University of Technology Sydney (UTS)
School of Mathematical and Physical Sciences
Faculty of Science

Statistical Analysis on Distributed Systems

by

Hon Hwang

A thesis submitted
in partial fulfilment of the
requirements for the degree

Doctor of Philosophy

Sydney, Australia

Certificate of authorship

I, Hon Hwang declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Science, at the University of Technology Sydney (UTS).

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution.
This research is supported by an Australian Government Research Training program.

Signature

Production Note:
Signature removed
prior to publication.

Date

19/07/2019

Acknowledgement

In addition to the Australian Government Research Training Program, this research is also supported by The Australian Research Council (ARC) Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS), UTS ARC Lab and the Bill & Melinda Gates Foundation. I wish to recognise the principal investigators and their study team members in the Ki (knowledge integration) team for their generous contribution of the data that made this thesis possible and the members of the Ki team who directly or indirectly contributed to the study. The article contents are the sole responsibility of the authors and may not necessarily represent the official views of the Bill & Melinda Gates Foundation or other agencies that may have supported the primary data studies used in the present study.

I like to express my deep gratitude to my PhD advisor Professor Louise Ryan. She took on the risk of accepting someone like me to be her PhD student when my academic background is not in in the areas of statistics. She is always positive, encouraging and generous with her time. I am always amazed at her level of knowledge and work rate. I am incredibly fortunate to have her as my PhD advisor, and I am very grateful.

I like to thank Professor Matt Wand for his time in helping with my thesis and for the weekly discussion sessions of our statistics group. I also like to thank Dr Ryan Hafen for the help with the computational aspects of this thesis. I also like to thank postdoctoral fellows that have helped and guided me, especially Dr Stephen T. Wright, and Dr Craig Anderson. I also wish to thank Jonathan French, who helped enormously with the Ki project.

Embarking on a PhD after years of working outside of academia was not a decision taken in an instance. I like to thank Sheldon Dealy and Dr Bing Liu for helping me make the decision years ago.

Undertaking a PhD is a challenging experience. I like to thank my fellow PhD students for the esprit de corps, which always raise my spirit during difficult times. So thanks to Tea, Alan, James, Bojana, Mahrita, and Jenny. My thanks also extend to former PhD students such as Marianne, Joey, Cathy, Hamidul, and Jarod.

Finally, I like to express my gratitude to my parents for putting up with me again; once during my undergraduate years and again during my PhD. Thanks for the patience and support which allowed me to focus more on my PhD. The thanks also extend to my siblings.

Contents

List of Figures

List of Tables

Abstract

Organisations are increasingly storing the large amount of data they collect using distributed computing system such as Hadoop. In these distributed computing systems, instead of storing all the data on a single computer, which is unfeasible in the big data setting, data are divided into subsets, which are then stored across many computers. These distributed data, however, present a challenge to many statistical methods and algorithms since they assume all the data are available at one location (e.g., in the memory of a single computer). This thesis explores how statistical analysis can be performed on distributed computing systems using the divide and recombine paradigm, where analysis is performed on individual subsets of the data, follow by combining the results of these analyses. This thesis demonstrates how existing statistical methods such as coarsening, the EM algorithm, and meta-analysis can be used in the divide and recombine paradigm to perform statistical analysis on big data. In addition, this thesis investigates the use of other statical methods such as multiple imputation, propensity scoring and penalised spline when working with so called 'messy' data. This thesis shows that the divide and recombine paradigm is more than just the simple averaging of results from the subsets. By tailoring existing statistical methods for divide and recombine, as well as abstracting the idea of distributed data, this thesis shows that many existing statical methods are still relevant in today's big data world.

