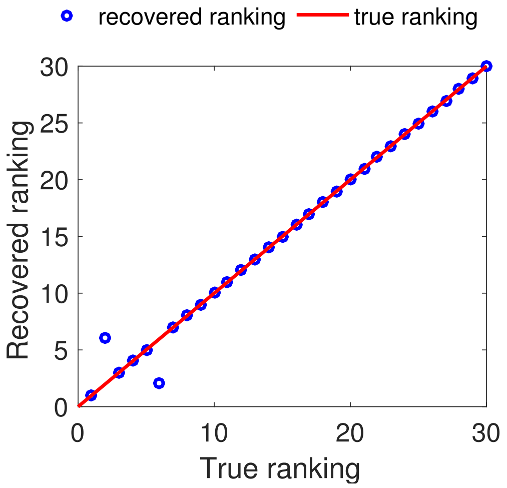


Figure 7.1. Syn-test1: 8.03% noise level, 2% missing edges, 8.05% reconst. error.

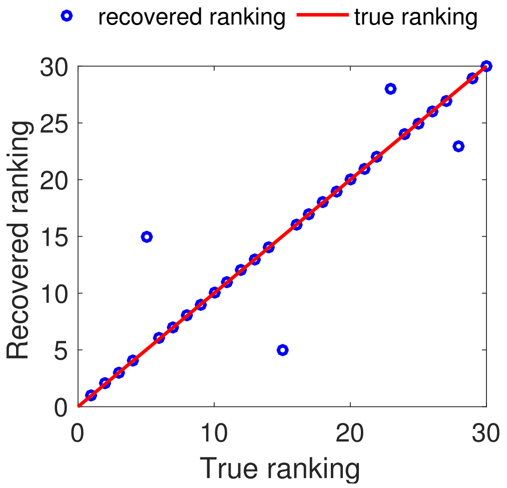


Figure 7.2. Syn-test2: 15.77% noise level, 8.73% missing edges, 13.95% reconst. error.

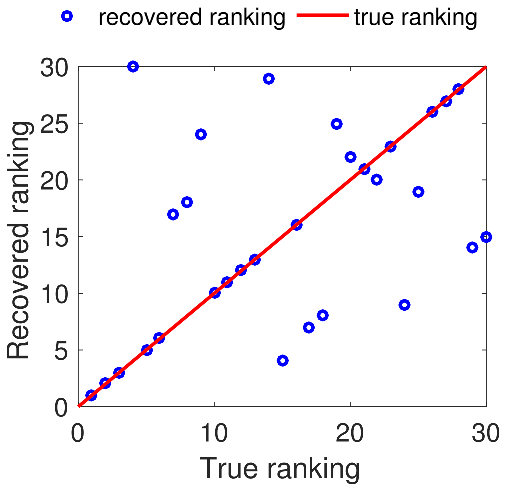


Figure 7.3. Syn-test3: 25.82% noise level, 17.45% missing edges, 19.02% reconst. error.

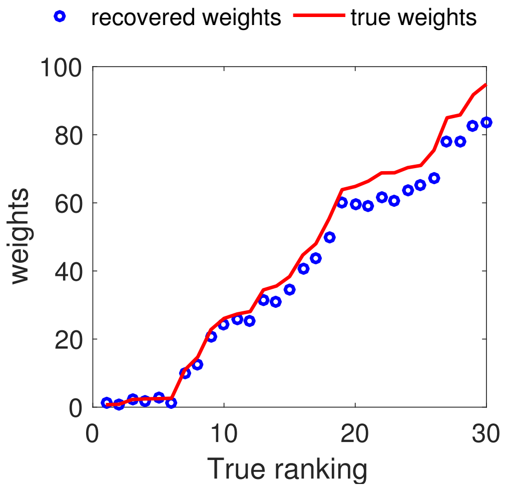


Figure 7.4. Syn-test3: Recovered weights sorted by non-decreasing magnitudes in D_{true}.

memory.

7.4.2 Synthetic graph experiments

We generate synthetic data using the following procedure:

- Initialize a diagonal matrix D_{true} whose entries are drawn uniformly at random from $(0, 100)$.

- Initialize a binary matrix A, whose entries are drawn from Bernoulli distribution Bern(0.5).
- Let $W_{true} = AD_{true}A^T$.
- Set $W_{noise} = W_{true} + \epsilon$, where ϵ are Gaussian white noises added to non-zero entries of W_{true} to mimic the noisy observations on the edge weights. We later tuned the standard deviation of ϵ to adjust the noise level.
- For each nonzero ij^{th} and ji^{th} entries of W_{noise}, delete ij^{th} and ji^{th} by Bernoulli distribution to capture the effect of *missing edges*.

W_{noise} is used as the observed network and D_{rec} is recovered from W_{noise}. The noise level is defined as $\|W_{noise} - W_{true}\|_2/\|W_{noise}\|_2$, and the reconstruction error is computed to be $\|AD_{rec}A^T - W_{true}\|_2/\|W_{true}\|_2$. We create graphs with 300 vertices, 30 communities, and with various noise levels and percentage of missing edges.

We compare rankings given by D_{true} and D_{rec} with varying noise levels and missing edges in Figures 7.1–7.3. The circles in these plots indicate the computed rankings, and the true rankings are sorted. Hence, if all rankings are successfully recovered, the circles will lie on the straight line. The noise level and the percentage of missing edges increases from Syn-test1 to Syn-test3.

The results show that when the noise level and the percentage of missing edges are not very high (Figures 7.1 and 7.2), the proposed method is able to recover most of the exact rankings of the communities. However, when the percentage of missing edges is higher than 15%, and the noise level is around 25%, the results worsen (Figure 7.3).

To further analyze these results, we compare the weights in D_{rec} and D_{true} for the Syn-test3 data. The weights in D_{true} are sorted in non-decreasing order and the component indices of D_{rec} are rearranged according to the non-decreasing order in D_{true} in Figure 7.4. The line plot for weights in D_{true} is monotonically non-decreasing. The dots corresponding to weights in D_{rec} are not monotone, but still follow the trend of the red curve. This indicates that the ranking mismatches are caused by the non-monotone portions of the blue dotted curve, due to communities with close weights in D_{true}. This observation aligns with intuition: when two communities have different but close weights, it is difficult to distinguish them in terms of the proposed ranking.

7.4.3 LFR graph experiments

The LFR benchmark graphs [189] are often used for testing community detection algorithms. The LFR random graphs are extensions of the planted partition model, which incorporates power-law distributions over node degrees and community sizes, and allows overlapping clusters. The list of ground truth communities provides information to construct the association matrix A. Before generating the benchmark graphs, one could configure the number of nodes in the network that have more than one community label, and the number of community labels for these nodes. In addition, one could also supply the mixing parameter $\mu \in [0, 1]$, which is the proportion of edges from a vertex that are inter-community edges. Therefore, $\mu = 0$ means that edges would be formed only between vertices in the same community, and thus there would be no noise due to mixing edges in the network. Other configuration parameters include the number of nodes in the network, as well as upper and lower bounds on the community size. The LFR network also contains noise due to missing edges. Note that in the LFR graph generator (we use the code provided at `https://sites.google.com/site/andrealancichinetti/Home`), increasing the mixing coefficient μ simultaneously increases the percentage of mixing and missing edges.

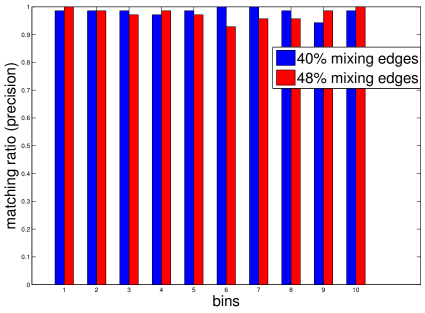


Figure 7.5. LFR-test1: Varying mixing edge percentage and uniform D_{true}.

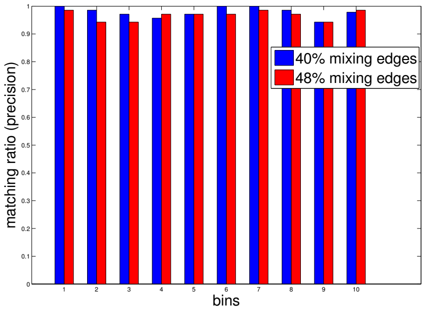


Figure 7.6. LFR-test1: Varying mixing edge percentage and normal D_{true}.

To understand the effect of noisy edges, we perform experiments on the following benchmark graph with overlapping communities: we generate a network with 3000

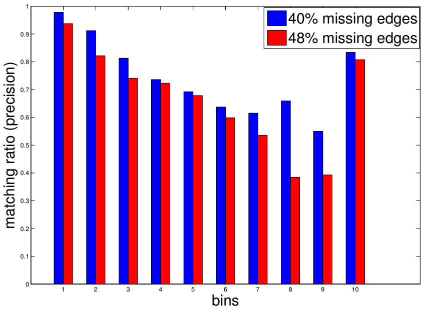


Figure 7.7. LFR-test2: Varying missing edge percentage and uniform D_{true}.

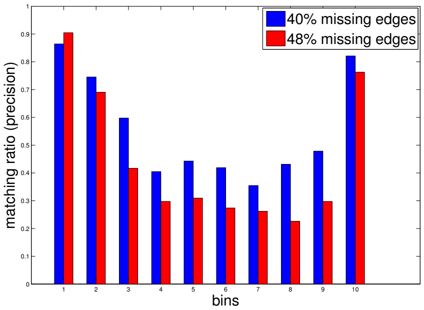


Figure 7.8. LFR-test2: Varying missing edge percentage and normal D_{true}.

vertices and 76 communities. 600 vertices have more then one community label, and each of these 600 vertices belongs to 3 communities. We examined the effect of increasing mixing edges when there are no missing edges (i.e, all intra-community edges were artificially added back). Also, in another experiment, we evaluated the influence of missing edges, where all the mixing edges across communities were removed. To summarize, *LFR-test1*: increase mixing edges, add back all the missing edges, and *LFR-test2*: increase missing edges, filter out all the mixing edges.

In evaluating the efficacy of our method in the subsequent experiments, we split the result ranking of communities into coarser-grained bins.

Definition. *Suppose the true community weights D_{true} are known. Let D_{rec} denote the weights of communities recovered by our factorization. Define the k^{th} bin ($k = 1$ to 10) to be the set of communities belonging to the percentile range $(10(k-1), 10k]$ according to the sorted order of D_{true}. We define the k^{th}* bin matching ratio *to be the proportion of communities in the k^{th} bin whose ranks are in the percentile range $(10(k-1), 10k]$ according to the sorted order of D_{rec}.*

For example, the 1^{st} bin matching ratio is the number of communities that rank in the first 10% in both D_{true} and D_{rec}, divided by the size of the bin. We will describe the construction of the ground-truth D_{true} in each experiment. If the ranking problem is viewed as a multi-class classification problem, and each bin is regarded as a class, the bin matching ratio is equivalent to the *precision* of each bin. In addition, we will use *Spearman's rank correlation coefficient* ρ to measure the overall ranking quality: $\rho = 1 - 6\sum_{i=1}^{p} d_i^2 / p(p^2-1)$, where d_i is the difference in rankings of D_{true} and D_{rec} for community i.

We constructed LFR-test1 using the following steps:

- Generate unweighted version of the LFR benchmark networks, W^{LFR}.
- Create ground-truth community weights D_{true}, where the community weights are drawn from a known distribution.
- Compute $V = AD_{true}A^T$.
- If $V_{ij} > 0$, assign $W_{ij} = V_{ij}$.

Let μ, σ be the mean and standard deviation of values in $\{V_{ij} \mid V_{ij} > 0\}$. If $\langle p, q\rangle \in E^{LFR}$ and $V_{pq} = 0$, i.e, a mixing edge, we draw a random weight r from a normal distribution $\mathcal{N}(\frac{\mu}{2}, \frac{\sigma}{2})$, and set $W_{pq} = r$. This procedure of edge weight construction serves two purposes: first, it reflects our assumption that link interaction strength between two nodes is predominantly explained by community structure and weights. Second, since there are no accepted methods for community rankings, pre-specifying D_{true} and using it as ground truth allows us to analyze our method and do control-variated experiments. In addition, in the fourth step, all the missing edges within communities are added back, hence all the communities become cliques in the network. The last step assigns edge weights to mixing edges, assuming mixing edges represent weaker connection strengths since they link across communities.

We first vary the percentage of mixing edges and repeat the experiment ten times using uniformly-distributed and normally-distributed community weights in D_{true}. The average bin matching ratios under these two distributions are shown in Figures 7.5 and 7.6. The precision are over 0.9 in all bins for both uniformly-distributed and normally-distributed ground truth community weights. There are no substantial distortions on retrieving the true cluster rankings when the percentage of mixing edges increased. Also, the Spearman rank correlation coefficient ρ was close to 1 in all the cases. This experiment shows that when the communities in the network are actually cliques and explain most of the edge weights, our method is robust against noise due to mixing edges. For LFR-test2, the graph generation process is as follows:

- Generate unweighted version of the LFR benchmark networks, W^{LFR}.

- Create ground-truth community weights D_{true}, where the community weights are drawn from a known distribution.
- Compute $AD_{true}A^T$.
- Let $W_{ij}^{LFR} = (AD_{true}A^T)_{ij}$ if $\langle i, j\rangle \in E^{LFR}$, $W_{ij}^{LFR} = 0$ otherwise. Hence, if $(AD_{true}A^T)_{ij} = 0$ then $W_{ij} = 0$, and all mixing edges are filtered out.

The first three steps are identical to the previous test. In addition, the percentage of missing edges in this test are the same as the percentage of mixing edges in the previous test. The results are shown in Figures 7.7 and 7.8.

When there are no mixing edges across communities in the graph, but with the same percentage of missing edges as the percentage of mixing edges in the previous experiment, the matching ratios in each bin deteriorate significantly. Moreover, we observe that the precision is higher for low-range bins (0-40%) and high-range bin (90-100%) in LFR-test2. This is more pronounced when the ground truth community weights are normally-distributed: the bar heights in Figure 7.8 follow a clear U-shaped curve. The spearman's ρ on LFR graph with 40% missing edges and 48% missing edges are 0.9728 and 0.9703 respectively for uniformly distributed weights, 0.9641 and 0.9589 for normally distributed weights. One possible explanation is that the weights for the highest-ranked and lowest-ranked communities are the ones that are farthest from the mean. Therefore, the ranking of communities with more extremal weights, which corresponds to highest-ranked and lowest-ranked communities, will be most effectively recovered. For normally-distributed community weights, most of the true community weights are concentrated around the mean, and there are less extremal weights. As a result, the precisions for the middle range bins in Figure 7.8 underperform the precisions for the highest and lowest bins by a significant margin.

We further investigate the distributions of the recovered weights. The weights in D_{true} are sorted in non-descending order and the component indexes of D_{rec} are rearranged according to the weight orderings in D_{true}. Figure 7.9 plots the recovered weights versus the normally-distributed true weights when there are 40% mixing edges in the graph in sorted order. Similarly, Figure 7.10 shows the weight comparison when there are 40% missing edges, but without any mixing edges. In the second case, there are larger discrepancies between the true weights and the

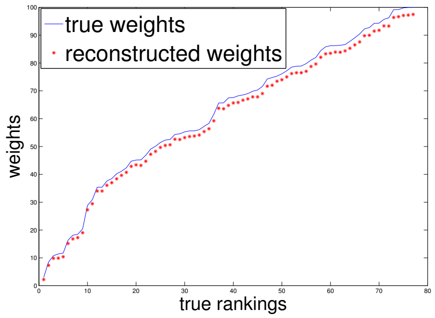


Figure 7.9. LFR-test1: 40% mixing edges.

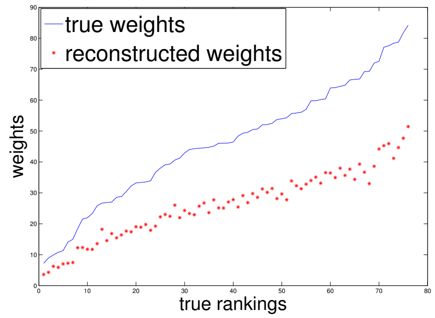


Figure 7.10. LFR-test2: 40% missing edges.

recovered weights. The recovered weights also oscillate when the true weights are sorted monotonically.

To summarize these two control-variated tests, each network has the same percentages of noisy edges in LFR-test1 and LFR-test2; however in LFR-test1, all noise is attributed to mixing edges and in LFR-test2, all noise is attributed to missing edges. Therefore, the experiments indicate that our method is more robust against noise due to mixing edges, but less effective in the presence of a large proportion of missing edges. The community weight and ranking recovery is most effective when the communities in the network are cliques.

7.4.4 LiveJournal experiments

We now run experiments on a snapshot of the LiveJournal network with ground-truth communities [180]. LiveJournal is a user-user network with ground-truth communities defined by interest groups that users have joined. From the data provided by Yang and Leskovec, we extracted 3818 communities, ranging in size from 5 to 50 users per group. There are 51367 users in all, and the extracted graph is the one induced by these 51367 users. Therefore, $W \in \mathbf{R}^{51367\times 51367}$, $A \in \mathbf{R}^{51367\times 3818}$. In this experiment, we compare our method to a *baseline* method. To the best of our knowledge, there is no prior work on optimization-based community ranking similar to what we propose. However, there is a lot of prior research on ranking individual nodes in the network based on algebraic, graph-theoretic, and geometric centralities. We define our baseline method to use a straightforward extension of node centrality to groups. For each community, we sum up the individual centrality

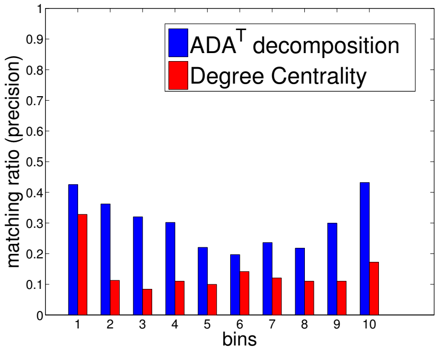


Figure 7.11. LJ-test1: uniform D_{true}.

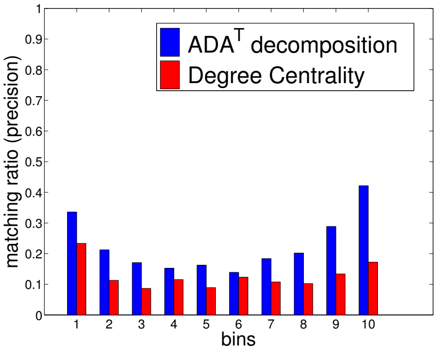


Figure 7.12. LJ-test2: normal D_{true}.

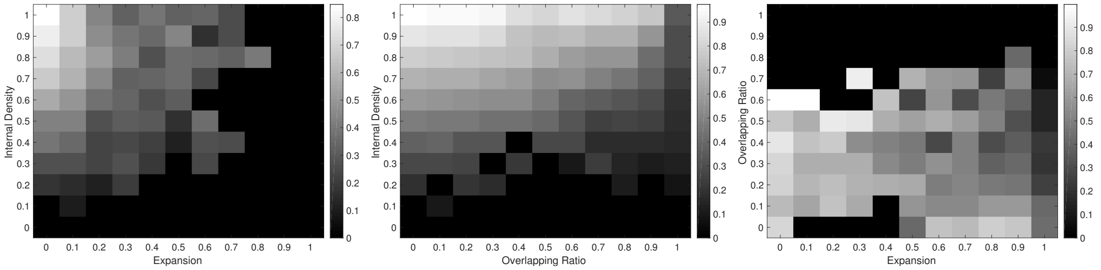


Figure 7.13. LJ-test1: Comparing obtained ranking to other community structure metrics.

scores for all nodes belonging to that community. The node centrality measure we used is degree centrality for weighted graphs, defined as: $c(i) = \sum_{j \in V} w_{ij}$.

We specify the ground-truth weight distribution D_{true} as before. We set $W_{ij} = \left(AD_{true}A^T\right)_{ij}$ if both $\left(AD_{true}A^T\right)_{ij} > 0$ and $\langle i, j \rangle \in E^{LJ}$. Therefore, we did not reintroduce any missing edges. There are 52% missing edges in this network. For mixing edges, we assign random weights from a normal distribution $\mathcal{N}(\frac{\mu}{2}, \frac{\sigma}{2})$, where μ and σ are the mean and standard deviation of the weights on other edges. Again, this setup allows us to determine the efficacy of our method. Unlike the previous experiments, we now use a real-world graph topology. Bin matching ratios for different community weight distributions are given in Figure 7.11 and Figure 7.12.

Although the matching ratios are not as high as the LFR networks, our factorization still outperforms the degree centrality ranking method. The low values for some bins are a result of the large percentage of missing edges in the network (52%). It also indicates that community ranking is a non-trivial task, and that a straightforward extension of node centrality to rank communities is ineffective. The Spearman's rank correlation coefficient ρ using the proposed ranking method is 0.7244 for uniformly distributed community weights and 0.6061 for normally distributed weights. These are also significantly higher than those obtained using the baseline method (0.2942 and 0.1829, respectively). Therefore, even though our proposed method does not perform very well locally on each bin of this data set, the ρ values are all above 0.5, which shows that globally, it produces a reasonable outcome.

We compare the community scores obtained from P1 to traditional cluster quality metrics. When formulation P1 is applied on unweighted graphs (assuming a self-loop on each node), Theorem states that the cluster scores we obtained in the optimal solution degenerates to internal density for non-overlapping communities. We want to understand the scores and hence rankings for other situations. For each community S in the LiveJournal crawl, we computed the following scores:

- internal density [178],
  $$\frac{e(S)}{|S|^2},$$
  where $e(S)$ is number of edges whose end points are both inside the cluster S.
- overlapping ratio,
  $$\frac{|v \in S, \exists S' s.t. v \in S'|}{|S|},$$
  which measures the percentage of vertices in S that are shared with another community.
- expansion [178],
  $$\frac{|\langle u,v\rangle \in E, u \in S, v \notin S|}{|\langle u,v\rangle \in E, u \in S, v \notin S| + |\langle u,v\rangle \in E, u,v \in S|},$$
  which is the percentage of mixing edges among all edges attached to S.

All these three community quality metrics are between 0 and 1. Each of the above three measurements is then split into ten equally spaced intervals and zero. For every pair of these metrics, the communities are assigned into the corresponding box bounded their values. The average value of the community scores from P1 is computed for each box. The results are displayed by heatmaps in Figure 7.13. The color intensity in most areas in the heatmaps are gradually changing. Note that if there are no communities fall in certain boxes of the metrics, the average values are set to 0.

The leftmost heatmap in Figure 7.13 shows the relation between community scores learned from P1 and internal density and expansion ratio. The higher P1 community scores are concentrated on the upper-left corner. The scores gradually reduce as the expansion ratio increases or the internal density decreases.

The center heatmap in Figure 7.13 conveys how internal density and overlapping ratio affect scores in P1. Again, higher community scores learned by P1 are distributed on the upper-left corner of the heatmap. The averaged scores of communities gradually reduce in lower-right portion of the heatmap. It also confirms the result of Theorem , that the scores learned by P1 are equal to the internal density for the case of non-overlapping communities.

The relation between the overlapping ratio, expansion ratio, and P1 scores is visualized in the rightmost heatmap. The averaged community scores in P1 decrease from bottom-left to upper-right. Communities that correspond to the first two columns of the heatmap have low expansion ratios. Note that P1 assigns high scores to communities within the first two columns of the heatmap, even some of them have high overlapping ratio. For example, communities with overlapping ratio of 0.4 in the first two columns are those sharing a large fraction of their nodes with other communities, but have a relatively low percentage of edges connecting to nodes outside their communities.

7.4.5 Running time Comparison

In this section, we present some results comparing running time of our optimized version of Coordinate Descent (OptCD), BaselineCD, and the Active-set based code of Luo et al. (ActiveQR) [184]. We solve the optimization problem on an LFR network with 3000 nodes, 76 communities, and 67744 edges, without reintroducing any missing edges or filtering of the mixing edges. The running times of the three methods are 0.49 seconds, 99 seconds, and 130 seconds, respectively. This result is an indicator of the considerable speedup achieved with our optimization. For larger graphs with millions of vertices and edges, it is infeasible to apply BaselineCD or ActiveQR due to memory restrictions, but OptCD still runs to completion in the order of minutes. The running time of OptCD is mainly dominated by the computation of the matrix-vector product $M^T\mathbf{m}_k$ in line 7 of Algorithm 9.

7.5 Chapter Conclusion

This chapter introduces a new community ranking scheme that is based on factorization of the adjacency matrix. This community ranking scheme has a simple combinatorial interpretation: using community structure to reconstruct the network, and distributing the internal interaction strength among overlapping communities. The interpretation of this ranking formulation is derived via analyzing the solution upper bound and the special case when communities has disjoint node sets. I show that the optimization problem reduces to Nonnegative least squares, and design a fast and memory-efficient solution strategy for this problem. The solution strategy exploits the sparsity structure inherent in the input matrices. The empirical evaluation reveals scenarios when this method is readily applicable. I find that synthetically-initialized rankings can be recovered with high precision if the intra-cluster weight/edge densities in the network being considered are sufficiently high. This scheme is also quite robust to addition of inter-cluster edges. The rankings of communities can be recovered even with a very high percentage of inter-cluster edges (greater than 50% in some cases) in the graph, provided the clusters have high intra-cluster density. The ranking scheme is also evaluated on two virtual social networks with known ground-truth community information and interpretations of results are presented.

Chapter 8

Graph Sparsification as a Knapsack Problem

A simple formulation for graph sparsification based on the 0/1 knapsack problem is presented in this chapter. Informally, the 0/1 knapsack problem can be stated as follows: given a collection of items, each with an associated weight and value, the goal is to choose a subset of the items such that the cumulative value of the chosen items is maximized, and the total weight of the selected subset of items is less than or equal to a user-specified weight budget. The main idea of our GSK formulation is to treat each edge in the graph as an item and to associate weights and values to them, so that sparsification strategies can be appropriately defined.

When sparsifying a graph, we would like to preserve structural or statistical properties of the original graph as much as possible, but reduce the number of edges. We assume that the vertex set V remains the same. Let $\tilde{G}(V, \tilde{E})$ denote the graph after sparsification. We would like to minimize the loss of information, modeled by an appropriate loss function $\mathcal{L}(G, \tilde{G})$. Since the set of vertices is unchanged, we may write $\min \mathcal{L}(G, \tilde{G})$ as $\max f(\tilde{E})$ for some function f. User-defined edge weights or costs $\{c_e \mid e \in E\}$ provide fine-grained control over whether an edge should be included or not. A user-specified knapsack cost upper bound W can also be set appropriately, in the same units as c, to control the sparsity of the simplified network. When f is a linear function, i.e., associating a set of profits or values $\{p_e \mid e \in E\}$ with edges, graph sparsification can be written as a linear knapsack problem:

$$\max_{\tilde{E}\subseteq E} \sum_{e\in\tilde{E}} p_e \;\text{ subject to }\; \sum_{e\in\tilde{E}} c_e \le W. \tag{GSK}$$

A natural way to use this problem formulation would be to set profits to computed global or local edge centrality values. Edge centrality values attempt to quantitatively rank edges. Sparsification using the above knapsack formulation would then mean filtering low centrality edges, while satisfying user-specified linear constraints.