1 Introduction

Consider a statistical analyst working for an organisation that has accumulated a large amount of data and also is continuously collecting more. Examples include organisations in the fields of health insurance, telecommunication, finance, retail and so on. In these industries, it is reasonable to expect there are lots of data on various parts of the business. For example, a telecommunication company can have data about its customers, and data on its telecommunication network. Suppose the analyst's task is to analyse the customer data in order to provide insight that can help the company provide more effective and useful services. A typical data analysis workflow may follow the workflow shown in Wickham and Grolemund (2016, Chapter 2) consisting of steps to import, tidy, and iterating over transform, visualise, and model steps, and finally communicating the result. The first task our analyst performs is to load the customer data into her computer (import step). Assuming the data is well organised, our analyst aims to perform some exploratory data analysis, before carrying out the modelling. However, in this case, the analyst is unable to import the data into her computer because the data are too large to be stored in the memory of her computer. Without being able to put all the data into a single location (the memory of her computer), our analyst will encounter difficulty in performing even the most basic exploratory data analysis task such as generating a histogram. Our analyst can expect even more difficult challenges when attempting to fit a statistical model using the large data set, because many statistical software packages assume that all the data are available at a single location, typically in the memory of the analyst's computer.

The situation described above is one of three scenarios our analyst can encounter when working with very large datasets. The three scenarios are

A) The data can be stored in the memory and non-memory storage (e.g., hard drive, or solid state drive) of a single computer;

B) The data are too large to be stored in the memory, but can be stored in non-memory storage of a single computer; and

C) The data are so large that they overwhelm both the memory and non-memory storage systems of a single computer.

The best case scenario is A in which our analyst does not need to take any special actions. Since all the data can be imported into the memory of her computer, she can perform all the data analysis steps. The statistical software packages she currently uses still work as all the data are in one location. For Scenario B, she can use a number of statistical and computational techniques such as data reduction, decomposition and data swapping (see discussion

below) so the data can be stored in the memory of a single machine, or appears to be stored in memory. However, because her company is continuously collecting data on its customers, it's a matter of time before these approaches prove inadequate as well. In addressing this problem(scenario C), many organisations utilise some form of distributed system, where the data are held across many computers. As an example, in many organisations, there are large networked storage spaces where project documents are stored. These networked storage spaces are typically provided through technologies such as network attached storage, where the data are stored across multiple computers. Another type of distributed system used for storage is a storage area network, which is typically used in areas such as video editing where large amount of video data is stored across many powerful computers. Commercial data storage systems such as Dropbox also have to utilise distributed storage systems to store all the data from its customers (Dropbox, n.d.) [1]. Even though distributed systems are initially used to address sizes of data, a common benefit of these systems is data redundancy. Many distributed systems can recover data that are lost when a node fails. One method of redundancy is to have multiple copies of data. Traditionally, to perform computation on these distributed data organisations typically have another cluster of powerful computers which can access the data using high speed data networks. In scientific computing, these are usually termed high performance computing (HPC) facilities. More recent distributed computing systems such as Hadoop (The Apache Software Foundation, n.d.[a]) [2] or Spark (The Apache Software Foundation, n.d.[d]) allow analysis to be performed on large data sets by storing the data across many off-the-shelf computers and running computations based on the Divide and Recombine (D&R) strategy[3], which can be implemented using the MapReduce [4] programming abstraction [5].

One can argue that using a distributed computing system is an over-elaborate solution to the problem of performing statistical analysis on very large datasets. David Donoho, in a talk given at Tukey Centennial Workshop Princeton University argued that '*Lost in the hoopla about such skills [Hadoop] is the embarrassing fact that once upon a time, one could do such computing tasks, and even much more ambitious ones, much more easily than in this fancy new setting! ... In this highly constrained world [distributed computing systems], the range of easily constructible algorithms shrinks dramatically compared to the single-processor [single*

[1] n.d. stands for no date. This abbreviation is typically used to indicate references with no specific publication date, such as websites and software

[2] [a],[b],...reference published in the same year, including no date references, by the same author.

[3] In Divide and Recombine, an operation is performed across individual subsets of data, the results are then combined. Section 1.3 has more detail on this topic.

[4] MapReduce is one method of implementing the Divide and Recombine strategy. Section2.3.1 has more detail.

[5] A description of a pattern of computational operation. Section1.3 refines this definition

machine] model, so one inevitably tends to adopt inferential approaches which would have been considered rudimentary or even inappropriate in olden times' (Donoho, 2017). However, recall that in our scenario, our analyst cannot even import the dataset into her computer, let alone performing statistical analysis. Nevertheless, distributed computing systems are more complex to use. Therefore, techniques that enable statistical analysis of very large data sets to be performed on a single computer should be explored.

1.1 Strategies to handle Big Data on a single computer

Scenario B shown above is fairly typical in many situations because of the amount of computer memory on a single computer is often much less than the amount of non-memory (secondary) storage (hard disk drive, or solid state drive). For example, a workstation level computer may have 32 GB of memory, while its secondary storage may be in the terabytes. To be able to analyse a dataset in the hundred of gigabytes on her computer, our analyst can utilise a number of computational techniques.

One computational technique that our analyst can benefit from is *swapping*. Swapping involves moving data between memory and non-memory storage to create the illusion that all the data are stored in memory. Modern operating systems such as GNU/Linux, Windows, macOS use swapping techniques in their *virtual memory* system that allows software to allocate more memory than physically possible. The `ff` package (Adler et al., 2014) of R (R Core Team, 2017) is an example that implements swapping. '*The `ff` package provides data structures that are stored on disk but behave (almost) as if they were in RAM by transparently mapping only a section (pagesize) in main memory*' (Adler et al., 2014). Another example of software library that let our analyst directly use swapping is MATLAB's `Tall Array` (The MathWorks, n.d.). The major downside to using swapping is slower data access, compared to accessing all the data directly from memory. Hadley Wickham, the Chief Scientist at RStudio, summarises this downside as '*Moving from in-memory to on-disk is an important transition because access speeds are so different. You can do quite naive computations on in-memory data and it'll be fast enough. You need to plan (and index) much more with on-disk data*' (Coyle, 2015). Since data in memory can be accessed much quicker than those on disk, moving data between these will incur a performance penalty.

In conjunction with swapping, our analyst can also upgrade her computer so it has more memory and a more powerful CPU. The combination of these approaches may be enough to allow her to import the data and perform exploratory data analysis. However, this approach of upgrading her computer can be financially costly. As an alternative, our analyst can apply statistical techniques to reduce her very large data set before upgrading her computer.

Conceptually, data reduction techniques scale down the data, as opposed to scaling up the algorithm (García, Luengo, and Herrera, 2016). This allows very large data sets to be stored in the memory of a single computer. For visualisation, the analyst has a variety of choices of data reduction techniques. These include sampling (also referred to as filtering), bin aggregation of data points, and model-based reduction techniques such as moving averages for time series data (Liu, Jiang, and Heer, 2013; Scheidegger, 2016). If the large data set is stored in a database, she can use *synopses* to perform Approximate Query Processing (AQP) (Cormode, 2011; Liu, Jiang, and Heer, 2013) [6]. A synopsis '*captures vital properties of the original data while typically occupying much less space*' (Cormode, 2011, p.2). Examples of synopsis include random samples, histograms and wavelets (Cormode et al., 2011). The methods described above reduce the dataset through some form of summary statistics. Although data reduction is achieved, if these summary statistics are used directly in statistical models, such as linear regression, the resulting estimates can be biased. Chapter 3 explores this issue and presents way of approximating the "complete data" model in the context of logistic regression.

An alternative approach to creating summary statistics to reduce data is through subsampling representative samples. Techniques such as *instance selection* from the field of data mining which '*consists of choosing a subset of data that could replace the whole data to achieve the original goal of DM [data mining] application* (García, Luengo, and Herrera, 2016, p.10). Instance Selection is achieved through clustering of data points around characteristics of interest, then selecting representative samples from each cluster. An interesting statistical method is the use of *coresets* (Campbell and Broderick, 2017; Huggins, Campbell, and Broderick, 2016). Coresets are a weighted subset of the data, where the weights are calculated such that the weighted log-likelihood approximates the log-likelihood of the original dataset (Campbell and Broderick, 2017). Huggins, Campbell, and Broderick (2016) showed the construction and use of coresets in Bayesian logistic regression setting. Huggins, Campbell, and Broderick (2016) extended this work further to address its shortcomings. Although using a representative sub-sample may result in estimates with less bias, compared to using summary statistics, the issue of obtaining a representative sub-sample is discussed briefly in Chapter 3. In particular, the limitation of sub-sampling in situations where the outcome event is rare.