This chapter makes the following contributions: In Section 8.1, we show that a number of prior edge sparsification methods can be expressed as special cases of the problem (GSK). We implement a fast solution strategy for this problem, discussed in Section 8.2. In Section 8.3, we perform a preliminary exploration of the design space enabled by our knapsack formulation, by encoding pairs of complementary edge centrality values as profits and costs. We also empirically evaluate these sparsification strategies and the performance of our solver on several test graph instances in Section 8.4. Graph sparsification can be used for identifying important routes on transportation networks, and eliminating roads with little traffics for informative visualization. Work performed in this chapter leads to the paper in [190].

8.1 Transforming Prior Methods to GSK

Graph edge sparsification is performed for a variety of reasons. The motivations for each of the following strategies are briefly discussed here. I then describe how they can be recast as instances of our proposed knapsack-based problem.

8.1.1 Backbone Extraction.

A large collection of relational data sets can be modeled as weighted graphs. Backbone extraction refers to the problem of determining a reduced representation of the original network with fewer edges. The backbone highlights the structure of the network and preserves key characteristics. Serrano et al. [191] propose a method to extract backbones from weighted graphs. First, edge weights are normalized locally. Let z_{ij} denote the weight of edge $\langle i, j\rangle$. The weight is changed to $w_{ij} = z_{ij}/\sum_{\{k|\langle i,k\rangle\in E\}} z_{ik}$. The denominator is the sum of the weights of all edges connected to vertex i. In case of undirected graphs, each edge is considered twice, and weights are normalized according to vertices i and j separately. In the next step, a statistical test against a null model of a normalized edge weight distribution is used to determine edges that are statistically significant. The null model assumes that normalized edge weights of a degree-k vertex are proportional to random assignments of k sub-intervals from $[0, 1]$. Under this null model, the probability of observing an edge with normalized weight of at least w_{ij} is $\alpha_{ij} \overset{\text{def}}{=} 1 - (k-1) \int_0^{w_{ij}} (1-x)^{k-2} dx$. A local significance level α is selected, and if $\alpha_{ij} < \alpha$, the observed edge is considered to be statistically significant and retained. Other edges are filtered.

Foti et al. [192] consider a related approach using a non-parametric statistical test. The same weight normalization step is performed, but the statistical test is replaced by $\alpha_{ij} = 1 - \frac{1}{d_i} \sum_{\{k | \langle i,k \rangle \in E\}} \mathbf{1}\{w_{ik} \leq w_{ij}\}$. Here, $\mathbf{1}\{\cdot\}$ denotes the indicator function, which equals 1 if the expression in the bracket is true, and 0 otherwise. d_i denotes the degree of vertex i. α_{ij} is considered to be the probability of empirically observing an edge with normalized weight at least w_{ij} locally at vertex i. These two backbone extraction methods can be viewed as a special case of the knapsack problem:

$$\max_{\tilde{E} \subseteq E} \sum_{i \in V} \sum_{\langle i,j \rangle \in \tilde{E}} (\alpha - \alpha_{ij}).$$

The above problem can be considered a Lagrangian relaxation of problem (GSK), and with no constraints. This problem can be easily solved by selecting edges with positive profits, i.e., $\alpha - \alpha_{ij} > 0$.

8.1.2 Similarity Filters for Clustering.

Satuluri et al. [193] use edge sparsification as a preprocessing step to enhance scalability of graph vertex clustering algorithms. The goal is not to sacrifice the quality of clustering results with the sparsified graph. Each edge $\langle i, j \rangle$ is evaluated based on the similarity of vertices i and j. Similarity is defined using the Jaccard score:

$$\texttt{Jac}_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|},$$

where $N(i)$ refers to the set of neighboring vertices of i. The authors propose two sparsification schemes based on the Jaccard score. The global similarity method first ranks each edge in the network using `Jac`; then the top $s\%$ edges ranked by

`Jac` are retained in the sparsified network. Therefore, the global similarity method could be viewed as problem (GSK) with unit edge weights, i.e., a edge cardinality constraint:

$$\max_{\tilde{E} \subseteq E} \sum_{e \in \tilde{E}} \texttt{Jac}_e \text{ subject to } |\tilde{E}| \leq W.$$

In addition to filtering edges globally, the authors also consider ranking edges locally at each vertex similar to [191] and [192]. Starting with a network of the same vertex set and an empty edge set, for each vertex $i \in V$, the top $\lceil d_i^\alpha \rceil$ (exponent $\alpha \in (0, 1)$) of the incident edges in the original network ranked by similarity score $\texttt{Jac}_{ij}$ are added back. The intuition is that similar vertices are more likely to reside in the same cluster, and so intra-cluster edges are retained. This problem can be viewed as an unconstrained knapsack problem by setting the profit values of edges that are to be filtered to zero, and the profit values for the edges to be retained to their Jaccard scores. We could impose additional filtering constraints to complement this local similarity filter, as we will discuss in Section 8.3.

8.1.3 Local Degree Filter.

Lindner et al. [194] recently evaluate a degree-based sparsification strategy. Similar to the Local Similarity filter method, each edge is first rated using a score function, and then a filtering threshold is applied locally at each vertex. For each edge $\langle i, j \rangle$ incident on vertex i, the score function $\texttt{Deg}_{ij}$ chosen here is the degree of vertex j. For each vertex i, the top-ranked $\lceil d_i^\alpha \rceil$ of attached edges are retained. The motivation behind using vertex degree for filtering is to preserve key graph vertices, or hub vertices, after sparsification. Analogous to the Local Similarity method, this method can be written as a special unconstrained case of problem (GSK).

8.1.4 Other Related Sparsification Methods.

Graph sparsification has been studied in several different contexts. Cut sparsifiers [195, 196] aim at creating a graph with the same set of vertices, but with fewer edges, such that every cut on the graph is preserved up to a multiplicative factor. Spectral sparsifiers by Spielman and Teng [197] is another well-known line of work. The goal here is to preserve the spectral properties of the graph Laplacian. Bonchi et al. [198] define the problem of activity-preserving graph simplification. Their approach requires additional information on activity traces in the network. The goal is to preserve observed activity traces while simplifying the graph.

A recent paper by Wilder and Sukthankar [199] formulates a sparsification method for social networks which aims to preserve the stationary distribution of random walks. Their method can be expressed as a nonlinear optimization problem with linear knapsack constraints [200, 201]:

$$\max_{\tilde{E} \subseteq E} f(\tilde{E}) \quad \text{subject to} \quad \sum_{e \in \tilde{E}} c_e \leq W. \tag{GSK-NL}$$

Optimization problems of the form (GSK-NL) have been studied and applied to other problems related to network analysis. Leskovec et al. [202] studied the problem of selecting a subset of vertices in a network for sensor placement under budget constraints. This application could also be viewed as a vertex-filtering based graph sparsification problem.

8.2 A Fast Approximation Scheme

In this section, we discuss our implementation of a fast approximation algorithm for the 0/1 knapsack problem. Note that the sparsification methods discussed in the prior section – backbone extraction [191, 203], Local Similarity Filter [193], and Local Degree Filter [194] – are straightforward to implement and do not need to be solved under the knapsack formulation. We want to create a general sparsification framework that permits global or local edge centrality-based filtering, with the flexibility of introducing user-defined linear constraints.

Given integer profits and weights, the problem (GSK) can be solved exactly in pseudopolynomial time using a dynamic programming-based scheme [204–206]. However, this approach is prohibitively expensive for large graphs, and in our sparsification scenarios, it is not guaranteed that profits and weights are integers. Approximation algorithms for the 0/1 knapsack problem and variants have been studied extensively [207–214]. There are two main kinds of approximation algorithms for knapsack: fully-polynomial time approximation schemes (FPTAS) that can achieve a result that is $1 - \epsilon$ of the optimal value for all input error tolerance values ϵ [207, 211–213]; linear-time greedy approximation algorithms that have approximation quality guarantees as a fixed percentage of the optimal value [208–210].

The complexity of FPTAS algorithms is independent of the knapsack cost bound W, but polynomial in the number of items. In this section, we apply a greedy approximation algorithm based on [202, 208, 214] and explain how it can be implemented in $O(|E|)$ time. We assume that the profits $\{p_e|e \in E\}$ and the costs $\{c_e|e \in E\}$ are precomputed.

The commonly-used greedy approximation method for the knapsack problem [208, 210, 214] starts with an empty set $E_0^{\mathcal{F}}$, and expands $E_k^{\mathcal{F}}$ to $E_{k+1}^{\mathcal{F}} = E_k^{\mathcal{F}} \cup \{e_k^{\mathcal{F}}\}$ by adding the edge $e_k^{\mathcal{F}}$ that gives the maximal "efficiency",

$$e_k^{\mathcal{F}} = \max_{e \in E \setminus E_k^{\mathcal{F}}} \frac{p_e}{c_e},$$

until reaching the $k^{\mathcal{F}}$ such that $\sum_{i=0}^{k^{\mathcal{F}}-1} c_{e_i^{\mathcal{F}}} \leq W < \sum_{i=0}^{k^{\mathcal{F}}} c_{e_i^{\mathcal{F}}}$. Selecting $\cup_{i=0}^{k^{\mathcal{F}}-1} e_i^{\mathcal{F}}$ or $\{e_{k^{\mathcal{F}}}^{\mathcal{F}}\}$ that corresponds to $\max\{\sum_{i=0}^{k^{\mathcal{F}}-1} p_{e_i^{\mathcal{F}}}, p_{e_{k^{\mathcal{F}}}^{\mathcal{F}}}\}$ gives a 2-approximation to the knapsack problem [208, 210, 214]. However, in the context of graph simplification formulation (GSK), if $p_{e_{k^{\mathcal{F}}}^{\mathcal{F}}} > \sum_{i=0}^{k^{\mathcal{F}}-1} p_{e_i^{\mathcal{F}}}$, constructing a simplified graph $\tilde{G} = (V, \{e_{k^{\mathcal{F}}}^{\mathcal{F}}\})$ with just a single edge $\{e_{k^{\mathcal{F}}}^{\mathcal{F}}\}$ may not be really usable in the context of graph sparsification. However, since the linear objective $f(\tilde{E}) = \sum_{e \in \tilde{E}} p_e$ is a submodular function with $f(E_A \cup \{e\}) - f(E_A) = f(E_B \cup \{e\}) - f(E_B) = p_e$, the greedy algorithm for submodular function maximization under knapsack constraints [202, 215] could be applied here. Analogous to expanding the edge set based on the greedy efficiency rule, the greedy profit rule initializes $E_0^{\mathcal{P}} = \emptyset$ and selects $e_k^{\mathcal{P}}$ in the k^{th} iteration that gives the largest profit:

$$e_k^{\mathcal{P}} = \max_{e \in E \setminus E_k^{\mathcal{P}}} p_e$$

until the iteration $k^{\mathcal{P}}$ such that adding $e_{k^{\mathcal{P}}}^{\mathcal{P}}$ will exceed the knapsack weight constraint in problem (GSK). Leskovec et al. [202] show that using $\max\{\sum_{i=0}^{k^{\mathcal{F}}-1} p_{e_i^{\mathcal{F}}}, \sum_{i=0}^{k^{\mathcal{P}}-1} p_{e_i^{\mathcal{P}}}\}$ results in a solution that is at least $\frac{1}{2}(1 - 1/e)$ of the optimal solution for general non-decreasing submodular functions. Hence, this result is applicable to problem (GSK) as well. The case of a single edge being retained is unlikely to occur for graphs we consider and when using edge centrality measures such as the Jaccard score.

The edge sets $\{e_i^{\mathcal{F}}\}_{i=0}^{k^{\mathcal{F}}-1}$ and $\{e_i^{\mathcal{P}}\}_{i=0}^{k^{\mathcal{P}}-1}$ can be determined by sorting $\{p_e\}_{e \in E}$ and $\{p_e/c_e\}_{e \in E}$. A faster method that avoids sorting is to use the weighted medians.Given a list of profits $\{p_e | e \in E\}$ and nonnegative costs $\{c_e | e \in E\}$, define the *weighted median profit* to be the profit $p^* \in \{p_e | e \in E\}$ such that

$$\sum_{e:p_e > p^*} c_e < W \leq \sum_{e:p_e \geq p^*} c_e.$$

Similarly, let $f_e = p_e/c_e$, and define the *weighted median efficiency* to be the $f^* \in \{f_e | e \in E\}$ satisfying

$$\sum_{e:f_e > f^*} c_e < W \leq \sum_{e:f_e \geq f^*} c_e$$

Using a slight modification of the selection sort algorithm, we can find the weighted medians p^* and f^* in $O(|E|)$ time [206, 214, 216]. After obtaining the weighted medians p^* and f^*, the set $\{e_i^{\mathcal{F}}\}_{i=0}^{k^{\mathcal{F}}-1}$ and $\{e_i^{\mathcal{P}}\}_{i=0}^{k^{\mathcal{P}}-1}$ of selected edges can be decided in two passes over the set of all edges. We describe this procedure for the greedy profit strategy. Let $E^{\mathcal{P}}$ denote the set of edges selected by the greedy profit rule. In one pass over E, we partition the edges into $\{e \mid p_e > p^*\}$, $\{e \mid p_e = p^*\}$ and $\{e \mid p_e < p^*\}$. All edges in $\{e \mid p_e > p^*\}$ can be added safely to $E^{\mathcal{P}}$ and all edges in $\{e \mid p_e < p^*\}$ can be disregarded by the greedy rule. During the first pass, the remainder weight $R = W - \sum_{e:p_e > p^*} c_e$ is computed. The second pass over $\{e \mid p_e = p^*\}$ can be performed in arbitrary order until the remainder weight R is exhausted. Therefore, the running time is $O(|E|)$. The greedy efficiency rule can be implemented in a similar manner. The pseudocode for the full solver is provided in Algorithm 11.