Since many multivariate statistical models are expressed as matrices, in the modelling stage of the analysis, our analyst can benefit from techniques from linear algebra such as matrix decomposition. Gentle (2009) cites the use of matrix decomposition, such as QR decomposition, as one of the top 10 algorithms '*with the greatest influence on the development and practice of science and engineering in the 20th century*' (Sullivan and Dongarra, 2000). An ex-

[6] AQP is a method of searching over a large dataset, where instead the queries are applied against the compressed (summarised) form of the dataset as opposed to over the entire dataset.

ample of R software package that make use of matrix decomposition to enable fitting of regression models using very large data sets is `biglm` (Lumley, 2013). The `biglm` package uses QR matrix decomposition and incremental updating, which allow it to to estimate regression coefficients without the need for all the data to be stored in memory in the first place.

In Bayesian modelling, for models where the posterior distributions cannot be computed, approximate inference algorithms such as Markov chain Monte Carlo (MCMC), and more recently variational approximation approaches are used (Lee and Wand, 2016; Ormerod and Wand, 2010). Variational approximation techniques such as Mean Field Variational Bayes can approximate the posterior distribution without the need to perform Markov chain Monte Carlo, which can be time consuming However, the algorithms from these techniques need to be modified to handle very large datasets.Campbell and Broderick (2017) and Huggins, Campbell, and Broderick (2016) summarise some of these modifications such as streaming methods (processing data one by one), subsampling and distributed method such as Consensus Monte Carlo (Scott and Blocker, 2013).

Larger memory, better algorithms and using technique such as swapping may allow our analyst to import the data and in some circumstances (such as when using `biglm`) to perform some analysis on the dataset. During analysis, our analyst will have to consider a related issue to data size, which is computational speed. An obvious approach is to utilise the capabilities of the modern CPU, in particular, the number of computational unit (cores) in a CPU. It is not uncommon to have ordinary desktop or laptops to have CPUs with 4 cores. Workstation and server class computers can have CPUs with 6 to 10 cores, as well as having multiple CPUs. To take advantage of multiple cores, our analyst can use software packages such as *parallel* (R Core Team, 2017) in R to parallelise her analysis code and algorithm. For example, she can use `mclapply()` and `parLapply()` functions to execute a a function in parallel.

In parallel to powerful CPUs, many modern computers also have powerful graphic processing units (GPU). Modern GPUs contain many specialised computing units (similar to computing cores of a CPU). For example, the Turing TU102 GPU from nVidia contains 4,608 compute cores for general computing and 576 *Tensor cores* specialised for deep learning application (nVidia Corporation, 2018). Compared to using multi-core capabilities of CPUs, using GPUs will require more complex modification of analysis code. This is because GPUs have completely different architecture from CPUs. For example, to use the computation capabilities of nVidia GPUs, the analyst will most likely need to modify her code to use nVidia's software library called CUDA (Nickolls et al., 2008). Another issue with incorporating GPU code is our analyst will need to further consider another level of data movement between the CPU and the GPU, since the GPUs have their own memory.

An example of a software package that incorporates the ideas above to model large data

set is the work by Wood et al. (2017). The large data set consists of daily measurements from the United Kingdom Black Smoke monitoring network, taken between 1961 to 2005. The data set contains 9,451,232 daily measurements across 2862 monitoring sites (Wood et al., 2017). To fit a Generalized Additive Model (GAM) on this large data set, the authors make use of statistical and computational techniques such as clever use of matrix operations, discretization of covariates and parallel computation implementation. In addition to providing computational efficiency, this combination allowed the authors to construct GAM for large data sets, which was not feasible before.