8.3 Constructing New Sparsification Schemes

We now discuss approaches to set the edge profits $\{p_e \mid e \in E\}$ and costs $\{c_e \mid e \in E\}$ in the GSK formulation. We have two goals in mind: First, the profits should attempt to preserve a structural property. Second, we want to reuse precomputed vertex/edge centrality scores or other information available apriori in order to set the edge costs. For example, in case of online social network analysis, suppose the PageRank scores for each vertex have been already computed. How can we use this information to sparsify the graph and perform additional network analysis tasks? Ideally, we might wish that after sparsification, the set of top-ranked vertices using

PageRank remains the same, and additionally other properties such as clustering coefficient distribution or community structure are also preserved. More generally, given a set of precomputed vertex centrality scores $\{s(u) \mid u \in V\}$, we devise a heuristic to construct a set of edge scores $\{s(u, v) \mid u \in V, \langle u, v\rangle \in E\}$. If our primary interest is to preserve the centrality rankings, then the edge scores could be used directly as profits. Otherwise, the edge scores could be transformed into costs, which would discriminate the edges based on their contribution to the vertex centralities in the original network.

We propose using a simple method for constructing edge scores that works on both directed and undirected graphs. For directed graphs, on each vertex $u \in V$, the centrality score $s(u)$ is distributed among its outgoing (or incoming) edges $\{\langle u, v\rangle \in E\}$. Each outgoing edge $e = \langle u, v\rangle$ obtains a fraction of $s(u)$, proportional to $d(v)/\sum_{\langle u,z\rangle \in E} d(z)$. The intuition for distributing the centrality scores on outgoing edges is that the influence of a vertex is more likely to propagate along its high-degree neighbors. For undirected graphs, we duplicate each edge $e \in E$ to create two directed links $\langle u, v\rangle$ and $\langle v, u\rangle$. This operation enlarges size of the knapsack decision variables from $|E|$ to $2|E|$. After solving problems (GSK), an edge e is retained in the simplified network if at least one of $\langle u, v\rangle$ and $\langle v, u\rangle$ is selected in the knapsack solution. We summarize this process of assigning edge scores in Algorithm 12.

Let $\mathtt{DPR}_{ij}$ denote the score on edge $\langle i, j\rangle$ when Algorithm 12 is applied to generate edge scores based on PageRank. Since $\mathtt{DPR}_{ij} \in (0, 1]$, $1 - \mathtt{DPR}_{ij}$ could be used as edge costs, so that a weak edge in the sense of contribution to PageRank has a large cost in problem (GSK). We extend the Jaccard score-based Local Similarity method and Global Similarity methods of Satuluri et al. [193] to use $(1 - \mathtt{DPR}_{ij})$ as edge costs, and enforcing constraints

$$\sum_{i \in V} \sum_{\langle i,j\rangle \in \tilde{E}} (1 - \mathtt{DPR}_{ij}) \leq W,$$

where W is a user-supplied parameter. W is set to be a percentage of $2|E|$, since $\sum_{i \in V} \sum_{\langle i,j\rangle \in E} (1 - \mathtt{DPR}_{ij}) = 2|E| - 1$. We also devise a new method that uses $\mathtt{DPR}_{ij}$ as the profit for edge $\langle i, j\rangle$, and enforces the unit-cost constraint $\sum_{i \in V} \sum_{\langle i,j\rangle \in \tilde{E}} 1 \leq W$. The motivation of this method is to preserve the vertices with top PageRank scores after sparsification.

Table 8.1. Graphs used in empirical evaluation.

Name	$\lvert V\rvert$ ($\times 10^6$)	$\lvert E\rvert$ ($\times 10^6$)
`com-Amazon` [217]	0.335	0.9
`com-DBLP` [217]	0.32	1.05
`com-Youtube` [217]	1.13	3
`roadNet-CA` [218]	1.97	5.53
`as-Skitter` [218]	1.7	11
`com-LiveJournal` [217]	4	34.68
`com-Orkut` [217]	3.07	117

Table 8.2. Problem (GSK)-based graph sparsification strategies used in empirical evaluation.

Label	$p(i,j)$	*Constraint*	*Motivation*
JLPR	$\frac{1}{d_i}\sum_{\{k\mid\langle i,k\rangle\in\tilde{E}\}}\mathbf{1}\{\mathtt{Jac}_{ik}\leq\mathtt{Jac}_{ij}\}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}(1-\mathtt{DPR}_{ij})\leq W$	local similarity +PageRank
JGPR	$\mathtt{Jac}_{ij}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}(1-\mathtt{DPR}_{ij})\leq W$	similarity +PageRank
PR	$\mathtt{DPR}_{ij}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}1\leq W$	PageRank
JG	$\mathtt{Jac}_{ij}$	$\sum_{i\in V}\sum_{\langle i,j\rangle\in\tilde{E}}1\leq W$	similarity [193]

8.4 Empirical Evaluation

In this section, we empirically evaluate six sparsification methods that can be expressed in the form of either problem (GSK) or its simpler unconstrained version. We used seven sparse graphs from the Stanford large network dataset collection SNAP [219], listed in Table 8.1. We implemented the approximate knapsack solution scheme in C+ and verified correctness with several test instances. The six sparsification methods considered are summarized in Tables 8.2 and 8.3. All

Table 8.3. Unconstrained sparsification strategies used in empirical evaluation.

Label	$p(i,j)$	*Motivation*
DL	retain top $\lceil d_i^\alpha\rceil$ edges ranked by $\mathtt{Deg}_{ij}$ locally	hub vertices [194]
JL	retain top $\lceil d_i^\alpha\rceil$ edges ranked by $\mathtt{Jac}_{ij}$ locally	local similarity [193]

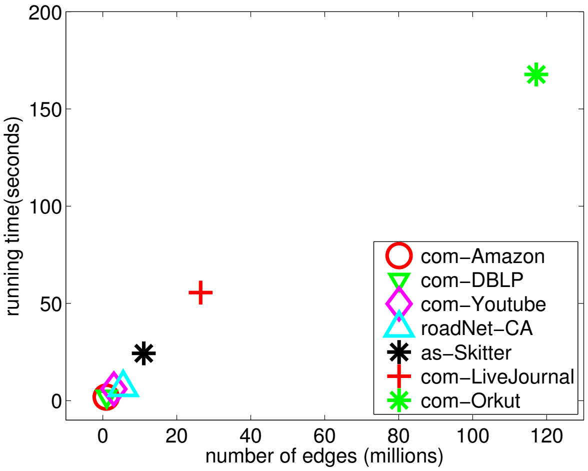


Figure 8.1. Solver running time on various graphs.

experiments were run on a single server of Cyberstar, a Penn State compute cluster. The server we run our programs on is a dual-socket quad-core Intel Nehalem system (Intel Xeon X5550 processor) with 32 GB main memory.

8.4.1 Solver Performance.

We evaluate performance of the solver for the problem (GSK). Unlike the exact pseudopolynomial dynamic programming approach, the running time of Algorithm 11 is independent of the user-supplied weight parameter W and the actual settings for the edge profits $\{p_e | e \in E\}$ and costs $\{c_e | e \in E\}$. We computed the execution time of the solver for the seven graphs in Table 8.1. The graph sizes differ by nearly two orders of magnitude, going from `com-Amazon` to `com-Orkut`. Figure 8.1 gives the running times. As the number of graph edges increases, the execution time appears to increase linearly. This empirical evaluation is one way of demonstrating the practical efficacy of the solver. Note that this figure only gives the solver execution time, and does not include the time taken to compute profits and costs. Our current implementations for PageRank and Jaccard scores are straightforward and untuned. For `com-DBLP`, the running times of JLPR for PageRank computation (costs), Jaccard score computation (profits), and the greedy approximation are 0.21, 2.28, and 2.1 seconds, respectively.

8.4.2 Comparing Sparsification Methods.

We now apply the solver to the six problem formulations in Tables 8.2 and 8.3. In addition, we sparsify the graphs using uniform random edge (RE) sampling as a baseline. The abbreviations used in the figures correspond to the methods in Tables 8.2 and 8.3. The knapsack bound W is user-supplied. We set the problem parameters to achieve various sparsification ratios of $|\tilde{E}|/|E|$. We use three structural properties to evaluate the sparsification schemes: vertex degree distribution, top-ranked PageRank vertices and average clustering coefficient. We use Spearman's rank correlation coefficient [220] to compute correlation between vertex degree rankings in the sparsified graph and in the original graph. This value will be close to 1 if the degree rankings are highly correlated. For PageRank, we evaluate the methods based on the proportion of the top 10% PageRank-ordered vertices in the original graph that are preserved after sparsification. We report experimental results on `com-Amazon`, `com-DBLP`, `com-Youtube` and `com-LiveJournal`.

Figure 8.2 plots the Spearman's correlation for vertex degree rankings at various sparsification ratios. There is no single method that consistently outperforms others. In [194], the authors perform a similar experiment, for vertex degree ranking using Spearman's rank correlation coefficient, over a large collection of social networks. Their results show that DL and RE consistently outperform JL. In our experiments, RE, PR, DL, and JLPR all perform qualitatively similarly on the social network `com-LiveJournal`. However, the performance diverges on the other graphs. PR's performance is also noteworthy: it performs significantly worse than RE for `com-Amazon` and `com-DBLP`, but better than all methods for `com-Youtube`.

Results showing the overlap ratio of top 10% PageRank-ordered vertices after applying a sparsification method are shown in Figure 8.3. As expected, PR consistently outperforms the other methods, since in PR, the edge profit `DPR` is based on the vertex PageRank scores. DL and JLPR give similar results for the overlap ratio in all four graphs, and slightly underperform PR. JGPR sparsification significantly distorts the top 10% PageRank-ordered vertex set in all experiments.

The comparison of JL, JLPR, JG, and JGPR on the Spearman's correlation of degrees and the overlap ratio of top 10% PageRank vertices are displayed in Figure 8.4. For all the graphs, the difference between JL and JLPR, and the difference between JG and JGPR is very small. Hence we do not show JL and JG

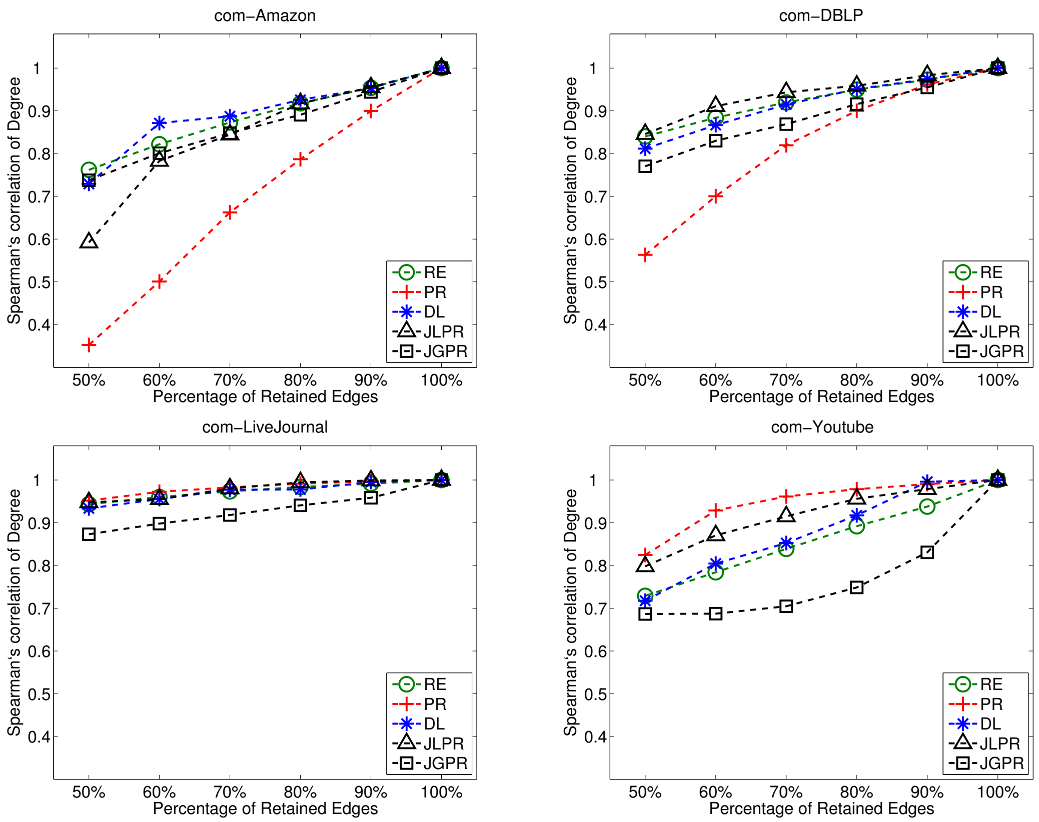


Figure 8.2. Spearman's rank coefficient of degree for the original graph and sparsified graph, at various sparsification thresholds.

results in Figures 8.2 and 8.3.

We report the deviation of average clustering coefficient Ave-CC from the original graph after sparsification on `com-DBLP` and `com-Amazon` in Figure 8.5. The deviation is computed as Ave-CC($\tilde{G}$) − Ave-CC(G). The average clustering coefficients for `com-DBLP` and `com-Amazon` are 0.6324 and 0.3967, respectively. These two graphs have the largest average clustering coefficient among the ones listed in Table 8.1. Surprisingly, PR causes the least deviation of average clustering coefficient on `com-DBLP`. JLPR and JGPR, which aim to preserve intra-cluster edges, produce positive deviations at some sparsification thresholds on `com-Amazon`. RE reduces the average clustering coefficient linearly on decreasing the number of retained edges, an observation also pointed out in [194].