1.2 Strategies to handle big data with many computers

So far our analyst has been attempting to perform the analysis on her own individual computer. However, since her company is continuously collecting data on its business, at some point using just a single computer to process the data will become infeasible. This situation, one can assume, can occur in large organisations with data-centric business, such as Google. In scientific research in areas such as genomics and astronomy, where the data sizes are in petabytes, the situation above can also occur.

The de facto computational approach in dealing with such issue is through distributed computing systems. In a distributed computing system, many computers called *nodes* are connected together to form a *cluster*. Traditionally, distributed computing systems have been very specialised to be computation centric or storage centric. An example of computation on a specialised distributed system is the Message Passing Interface (MPI) type distributed systems, which has been the traditional way of performing distributed computing in scientific computing. Some storage centric distributed systems such as SAN have been outlined in earlier section. Other examples of storage centric distributed systems include Google File System (GFS) (Ghemawat, Gobioff, and Leung, 2003) and Google's BigTable (Chang et al., 2006). By combining many machines these systems have very large storage capacities. GFS has storage capacity of over 300 terabytes (1 terabyte is 1,000 gigabytes) and Google's BigTable has capacity in the scale of petabytes (1,000 terabytes or 1 million gigabytes). Another advantage of using many computers is that these systems can continue to operate even when some nodes fail or encounter error.

The large storage capabilities and high-availability of distributed systems however, have two major downsides. Firstly, there is performance bottleneck due to interconnectivity nature of the nodes. Earlier section allude to the characteristic that data transfer from disk is slower than data transfer from memory. Data transfer across computer network is generally even slower than disk. As a result, in distributed computing systems, data transfer between the

nodes is a major performance bottleneck. Scott and Blocker (2013) state '*A fact which is well known to parallel computing experts, but often surprising to novices: passing messages [data] among a large number of machines is expensive, regardless of the size of the messages being passed*'. In many of these systems, this performance bottleneck occurs because to perform computation, data are moved to the nodes that will perform the computation.

The second major disadvantage is that since a distributed system need to manage many nodes (multiple computers), developing software to run on a distributed computing system is much more complex, particularly dealing with situations where a node fails. A famous list in software development titled *Fallacies of Distributed Computing* outlines the key assumptions that a distributed computing software needs to address (Rotem-Gal-Oz, Arnon, n.d.).

However, a new approach to distributed system has emerged in recent years which addresses these limitations. In this new approach, which take advantage of the computational abilities of ordinary desktop computers, the nodes perform both computation and storage. Since the nodes are general purpose, they can be made up of commodity hardware (off the shelf, ordinary desktop computers), as opposed to specialised hardware. In conjunction with general purpose nodes, these new distributed computing systems also utilise software abstractions [7], that allow software to be developed much more easily. Two well know distributed computing systems with this new approach are Hadoop (The Apache Software Foundation, n.d.[a]) and Spark (The Apache Software Foundation, n.d.[d]). The combination of non-specialised nodes and software abstraction allow these new system to have a completely opposite computational model to traditional systems, where the computation code goes to the node with the data. Since the size of computation code is much smaller compared to data used for analysis, the impact of performance bottleneck due to data transfer is greatly reduced. Although new approaches to distributed computing systems make performing statistical analysis on very large data sets more feasible, they are nevertheless complicated to setup and use. In addition, the division of data across many nodes fundamentally challenges the assumption of many statistical algorithms that expect all the data to be available in memory of a *single* computer.

A popular optimisation technique used in machine learning application is Stochastic Gradient Descent (SGD) (Kiefer and Wolfowitz, 1952; Robbins and Monro, 1951). Stochastic gradient descent is an iterative algorithm that performs optimisation (finding maximum or minimum of a function) based on the first derivative of a function. In each iteration, instead of estimating the gradient using all the data, only a single observation is used; hence the stochastic

[7] Software abstractions capture the essential characteristics of data structures or computational operations. They are abstract in the sense of being generic, which allows their underlying concepts to be applied in different contexts. E.g., MapReduce, which will be discussed later is an abstraction that encapsulates the operations of Map, which applies the same operation across many subsets of data, and Reduce, which combines the results of Map.

nature of this technique. Since each iteration does not process over all the data, stochastic gradient descent is a popular optimisation algorithm (Dean et al., 2012). A variation on stochastic gradient descent, called batch stochastic gradient descent, estimates the gradient in each iteration using a group of observations rather than a single observation. Eve though the gradient update in each iteration is based on a single observation or a group of observations, all the data eventually may be processed (depending on the convergence). Since data transfer in distributed computing systems can result in degradation of computational performance, alternative methods by Dean et al. (2012) and McMahan et al. (2017) attempt to extend batch stochastic gradient descent to distributed computing systems by performing gradient estimation in parallel. In both works, each shard of the distributed data is considered as a batch of data. Dean et al. (2012) address the issue of keeping track of the gradient estimates from the nodes of the distributed computing system across the iterations of batch stochastic gradient descent. McMahan et al. (2017) extend the idea of distributed data into the context of mobile phones such that data on a user's mobile phone is considered as a batch of data. The mobile phones then perform stochastic gradient descent in a parallel fashion and the overall gradient across all the data is estimated by averaging the gradients from each mobile phone.

1.3 Outline of thesis

The aim of this thesis is to explore ways to performing statistical analysis in the context of distributed systems, specifically, in cases where the data are distributed. In particular, explore the way existing statistical procedures can be applied to these distributed data setting.

Chapter 2 begins by exploring the concept of divide and recombine (D&R), which is a natural fit to modern distributed computing system such as Hadoop and Spark. The aim of Chapter 2 is to provide an overview of modern distributed computing systems and show that the way data stored in these systems require different approach to performing statistical analysis. In particular, the Divide and Recombine (D&R) approach. In D&R, computations are performed on each unit of distributed data (also known as *shards*, *subsets* or *blocks*), then the results are combined. Although this approach is computational, it differs from other techniques such as swapping in that computations are performed in parallel across the nodes, hence taking advantage of the computational power of distributed computing systems.

By the end of Chapter 2, it'll become clear that even with divide and recombine paradigm, and MapReduce abstraction, performing statistical analysis on distributed computing systems is not a trivial task. Crucially, the performance bottleneck in distributed computing systems means being able to process all the data on a single computer does has an appeal.

Since data transfer is a source of performance bottleneck, and many existing statistical

procedures don't work in distributed computing settings, the aim of Chapter 3 is to allow perform data reduction on the distributed data so that statistical analysis can be performed on a single desktop machine. The techniques used for data reduction are statistical techniques such as coarsening, as opposed to computational techniques. The bonus is that since coarsening is a simple operation, it can be executed in parallel in a distributed computing system. Statistical analysis is then performed on the reduced data set. To recover the complete data model, EM algorithm is used to perform statistical inference. By using existing statistical techniques, an analyst can apply these techniques without the need for deep understanding of distributed computing systems.

We can expand the definition of distributed data such that we can see how existing statistical techniques have similar ideas. In particular, instead of dividing data arbitrarily, it is possible to divide the data in particular ways to take advantage of existing statistical methods. In Chapter 4, with an understanding of D&R, data can be divided in a particular way so that the distributed data mimics independent studies of traditional meta-analysis. In this chapter, techniques from biostatistics such as propensity scoring, multiple imputation and random effect meta-analysis are used to perform statistical analysis on distributed data. The techniques are demonstrated using longitudinal child growth data from Healthy birth, growth and development knolwedge integration (HBGDki) funded by Bill and Melinda Gates foundation.

Related to the HBGDki data set, Chapter 5 explores the key issue of appropriate smoothing parameter of penalised splines when modelling the derivatives of the data. Here, given the large data size of HBGDki, it is time consuming to numerically create the derivative curves. A model-based approach can speed up computational time, therefore it is preferred. The question is whether using the smoothing parameter of the data is good enough to model the derivatives.