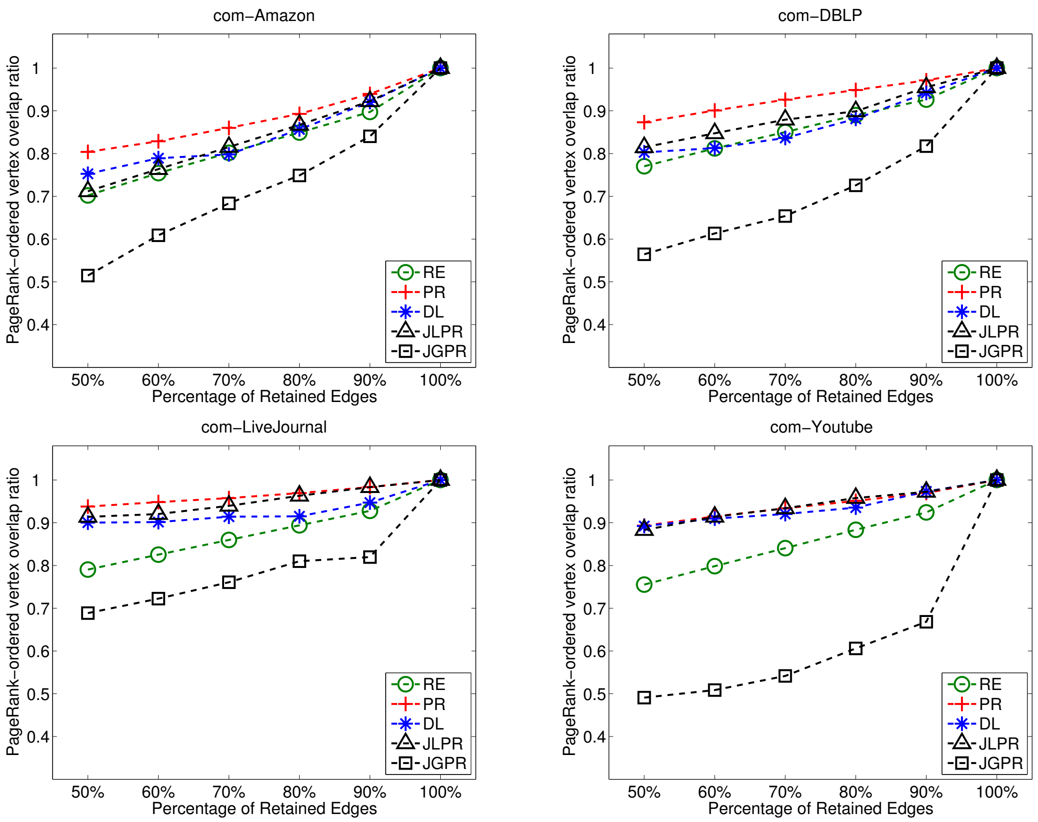


Figure 8.3. The fraction of vertices that overlap in the top 10% PageRank-ordered vertex sets in the original and the sparsified graph, at various sparsification thresholds.

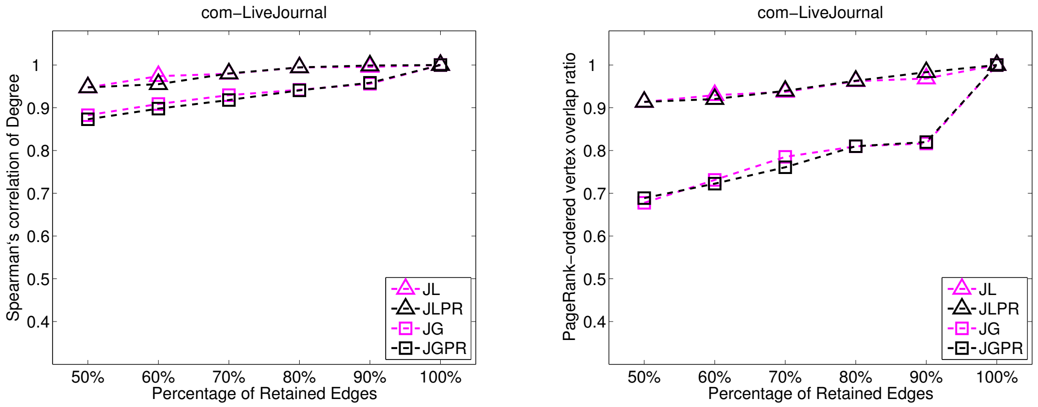


Figure 8.4. Comparing Jaccard similarity-based sparsifier performance on the `com-LiveJournal` graph.

8.5 Chapter Conclusion

In this work, I explore using the knapsack problem for graph edge sparsification. Sparsifying large graphs aids in graph visualization and serves as a speedup technique

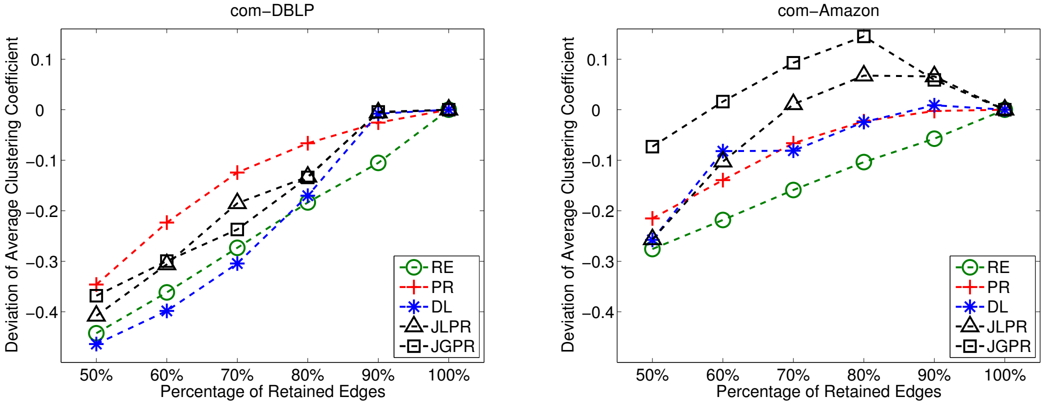


Figure 8.5. Deviation of average clustering coefficient at various sparsification thresholds.

for graph computations such as community detection and centrality analysis. The proposed knapsack-based sparsification permits both global and local centrality-based edge filtering, and additionally lets us specify fine-grained linear constraints. A greedy linear-time approximation solution scheme is implemented for this problem. This sparsification scheme may be used for scalable graph visualization and as a general speedup strategy in the future.

Algorithm 11 Pseudocode for the GSK problem solver.

Input: $G(V, E)$, profits $\{p_e | e \in E\}$, costs $\{c_e | e \in E\}$, weight constraint W.
Output: $\tilde{E}$

```
$E^{\mathcal{P}} \leftarrow \emptyset$
$S(\mathcal{P}) \leftarrow 0$                                ▷ greedy profit rule solution value
$R(\mathcal{P}) \leftarrow W$                                ▷ remainder weight
Set $p^*$ to computed weighted median profit
Initialize array $T(\mathcal{P})$
for $e \in E$ do                                             ▷ $O(|E|)$
    if $p_e > p^*$ then
        add $e$ to $E^{\mathcal{P}}$
        $S(\mathcal{P}) \leftarrow S(\mathcal{P}) + p_e$
        $R(\mathcal{P}) \leftarrow R(\mathcal{P}) - c_e$
    else if $p_e = p^*$ then
        add $e$ to $T(\mathcal{P})$
    end if
end for
add $e \in T(\mathcal{P})$ to $E^{\mathcal{P}}$ until $R(\mathcal{P})$ is exhausted
for $e \in E$ do
    Compute efficiency $f_e \leftarrow p_e / c_e$
end for
$E^{\mathcal{F}} \leftarrow \emptyset$
$S(\mathcal{F}) \leftarrow 0$                                ▷ greedy efficiency rule solution value
$R(\mathcal{F}) \leftarrow W$                                ▷ remainder weight
Set $f^*$ to computed weighted median efficiency
Initialize array $T(\mathcal{F})$
for $e \in E$ do                                             ▷ $O(|E|)$
    if $f_e > f^*$ then
        add $e$ to $E^{\mathcal{F}}$
        $S(\mathcal{F}) \leftarrow S(\mathcal{F}) + p_e$
        $R(\mathcal{F}) \leftarrow R(\mathcal{F}) - c_e$
    else if $f_e = f^*$ then
        add $e$ to $T(\mathcal{F})$
    end if
end for
add $e \in T(\mathcal{F})$ to $E^{\mathcal{F}}$ until $R(\mathcal{F})$ is exhausted
if $S(\mathcal{P}) \geq S(\mathcal{F})$ then
    return $E^{\mathcal{P}}$
else
    return $E^{\mathcal{F}}$
end if
```

Algorithm 12 `DistributeCentrality`

```
Input: network $G(V, E)$, centrality $\{s(u) | u \in V\}$
Output: edge scores $\{s(u, v) | u \in V, \langle u, v\rangle \in E\}$
if $G$ is undirected then
    for $e \in E$ do
        $u, v \leftarrow$ end points of $e$
        create directional edges $\langle u, v\rangle$ and $\langle v, u\rangle$
    end for
end if
for $u \in V$ do
    for $\forall\langle u, v\rangle$ attached to $u$ do
        $s(u, v) \leftarrow \Big(d(v) / \sum_{\langle u,z\rangle \in E} d(z)\Big) s(u)$
    end for
end for
```

Chapter 9
Conclusions and Future Work

9.1 Summary of Contributions

This dissertation studies temporal prediction problems on traffic flow data and machine learning with structural knowledge for accident factor analysis. Optimization is an recurring theme throughout the development of different methods. We focus on the robustness, multi-modality, and non-stationary aspects in the development of learning algorithms for traffic flow prediction. Many existent methods have been proposed in the literature for traffic flow prediction [49, 51, 221–224] (additionally, see [225] and references therein). To avoid the selection of a particular method at the price of discarding others, a consensus ensemble learning approach which robustly combines the result from different models was developed in Chapter 3. The ensemble learning scheme utilizes a covariance-regularizer to balance the trade-offs between the predictive accuracy and mutual dependency among different models. Computational efficiency and non-stationarity is viewed as an integrated problem in Chapter 4. In the presence of non-stationary observations, computationally efficient model adaptation to data is an alternative strategy to statistical removal of the non-stationary trends. An online hyperparameter optimization algorithm was proposed in Chapter 4. Theoretical guarantees based on regret minimization framework [27–30] was provided. This algorithm enables adaptive hyperparameter tuning in data streaming environment and much outperforms models with statically pre-selected hyperparameters.

We took a structural learning approach to model the latent heterogeneity in traffic accidents and assist the analysis of factors influencing the severity of crashes.

Structural learning refers to generate hypothesis from data with rich internal structure [11]. Often, we have a broad prior knowledge about the structure of latent heterogeneity among the observations. For example, by the Central Limit Theorem, it is natural to assume that individual differences with respect to the degree of injury given some accident factors follow a Gaussian distribution. The parameters of the latent distribution is unknown and will be inferred from data via an optimization formulation. This approach is known as the mixed logit modeling [12–18] in transportation literature. A scalable stochastic optimization algorithm was developed in Chapter 5 to solve the mixed logit parameter estimation problem. On the other hand, relaxing the distribution assumptions on the individual heterogeneity improves the robustness, from both modeling and optimization perspective. In many problems, researchers have found that intrinsic low dimension structure exists despite that the observed data is noisy and possibly with missing values [38–42] (and references therein). It is not surprise if the latent heterogeneous effect resides in low-rank space. Therefore, a latent effect logistic regression model was developed in Chapter 6 requiring no distributional priors on the latent parameters, only assuming the existence of low-rank structure. Chapter 7 and Chapter 8 introduced pre-processing techniques for exploratory analysis on network data.

9.2 Future Directions

9.2.1 Hyperparameter optimization for non-smooth problems

In Chapter 4, hyperparameter optimization is formulated as a bi-level programming problem [3–8]. The analytical hyper-gradient is available in the case of kernel models. In general, analytical hyper-gradient is not available. When the nested problem, i.e., the training objective function in the model, is smooth, the gradient with respect to the hyperparameters can be computed via implicit differentiation. However, implicit differentiation techniques required the computation of second derivative matrices. An interesting extension is to use approximate Hessian matrices in the computation of hyper-gradient. For example, applying the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [34,35,37] formula or Barzilai-Borwein (BB) [226] method. Implicit differentiation cannot be applied directly if the inner problem is non-smooth. Another extension is developing hyperparameter optimization method with smoothing techniques [88] for the inner problem.

9.2.2 Theoretical investigations of the latent effect model

In Chapter 6, a convex formulation for logistic regression with heterogeneous latent effect was proposed. This model is constructed based on the following ideas: modeling heterogeneity without losing the convexity of objective function, separation of sparse common effect from low-rank heterogeneous effect. Chapter 6 focused on the model proposal and the computational aspects. Theoretical investigations on the model and analysis on its connections to other low-rank and sparse decompositions [38–42] is important. In addition, empirical risk minimization setup with other loss functions is a natural extension to the latent logistic regression model. The theoretical study should consider the statistical properties under general convex loss functions.

9.2.3 Latent graphical inference for mixed logit

The latent effect logistic regression via low-rank and sparse decomposition described in Chapter 6 relaxes the distributional assumptions on the heterogeneous effects. Sometimes, practitioners do want to impose distributional restrictions to assist the interpretation of results. However, as we shown in Chapter 5, Gaussian mixing distribution with Maximum Simulated Likelihood estimation leads to a non-convex problem and can be unstable. A possible direction to perform convex optimization with mixing distributions is to follow a two stage strategy. In the first stage, latent effect for each observation is inferred using the low-rank and sparse decomposition-based logistic model described in Chapter 6. In the second stage, once Gaussian assumptions are placed on the latent effects, we can perform convex Gaussian graphical model inference [227–230] given the heterogeneous effects obtained from the first step.

Bibliography

[1] IEEE Transactions on Intelligent Transportation Systems Editorial Board, "Aim and Scope," `https://ieeexplore.ieee.org/xpl/aboutJournal.jsp?punumber=6979#AimsScope`, accessed: 2018-09-30.

[2] United States Department of Transportation, "ITS Research Fact Sheets - Benefits of Intelligent Transportation Systems," `https://www.its.dot.gov/factsheets/benefits_factsheet.htm`, accessed: 2018-09-30.

[3] Maclaurin, D., D. Duvenaud, and R. Adams (2015) "Gradient-based hyperparameter optimization through reversible learning," in *Proc. ICML*.

[4] Pedregosa, F. (2016) "Hyperparameter optimization with approximate gradient," in *Proceedings of the 33rd International Conference on Machine Learning*, pp. 737–746.

[5] Luketina, J., M. Berglund, K. Greff, and T. Raiko (2016) "Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, pp. 2952–2960.

[6] Franceschi, L., M. Donini, P. Frasconi, and M. Pontil (2017) "Forward and Reverse Gradient-Based Hyperparameter Optimization," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, PMLR, International Convention Centre, Sydney, Australia, pp. 1165–1173.

[7] Franceschi, L., P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil (2018) "Bilevel Programming for Hyperparameter Optimization and Meta-Learning," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, PMLR, StockholmsmÃďssan, Stockholm Sweden, pp. 1568–1577.

[8] Zhan, H., G. Gomes, X. S. Li, K. Madduri, and K. Wu (2018) "Efficient Online Hyperparameter Learning for Traffic Flow Prediction," in *2018 IEEE 21th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, pp. 1–6.

[9] Schölkopf, B., R. Herbrich, and A. J. Smola (2001) "A generalized representer theorem," in *Proceeding of the International Conference on Computational Learning Theory (COLT)*.

[10] Schlkopf, B., A. J. Smola, and F. Bach (2018) "Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond," .

[11] Dietterich, T. G., P. Domingos, L. Getoor, S. Muggleton, and P. Tadepalli (2008) "Structured machine learning: the next ten years," *Machine Learning*, **73**(1), p. 3.

[12] McFadden, D. (1989) "A method of simulated moments for estimation of discrete response models without numerical integration," *Econometrica: Journal of the Econometric Society*, pp. 995–1026.

[13] Train, K. (2001) "A comparison of hierarchical Bayes and maximum simulated likelihood for mixed logit," *University of California, Berkeley*, pp. 1–13.

[14] Hensher, D. A. and W. H. Greene (2003) "The mixed logit model: the state of practice," *Transportation*, **30**(2), pp. 133–176.

[15] Hole, A. R. (2007) "Estimating mixed logit models using maximum simulated likelihood," *Stata Journal*, **7**(3), pp. 288–401.

[16] Train, K. E. (2009) *Discrete choice methods with Simulation*, 2 ed., Cambridge University Press.

[17] Park, S. and S. Gupta (2012) "Comparison of SML and GMM estimators for the random coefficient logit model using aggregate data," *Empirical Economics*, **43**(3), pp. 1353–1372.

[18] Train, K. (2016) "Mixed logit with a flexible mixing distribution," *Journal of choice modelling*, **19**, pp. 40–53.

[19] Bottou, L. (2010) "Large-Scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Springer, pp. 177–186.

[20] Moulines, E. and F. R. Bach (2011) "Non-asymptotic analysis of stochastic approximation algorithms for machine learning," in *Advances in Neural Information Processing Systems*, pp. 451–459.

[21] Sra, S., S. Nowozin, and S. J. Wright (2012) *Optimization for machine learning*, Mit Press.

[22] Ghadimi, S. and G. Lan (2013) "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, **23**(4), pp. 2341–2368.

[23] Johnson, R. and T. Zhang (2013) "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in neural information processing systems*, pp. 315–323.

[24] Schmidt, M., N. Le Roux, and F. Bach (2017) "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, **162**(1-2), pp. 83–112.

[25] Gourieroux, C. and A. Monfort (1990) "Simulation based inference in models with heterogeneity," *Annales d'Economie et de Statistique*, pp. 69–107.

[26] Bergstra, J. and Y. Bengio (2012) "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, **13**(Feb), pp. 281–305.

[27] Zinkevich, M. (2003) "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. ICML.*

[28] Cesa-Bianchi, N. and G. Lugosi (2006) *Prediction, learning, and games*, Cambridge university press.

[29] Hazan, E. (2012) "The convex optimization approach to regret minimization," in *Optimization for Machine Learning* (S. Sra, S. Nowozin, and S. J. Wright, eds.), chap. 10, MIT Press, pp. 287–304.

[30] Hazan, E., K. Singh, and C. Zhang (2017) "Efficient Regret Minimization in Non-Convex Games," in *Proceedings of the 34rd International Conference on Machine Learning.*

[31] Lee, L.-F. (1992) "On efficiency of methods of simulated moments and maximum simulated likelihood estimation of discrete response models," *Econometric Theory*, **8**(4), pp. 518–552.

[32] Hajivassiliou, V. A. and P. A. Ruud (1994) "Classical estimation methods for LDV models using simulation," *Handbook of econometrics*, **4**, pp. 2383–2441.

[33] Goldfarb, D. (1970) "A family of variable-metric methods derived by variational means," *Mathematics of computation*, **24**(109), pp. 23–26.

[34] Shanno, D. F. (1970) "Conditioning of quasi-Newton methods for function minimization," *Mathematics of computation*, **24**(111), pp. 647–656.

[35] Broyden, C. G. (1970) "The convergence of a class of double-rank minimization algorithms 1. general considerations," *IMA Journal of Applied Mathematics*, **6**(1), pp. 76–90.

[36] Fletcher, R. (1970) "A new approach to variable metric algorithms," *The computer journal*, **13**(3), pp. 317–322.

[37] Nocedal, J. and S. Wright (2006) *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering.

[38] Candès, E. J. and B. Recht (2009) "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, **9**(6), p. 717.

[39] Candès, E. J., X. Li, Y. Ma, and J. Wright (2011) "Robust principal component analysis?" *Journal of the ACM (JACM)*, **58**(3), p. 11.

[40] Soltanolkotabi, M., E. Elhamifar, E. J. Candes, et al. (2014) "Robust subspace clustering," *The Annals of Statistics*, **42**(2), pp. 669–699.

[41] Aybat, N. S., D. Goldfarb, and S. Ma (2014) "Efficient algorithms for robust and stable principal component pursuit problems," *Computational Optimization and Applications*, **58**(1), pp. 1–29.

[42] Udell, M. and A. Townsend (2018) "Why are Big Data Matrices Approximately Low Rank?" *SIAM Mathematics of Data Science (SIMODS), to appear*, `1705.07474`.

[43] Beck, A. and M. Teboulle (2009) "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, **2**(1), pp. 183–202.

[44] Cai, J.-F., E. J. Candès, and Z. Shen (2010) "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, **20**(4), pp. 1956–1982.

[45] Toh, K.-C. and S. Yun (2010) "An Accelerated Proximal Gradient Algorithm for Nuclear Norm Regularized Least Squares Problems," *Pacific Journal of Optimization*, **6**.

[46] Ma, S., D. Goldfarb, and L. Chen (2011) "Fixed point and Bregman iterative methods for matrix rank minimization," *Mathematical Programming*, **128**(1-2), pp. 321–353.

[47] Jenatton, R., J. Mairal, G. Obozinski, and F. Bach (2011) “Proximal methods for hierarchical sparse coding,” *Journal of Machine Learning Research*, **12**(Jul), pp. 2297–2334.

[48] Taieb, S. B. and R. Hyndman (2014) “Boosting multi-step autoregressive forecasts,” in *International Conference on Machine Learning*, pp. 109–117.

[49] Wu, C.-J., T. Schreiter, R. Horowitz, and G. Gomes (2014) “Traffic Flow Prediction Using Optimal Autoregressive Moving Average with Exogenous Input-Based Predictors,” *Transportation Research Record: Journal of the Transportation Research Board*, **2421**, pp. 125–132.

[50] Lai, T. L. (1993) “Recursive estimation in ARMAX models,” in *New Directions in Time Series Analysis: Part II* (D. Brillinger, P. Caines, J. Geweke, E. Parzen, M. Rosenblatt, and M. S. Taqqu, eds.), pp. 263–288.

[51] Coogan, S., C. Flores, and P. Varaiya (2017) “Traffic Predictive Control from Low-Rank Structure,” *Transportation Research Part B: Methodological*, **97**, pp. 1–22.

[52] Wold, S., A. Ruhe, H. Wold, and W. J. Dunn, III (1984) “The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses,” *Journal on Scientific and Statistical Computing*, **5**(3), pp. 735–743.

[53] Drucker, H., C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik (1997) “Support vector regression machines,” in *Advances in neural information processing systems (NIPS)*, pp. 155–161.

[54] Vapnik, V., S. E. Golowich, and A. Smola (1996) “Support vector method for function approximation, regression estimation, and signal processing,” in *Advances in Neural Information Processing Systems (NIPS)*.

[55] Zhang, Y. and Y. Xie (2008) “Forecasting of short-term freeway volume with v-support vector machines,” *Transp. Res. Rec.*, **2024**, pp. 92–99.

[56] Wei, D. and H. Liu (2013) “An adaptive-margin support vector regression for short-term traffic flow forecast,” *Journal of Intelligent Transportation Systems*, **17**(4), pp. 317–327.

[57] Vapnik, V. N. (1998) *Statistical learning theory*, Wiley New York.

[58] Bishop, C. M. (2006) *Pattern Recognition and Machine Learning*, Springer-Verlag New York.

[59] Chang, C.-C. and C.-J. Lin (2011) "LIBSVM: a library for support vector machines," *Transactions on Intelligent Systems and Technology*, **2**(3), pp. 27:1–27:27.

[60] Fan, R.-E., P.-H. Chen, and C.-J. Lin (2005) "Working set selection using second order information for training support vector machines," *Journal of Machine Learning Research*, **6**, pp. 1889–1918.

[61] Xie, Y., K. Zhao, Y. Sun, and D. Chen (2010) "Gaussian processes for short-term traffic volume forecasting," *Transportation Research Record: Journal of the Transportation Research Board*, **2165**, pp. 69–78.

[62] Rasmussen, C. E. and C. K. I. Williams (2006) *Gaussian Processes for Machine Learning*, MIT Press.

[63] Zhan, H., G. Gomes, X. S. Li, K. Madduri, A. Sim, and K. Wu (2018) "Consensus Ensemble System for Traffic Flow Prediction," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12.

[64] Cormode, G., V. Shkapenyuk, D. Srivastava, and B. Xu (2009) "Forward decay: A practical time decay model for streaming systems," in *Proc. Int'l. Conf. on Data Engineering (ICDE).*

[65] Hastie, T., R. Tibshirani, and J. Friedman (2009) *The Elements of Statistical Learning*, Springer-Verlag New York.

[66] "MathWorks® Documentation, fitrgp," `https://www.mathworks.com/help/stats/fitrgp.html`, accessed: 2017-09-14.

[67] "MathWorks® Documentation, fitrsvm," `https://www.mathworks.com/help/stats/fitrsvm.html`, accessed: 2017-09-14.

[68] Wolpert, D. H. (1992) "Stacked generalization," *Neural Networks*, **5**(2), pp. 241–259.

[69] Zhou, Z.-H. (2012) *Ensemble methods: foundations and algorithms*, Chapman and Hall/CRC.

[70] Granger, C. W. J. and R. Ramanathan (1984) "Improved methods of combining forecasts," *Journal of Forecasting*, **3**(2), pp. 197–204.

[71] Smith, J. and K. F. Wallis (2009) "A simple explanation of the forecast combination puzzle," *Oxford Bulletin of Economics and Statistics*, **71**(3), pp. 331–355.

[72] Genre, V., G. Kenny, A. Meyler, and A. Timmermann (2013) "Combining expert forecasts: Can anything beat the simple average?" *International Journal of Forecasting*, **29**(1), pp. 108–121.

[73] Clemen, R. T. (1989) "Combining forecasts: A review and annotated bibliography," *International Journal of Forecasting*, **5**(4), pp. 559–583.

[74] Hou, Y., P. Edara, and C. Sun (2015) "Traffic flow forecasting for urban work zones," *IEEE Transaction on Intelligent Transportation Systems*, **16**(4), pp. 1761–1770.

[75] Sun, S. (2009) "Traffic flow forecasting based on multitask ensemble learning," in *Proc. ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC)*.

[76] Breiman, L. (1996) "Stacked regressions," *Machine Learning*, **24**(1), pp. 49–64.

[77] Li, L., X. Chen, and L. Zhang (2014) "Multimodel ensemble for freeway traffic state estimations," *IEEE Transaction on Intelligent Transportation Systems*, **15**(3), pp. 1323–1336.

[78] Tan, M.-C., S. C. Wong, J.-M. Xu, Z.-R. Guan, and P. Zhang (2009) "An aggregation approach to short-term traffic flow prediction," *IEEE Transaction on Intelligent Transportation Systems*, **10**(1), pp. 60–69.

[79] Bengio, Y. (2000) "Gradient-based optimization of hyperparameters," *Neural computation*, **12**(8), pp. 1889–1900.

[80] Seeger, M. (2007) "Cross-Validation Optimization for Large Scale Hierarchical Classification Kernel Methods," in *Advances in Neural Information Processing Systems 19* (B. Schölkopf, J. C. Platt, and T. Hoffman, eds.), MIT Press, pp. 1233–1240.

[81] Foo, C.-s., C. B. Do, and A. Y. Ng (2008) "Efficient multiple hyperparameter learning for log-linear models," in *Proceedings of the 21nd International Conference on Neural Information Processing Systems*.

[82] Bergstra, J. S., R. Bardenet, Y. Bengio, and B. Kégl (2011) "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, pp. 2546–2554.

[83] Kandasamy, K., J. Schneider, and B. Póczos (2015) "High dimensional Bayesian optimisation and bandits via additive models," in *International Conference on Machine Learning*, pp. 295–304.

[84] Klein, A., S. Falkner, S. Bartels, P. Hennig, and F. Hutter (2017) "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets," in *Artificial Intelligence and Statistics*, pp. 528–536.

[85] Jamieson, K. and A. Talwalkar (2016) "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial Intelligence and Statistics*, pp. 240–248.

[86] Li, L., K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2017) "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, **18**(1), pp. 6765–6816.

[87] Zhan, H., G. Gomes, X. S. Li, K. Madduri, and K. Wu (2018) "Efficient Online Hyperparameter Optimization for Kernel Ridge Regression with Applications to Traffic Time Series Prediction," *arXiv preprint arXiv:1811.00620*.

[88] Rockafellar, R. T. (2015) *Convex analysis*, Princeton university press.

[89] Snoek, J., H. Larochelle, and R. P. Adams (2012) "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959.

[90] MacKay, D. J. C., "Introduction to Gaussian processes," `http://www.inference.org.uk/mackay/gpB.pdf`, last accessed Oct 2017.

[91] Wang, W. and M. A. Carreira-Perpinán (2013) "Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application," *arXiv preprint arXiv:1309.1541*.

[92] Ghadimi, S., G. Lan, and H. Zhang (2016) "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization," *Mathematical Programming*, **155**(1-2), pp. 267–305.

[93] Hazan, E. (2008) "Extracting certainty from uncertainty: Regret bounded by variation in costs," in *Proceeding of the International Conference on Computational Learning Theory (COLT)*.

[94] Hazan, E. and S. Kale (2009) "On stochastic and worst-case models for investing," in *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, Curran Associates Inc., pp. 709–717.

[95] Chiang, C.-K., T. Yang, C.-J. Lee, M. Mahdavi, C.-J. Lu, R. Jin, and S. Zhu (2012) "Online optimization with gradual variations," in *Proceeding of the International Conference on Computational Learning Theory (COLT)*.

[96] Connected Corridors, "I-210 Pilot ICM Project," `http://ccdocs.berkeley.edu`, last accessed Oct 2017.

[97] Shankar, V. and F. Mannering (1996) "An exploratory multinomial logit analysis of single-vehicle motorcycle accident severity," *Journal of safety research*, **27**(3), pp. 183–194.

[98] Carson, J. and F. Mannering (2001) "The effect of ice warning signs on ice-accident frequencies and severities," *Accident Analysis & Prevention*, **33**(1), pp. 99–109.

[99] Ulfarsson, G. F. and F. L. Mannering (2004) "Differences in male and female injury severities in sport-utility vehicle, minivan, pickup and passenger car accidents," *Accident Analysis & Prevention*, **36**(2), pp. 135–147.

[100] Khorashadi, A., D. Niemeier, V. Shankar, and F. Mannering (2005) "Differences in rural and urban driver-injury severities in accidents involving large-trucks: an exploratory analysis," *Accident Analysis & Prevention*, **37**(5), pp. 910–921.

[101] Bhat, C. R. (2011) *The MACML estimation of the normally-mixed multinomial logit model*, *Tech. rep.*, University of Texas at Austin, Department of Civil, Architectural and Environmental Engineering, Austin, Texas.

[102] McFadden, D. (1981) "Econometric Models of Probabilistic Choice," in *STRUCTURAL ANALYSIS OF DISCRETE DATA WITH ECONOMETRIC APPLICATIONS* (C. Manski and D. McFadden, eds.), chap. 5, MIT Press, pp. 198–272.

[103] McFadden, D. and K. Train (2000) "Mixed MNL models for discrete response," *Journal of Applied Econometrics*, **15**, pp. 447–470.

[104] McFadden, D. (2001) "Economic choices," *The American economic review*, **91**(3), pp. 351–378.

[105] Le Cessie, S. and J. C. Van Houwelingen (1992) "Ridge estimators in logistic regression," *Applied statistics*, pp. 191–201.

[106] Mannering, F. L. and C. R. Bhat (2014) "Analytic methods in accident research: Methodological frontier and future directions," *Analytic methods in accident research*, **1**, pp. 1–22.

[107] StataCorp (2015) *Stata 15 Base Reference Manual.*

[108] Hilbe, J. (2006) "A Review of LIMDEP 9.0 and NLOGIT 4.0," *The American Statistician*, **60**, pp. 187–202.
URL `https://EconPapers.repec.org/RePEc:bes:amstat:v:60:y:2006:m:may:p:187-202`

[109] Regier, D. A., M. Ryan, E. Phimister, and C. A. Marra (2009) "Bayesian and classical estimation of mixed logit: an application to genetic testing," *Journal of health economics*, **28**(3), pp. 598–610.

[110] Hadfield, J. D. et al. (2010) "MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package," *Journal of Statistical Software*, **33**(i02).

[111] Steele, F., E. Washbrook, C. Charlton, and W. J. Browne (2016) "A longitudinal mixed logit model for estimation of push and pull effects in residential location choice," *Journal of the American Statistical Association*, **111**(515), pp. 1061–1074.

[112] Train, K. (2000) *Halton Sequences for Mixed Logit*, *Tech. rep.*, University of California Berkeley, Department of Economics, Berkeley, California.

[113] Bhat, C. R. (2001) "Quasi-random maximum simulated likelihood estimation of the mixed multinomial logit model," *Transportation Research Part B: Methodological*, **35**(7), pp. 677–693.

[114] Bastin, F., C. Cirillo, and P. L. Toint (2004) "Estimating mixed logit models with quasi-Monte Carlo sequences allowing practical error estimation," in *the European Transport Conference*, Strasbourg, France.

[115] Munger, D., P. L'Ecuyer, F. Bastin, C. Cirillo, and B. Tuffin (2012) "Estimation of the mixed logit likelihood function by randomized quasi-Monte Carlo," *Transportation Research Part B: Methodological*, **46**(2), pp. 305–320.

[116] Bastin, F., C. Cirillo, and P. L. Toint (2006) "Application of an adaptive Monte Carlo algorithm to mixed logit estimation," *Transportation Research Part B: Methodological*, **40**(7), pp. 577–593.

[117] Bastin, F., C. Cirillo, and S. Hess (2005) "Evaluation of optimization methods for estimating mixed logit models," *Transportation Research Record: Journal of the Transportation Research Board*, (1921), pp. 35–43.

[118] Mai, A. T., F. Bastin, and M. Toulouse (2014) *On Optimization Algorithms for Maximum Likelihood Estimation*, *Tech. rep.*, CIRRELT, Montréal, Canada.

[119] Hole, A. R. and H. I. Yoo (2017) "The use of heuristic optimization algorithms to facilitate maximum simulated likelihood estimation of random parameter logit models," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, **66**(5), pp. 997–1013.

[120] Robbins, H. and S. Monro (1985) "A stochastic approximation method," in *Herbert Robbins Selected Papers*, Springer, pp. 102–109.

[121] Lederrey, G., V. Lurkin, and M. Bierlaire (2018) "SNM: Stochastic Newton Method for Optimization of Discrete Choice Models," in *2018 IEEE 21th International Conference onIntelligent Transportation Systems (ITSC)*, IEEE, pp. 1–6.

[122] Nemirovski, A., A. Juditsky, G. Lan, and A. Shapiro (2009) "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, **19**(4), pp. 1574–1609.

[123] Hoerl, A. E. and R. W. Kennard (1970) "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, **12**(1), pp. 55–67.

[124] Lan, G. (2012) "An optimal method for stochastic composite optimization," *Mathematical Programming*, **133**(1-2), pp. 365–397.

[125] Bottou, L. (2012) "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, Springer, pp. 421–436.

[126] Needell, D., R. Ward, and N. Srebro (2014) "Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm," in *Advances in Neural Information Processing Systems*, pp. 1017–1025.

[127] Chee, J. and P. Toulis (2018) "Convergence diagnostics for stochastic gradient descent with constant learning rate," in *International Conference on Artificial Intelligence and Statistics*, pp. 1476–1485.

[128] Bottou, L., F. E. Curtis, and J. Nocedal (2018) "Optimization methods for large-scale machine learning," *SIAM Review*, **60**(2), pp. 223–311.

[129] Nesterov, Y. (1983) "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," *Soviet Mathematics Doklady*, **27**(a), pp. 372–376.

[130] MATLAB (2018) *version 9.5.0 (R2010b)*, The MathWorks Inc., Natick, Massachusetts.

[131] Li, Q., Y. Zhou, Y. Liang, and P. K. Varshney (2017) "Convergence Analysis of Proximal Gradient with Momentum for Nonconvex Optimization," in *International Conference on Machine Learning*, pp. 2111–2119.

[132] Boost (2018), “Boost C++ Libraries,” `http://www.boost.org/`, last accessed 2018-11-21.

[133] The California Highway Patrol (2014), “Statewide Integrated Traffic Records System,” .
URL `\url{http://iswitrs.chp.ca.gov/Reports/jsp/userLogin.jsp}`

[134] NHTSA (2017) *TRAFFIC SAFETY FACTS 2015*, *Tech. rep.*, National Highway Traffic Safety Administration, Washington, DC.
URL `\url{https://crashstats.nhtsa.dot.gov/}`

[135] McFadden, D. (1974) “Conditional Logit Analysis of Qualitative Choice Behavior,” *Frontiers in Econometrics*, pp. 105–142.

[136] Revelt, D. and K. Train (1998) “Mixed logit with repeated choices: households’ choices of appliance efficiency level,” *Review of economics and statistics*, **80**(4), pp. 647–657.

[137] NHTSA (2007) *TRAFFIC SAFETY FACTS 2007*, *Tech. rep.*, National Highway Traffic Safety Administration, Washington, DC.
URL `\url{https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/810993}`

[138] Von Luxburg, U. (2007) “A tutorial on spectral clustering,” *Statistics and computing*, **17**(4), pp. 395–416.

[139] Mannering, F. (2018) “Temporal instability and the analysis of highway accident data,” *Analytic methods in accident research*, **17**, pp. 1–13.

[140] Chandrasekaran, V., P. A. Parrilo, A. S. Willsky, et al. (2012) “Latent variable graphical model selection via convex optimization,” *The Annals of Statistics*, **40**(4), pp. 1935–1967.

[141] Richard, E., P.-A. Savalle, and N. Vayatis (2012) “Estimation of Simultaneously Sparse and Low Rank Matrices,” in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML’12, USA, pp. 51–58.

[142] Meng, Z., B. Eriksson, and A. Hero (2014) “Learning latent variable Gaussian graphical models,” in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1269–1277.

[143] Oymak, S., A. Jalali, M. Fazel, Y. C. Eldar, and B. Hassibi (2015) “Simultaneously Structured Models With Application to Sparse and Low-Rank Matrices,” *IEEE Transactions on Information Theory*, **61**(5), pp. 2886–2908.

[144] Chen, Y., Y. Chi, and A. J. Goldsmith (2015) "Exact and stable covariance estimation from quadratic sampling via convex programming," *IEEE Transactions on Information Theory*, **61**(7), pp. 4034–4059.

[145] Saunderson, J., V. Chandrasekaran, P. A. Parrilo, and A. S. Willsky (2012) "Diagonal and low-rank matrix decompositions, correlation matrices, and ellipsoid fitting," *SIAM Journal on Matrix Analysis and Applications*, **33**(4), pp. 1395–1416.

[146] Liutkus, A. and K. Yoshii (2017) "A diagonal plus low-rank covariance model for computationally efficient source separation," in *Machine Learning for Signal Processing (MLSP), 2017 IEEE 27th International Workshop on*, IEEE, pp. 1–6.

[147] Lee, D. D. and H. S. Seung (1999) "Learning the parts of objects by non-negative matrix factorization," *Nature*, **401**(6755), p. 788.

[148] ——— (2001) "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, pp. 556–562.

[149] Ding, C., X. He, and H. D. Simon (2005) "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, SIAM, pp. 606–610.

[150] Ding, C., T. Li, and W. Peng (2008) "On the equivalence between nonnegative matrix factorization and probabilistic latent semantic indexing," *Computational Statistics & Data Analysis*, **52**(8), pp. 3913–3927.

[151] Abdi, H. and L. J. Williams (2010) "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, **2**(4), pp. 433–459.

[152] Gillis, N. (2014) "The why and how of nonnegative matrix factorization," *Regularization, Optimization, Kernels, and Support Vector Machines*, **12**(257).

[153] Fazel, M., H. Hindi, and S. P. Boyd (2001) "A rank minimization heuristic with application to minimum order system approximation," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 6, IEEE, pp. 4734–4739.

[154] Recht, B., W. Xu, and B. Hassibi (2008) "Necessary and sufficient conditions for success of the nuclear norm heuristic for rank minimization," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, IEEE, pp. 3065–3070.

[155] Recht, B., M. Fazel, and P. A. Parrilo (2010) "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization," *SIAM review*, **52**(3), pp. 471–501.

[156] Yuan, M. and Y. Lin (2006) "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **68**(1), pp. 49–67.

[157] OâĂŹdonoghue, B. and E. Candes (2015) "Adaptive restart for accelerated gradient schemes," *Foundations of computational mathematics*, **15**(3), pp. 715–732.

[158] Halko, N., P.-G. Martinsson, and J. A. Tropp (2011) "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, **53**(2), pp. 217–288.

[159] Friedman, J., T. Hastie, H. Höfling, R. Tibshirani, et al. (2007) "Pathwise coordinate optimization," *The Annals of Applied Statistics*, **1**(2), pp. 302–332.

[160] Friedman, J., T. Hastie, and R. Tibshirani (2010) "Regularization paths for generalized linear models via coordinate descent," *Journal of statistical software*, **33**(1), p. 1.

[161] Shankar, V., R. Albin, J. Milton, and F. Mannering (1998) "Evaluating median crossover likelihoods with clustered accident counts: An empirical inquiry using the random effects negative binomial model," *Transportation Research Record: Journal of the Transportation Research Board*, (1635), pp. 44–48.

[162] Anastasopoulos, P. C. and F. L. Mannering (2009) "A note on modeling vehicle accident frequencies with random-parameters count models," *Accident Analysis & Prevention*, **41**(1), pp. 153–159.

[163] Kim, J.-K., G. F. Ulfarsson, V. N. Shankar, and F. L. Mannering (2010) "A note on modeling pedestrian-injury severity in motor-vehicle crashes with the mixed logit model," *Accident Analysis & Prevention*, **42**(6), pp. 1751–1758.

[164] Anastasopoulos, P. C. and F. L. Mannering (2011) "An empirical assessment of fixed and random parameter logit models using crash-and non-crash-specific injury data," *Accident Analysis & Prevention*, **43**(3), pp. 1140–1147.

[165] Morgan, A. and F. L. Mannering (2011) "The effects of road-surface conditions, age, and gender on driver-injury severities," *Accident Analysis & Prevention*, **43**(5), pp. 1852–1863.

[166] Kim, J.-K., G. F. Ulfarsson, S. Kim, and V. N. Shankar (2013) "Driver-injury severity in single-vehicle crashes in California: a mixed logit analysis of heterogeneity due to age and gender," *Accident Analysis & Prevention*, **50**, pp. 1073–1081.

[167] Brin, S. and L. Page (1998) "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, **30**, pp. 107–117.

[168] Gleich, D. F. (2015) "PageRank Beyond the Web," *SIAM Review*, **57**(3), pp. 321–363.

[169] Fortunato, S. (2010) "Community detection in graphs," *Physics Reports*, **486**(3), pp. 75–174.

[170] Everett, M. G. and S. P. Borgatti (1999) "The centrality of groups and classes," *Journal of Mathematical Sociology*, **23**(3), pp. 181–201.

[171] Zhan, H. and K. Madduri (2017) "Analyzing Community Structure in Networks," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, pp. 1540–1549.

[172] Li, T. (2005) "A General Model for Clustering Binary Data," in *Proc. ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data Mining (KDD).*

[173] Lee, D. D. and H. S. Seung (1999) "Learning the parts of objects by non-negative matrix factorization," *Nature*, **401**, pp. 788–791.

[174] ——— (2001) "Algorithms for Non-negative Matrix Factorization," in *Advances in Neural Information Processing Systems (NIPS) 13* (T. K. Leen, T. G. Dietterich, and V. Tresp, eds.), MIT Press, pp. 556–562.

[175] Zhang, Y. and D.-Y. Yeung (2012) "Overlapping community detection via bounded nonnegative matrix tri-factorization," in *Proc. ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data Mining (KDD).*

[176] Wang, F., T. Li, X. Wang, S. Zhu, and C. Ding (2011) "Community discovery using nonnegative matrix factorization," *Data Mining and Knowledge Discovery*, **22**(3), pp. 493–521.

[177] Newman, M. E. (2006) "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, **103**(23), pp. 8577–8582.

[178] Radicchi, F., C. Castellano, F. Cecconi, V. Loreto, and D. Parisi (2004) "Defining and identifying communities in networks," *Proceedings of the National Academy of Sciences of the United States of America*, **101**(9), pp. 2658–2663.

[179] Fortunato, S. (2010) “Community detection in graphs,” *Physics Reports*, **486**(3), pp. 75–174.

[180] Yang, J. and J. Leskovec (2015) “Defining and evaluating network communities based on ground-truth,” *Knowledge and Information Systems*, **42**(1), pp. 181–213.

[181] Paatero, P. and U. Tapper (1994) “Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values,” *Environmetrics*, **5**(2), pp. 111–126.

[182] Yu, H.-F., C.-J. Hsieh, S. Si, and I. S. Dhillon (2014) “Parallel matrix factorization for recommender systems,” *Knowledge and Information Systems*, **41**(3), pp. 793–819.

[183] Lawson, C. L. and R. J. Hanson (1974) *Solving least squares problems*, SIAM.

[184] Luo, Y. and R. Duraiswami (2011) “Efficient parallel nonnegative least squares on multicore architectures,” *SIAM Journal on Scientific Computing*, **33**(5), pp. 2848–2863.

[185] Bertsekas, D. P. (1999) *Nonlinear programming*, Athena Scientific.

[186] Franc, V., V. Hlaváč, and M. Navara (2005) “Sequential coordinate-wise algorithm for the non-negative least squares problem,” in *Computer Analysis of Images and Patterns* (A. Gagalowicz and W. Philips, eds.), vol. 3691 of *Lecture Notes in Computer Science*, Springer, pp. 407–414.

[187] Hsieh, C.-J. and I. S. Dhillon (2011) “Fast coordinate descent methods with variable selection for non-negative matrix factorization,” in *Proc. ACM SIGKDD Int’l. Conf. on Knowledge Discovery and Data Mining (KDD).*

[188] Johansson, B., T. Elfving, V. Kozlov, Y. Censor, P.-E. Forssén, and G. Granlund (2006) “The application of an oblique-projected Landweber method to a model of supervised learning,” *Mathematical and computer modelling*, **43**(7), pp. 892–909.

[189] Lancichinetti, A., S. Fortunato, and F. Radicchi (2009) “Benchmark graphs for testing community detection algorithms,” *Physical Review E*, **78**, p. 046110.

[190] Zhan, H. and K. Madduri (2016) “GSK: Graph Sparsification as a Knapsack Problem Formulation,” in *Proc. 3rd SDM Workshop on Mining Networks and Graphs (MNG).*

[191] Serrano, M. Á., M. Boguná, and A. Vespignani (2009) "Extracting the multiscale backbone of complex weighted networks," *Proceedings of the National Academy of Sciences (PNAS)*, **106**(16), pp. 6483–6488.

[192] Foti, N. J., J. M. Hughes, and D. N. Rockmore (2011) "Nonparametric sparsification of complex multiscale networks," *PLoS ONE*, **6**(2), p. e16431.

[193] Satuluri, V., S. Parthasarathy, and Y. Ruan (2011) "Local graph sparsification for scalable clustering," in *Proc. ACM SIGMOD Int'l. Conf. on Management of Data (SIGMOD)*, ACM, pp. 721–732.

[194] Lindner, G., C. L. Staudt, M. Hamann, H. Meyerhenke, and D. Wagne (2015) "Structure-Preserving Sparsification of Social Networks," in *Proc. IEEE/ACM Int'l. Conf. on Advances in Social Networks Analysis and Mining (ASONAM)*, IEEE/ACM, pp. 448–454.

[195] Fung, W. S., R. Hariharan, N. J. Harvey, and D. Panigrahi (2011) "A general framework for graph sparsification," in *Proc. 43rd Annual ACM Symp. on Theory of Computing (STOC)*, ACM, pp. 71–80.

[196] Benczúr, A. A. and D. R. Karger (2015) "Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs," *SIAM Journal on Computing*, **44**(2), pp. 290–319.

[197] Spielman, D. A. and S.-H. Teng (2011) "Spectral sparsification of graphs," *SIAM Journal on Computing*, **40**(4), pp. 981–1025.

[198] Bonchi, F., G. D. F. Morales, A. Gionis, and A. Ukkonen (2013) "Activity preserving graph simplification," *Data Mining and Knowledge Discovery*, **27**(3), pp. 321–343.

[199] Wilder, B. and G. Sukthankar (2015) "Sparsification of Social Networks Using Random Walks," in *Proc. 8th ASE Int'l. Conf. on Social Computation (SocialCom)*, ASE.

[200] Bretthauer, K. M. and B. Shetty (2002) "The nonlinear knapsack problem–algorithms and applications," *European Journal of Operational Research*, **138**(3), pp. 459–472.

[201] Sharkey, T. C., H. E. Romeijn, and J. Geunes (2011) "A class of nonlinear nonseparable continuous knapsack and multiple-choice knapsack problems," *Mathematical Programming*, **126**(1), pp. 69–96.

[202] Leskovec, J., A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance (2007) "Cost-effective outbreak detection in networks," in *Proc. 13th ACM SIGKDD Int'l. Conf. on Knowledge discovery and data mining (KDD)*, ACM, pp. 420–429.

[203] Slater, P. B. (2009) "Multiscale Network Reduction Methodologies: Bistochastic and Disparity Filtering of Human Migration Flows between 3,000+ US Counties," *arXiv preprint arXiv:0907.2393*.

[204] Bellman, R. (1956) "Notes on the theory of dynamic programming IV-Maximization over discrete sets," *Naval Research Logistics Quarterly*, **3**(1-2), pp. 67–70.

[205] Kleinberg, J. M. and Éva Tardos (2006) *Algorithm Design*, first ed., Addison-Wesley.

[206] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009) *Introduction to Algorithms (3. ed.)*, MIT Press.
URL `http://mitpress.mit.edu/books/introduction-algorithms`

[207] Sahni, S. (1975) "Approximate algorithms for the 0/1 knapsack problem," *Journal of the ACM*, **22**(1), pp. 115–124.

[208] Balas, E. and E. Zemel (1980) "An algorithm for large zero-one knapsack problems," *operations Research*, **28**(5), pp. 1130–1154.

[209] Pisinger, D. (1999) "Linear time algorithms for knapsack problems with bounded weights," *Journal of Algorithms*, **33**(1), pp. 1–14.

[210] Fayard, D. and G. Plateau (1982) "An algorithm for the solution of the 0–1 knapsack problem," *Computing*, **28**(3), pp. 269–287.

[211] Ibarra, O. H. and C. E. Kim (1975) "Fast approximation algorithms for the knapsack and sum of subset problems," *Journal of the ACM*, **22**(4), pp. 463–468.

[212] Lawler, E. L. (1979) "Fast approximation algorithms for knapsack problems," *Mathematics of Operations Research*, **4**(4), pp. 339–356.

[213] Kellerer, H. and U. Pferschy (1999) "A new fully polynomial time approximation scheme for the knapsack problem," *Journal of Combinatorial Optimization*, **3**(1), pp. 59–71.

[214] Korte, B. and J. Vygen (2008) *Combinatorial optimization: Theory and algorithms*, fourth ed., Springer-Verlag Berlin Heidelberg.

[215] Krause, A. and D. Golovin (2012) "Submodular function maximization," *Tractability: Practical Approaches to Hard Problems*, **3**, p. 19.

[216] Blum, M., R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan (1973) "Time bounds for selection," *Journal of computer and system sciences*, **7**(4), pp. 448–461.

[217] Yang, J. and J. Leskovec (2015) "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, **42**(1), pp. 181–213.

[218] Leskovec, J., J. Kleinberg, and C. Faloutsos (2005) "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int'l. Conf. on Knowledge discovery and data mining (KDD)*, ACM, pp. 177–187.

[219] "SNAP Stanford Large Network Dataset Collection," `http://snap.stanford.edu/data/index.html`, last accessed Jan 2016.

[220] Spearman, C. (1904) "The Proof and Measurement of Association between Two Things," *The American journal of psychology*, **15**(1), pp. 72–101.

[221] Ghosh, B., B. Basu, and M. O'Mahony (2009) "Multivariate short-term traffic flow forecasting using time-series analysis," *IEEE Transactions on Intelligent Transportation Systems*, **10**(2), pp. 246–254.

[222] Pascale, A. and M. Nicoli (2011) "Adaptive Bayesian NETWORK FOR TRAFFIC FLOW PREDICTION," in *Proceeding of the IEEE Statistical Signal Processing (SSP) Workshop*.

[223] Cremer, M. and H. Keller (1987) "A new class of dynamic methods for the identification of origin-destination flows," *Transportation Research Part B: Methodological*, **21**(2), pp. 117–132.

[224] Polson, N. and V. Sokolov (2018) "Bayesian particle tracking of traffic flows," *IEEE Transactions on Intelligent Transportation Systems*, **19**(2), pp. 345–356.

[225] Seo, T., A. M. Bayen, T. Kusakabe, and Y. Asakura (2017) "Traffic state estimation on highway: A comprehensive survey," *Annual Reviews in Control*, **43**, pp. 128–151.

[226] Barzilai, J. and J. M. Borwein (1988) "Two-point step size gradient methods," *IMA journal of numerical analysis*, **8**(1), pp. 141–148.

[227] Meinshausen, N. and P. Bühlmann (2006) "High-dimensional graphs and variable selection with the lasso," *The annals of statistics*, pp. 1436–1462.

[228] Yuan, M. and Y. Lin (2007) "Model selection and estimation in the Gaussian graphical model," *Biometrika*, **94**(1), pp. 19–35.

[229] Banerjee, O., L. E. Ghaoui, and A. dâĂŹAspremont (2008) "Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data," *Journal of Machine learning research*, **9**(Mar), pp. 485–516.

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

Vita

Hongyuan Zhan

Hongyuan Zhan is a PhD candidate in the Computer Science and Engineering department at The Pennsylvania State University, University Park. He received the Bachelor degree in Mathematics with Honors from Penn State in 2014. His research interests are at the intersection of machine learning for time-series and optimization, with applications to intelligent transportation and finance.