2 Performing statistical analysis on distributed data using Divide and Recombine

2.1 Motivation

Many implementations of statistical algorithms assume that the data are accessible from the memory of a single computer. This assumption does not hold in situations where the size of data is really large e.g., petabytes scale data (1 petabyte = 1,000 terabytes, 1 terabyte = 1,000 gigabytes). To store large datasets, organisations use distributed computing systems, which combine many computers together, as opposed to storing data on a single computer. Traditionally, distributed computing systems are highly specialised and it is difficult to develop software utilising them, However, modern distributed computing systems such as Hadoop and Spark allows both storage and computation to be done using off the shelf computers. In addition, programming abstraction such as MapReduce allows software to be developed easily. Many existing statistical algorithms however, assume data are in a single location (the memory of a single computer), as opposed to the data being distributed (across many computers). Data processing paradigm such as Divide and Recombine (D&R) enables statistical inference algorithms to be developed in MapReduce fashion. This chapter shows how by structuring statistical inference algorithm using D&R, statistical inference can be performed on distributed data. As an example, logistic regression is implemented using D&R and applied to a large dataset. However, processing data on distributed computing systems remain a challenge. MapReduce and D& are more suitable for algorithms that are embarrassingly parallel, and not all algorithms can be applied to distributed data. One method to address this problem is to reduce the data residing on the distributed computing system and then execute the algorithm using the reduced data set on a single computer. In this chapter, a particular data reduction method is shown. This method is applicable to situations where the covariates are all categorical. Data reduction techniques with minimal information loss may allow many types of statistical inference algorithms to be used in conjunction with D&R on distributed data.

2.2 Introduction

An implicit assumption made by many existing statistical algorithms is that the data they operate on are available in a single location, typically the memory of the computer. In cases where data sizes are very large, e.g., this assumption can fail. The most obvious solution in these cases is to increase the memory capacity of the computer. Another possible solution is to utilise some form of *swapping* technique where blocks of data are moved between the

memory and other storage mechanisms to create an illusion that all the data are in memory. However, at some point due to every increasing data sizes, these single computer solutions become infeasible. One can imagine companies such as Google and Dropbox do not store all their data on a single computer.

A very popular method of dealing with very large data sets is to form distributed computing clusters, which is made up of many computers (or computing related devices). Traditionally this setup can be seen in the area of data storage. Organisations typically use some form of distributed storage systems, such as multiple server class computers each having multiple hard drives set up in Redundant Array of Independent Disks (RAID) configuration. Storing data across many nodes not only allow very large data sets to be stored, this setup also allows organisations to cope with data growth since more nodes can be added to the system to increase storage. To allow computation to be performed, these distributed storage systems are then combined with very powerful computing setups typically found in High Performance Computing facilities. However, since storage and computation are separate, data have to be moved between these systems. The steps to perform computation first consist of coping data from the storage system into computers of the high performance computing facility. Computations are performed on these computers. The results of the computation are then copied back into the storage system. In contrast, recent popular distributed computing systems such as Hadoop (The Apache Software Foundation, n.d.[a]) and Spark (The Apache Software Foundation, n.d.[d]) take a different approach. In these systems all the computers (nodes) perform dual role of storage and computation. When performing computation, instead of copying data to the nodes, computation code are send to the nodes. Since computation code are generally much smaller than data, this the problem of computing performance bottleneck due to movement of data is alleviated (Scott and Blocker, 2013).

However, there is still the complication of managing these systems. Traditionally, distributed computing systems are difficult to program because the analyst has to write software not only to perform data analysis, code need to be written to manage the underlying distributed computing system (e.g., what to do when a node fails). In order to ease the development of software on distributed computing systems, in particular the model of Hadoop where the nodes store units of distributed data that are sometimes called *shards* (Dean et al., 2012; Scott and Blocker, 2013), programming abstraction such as MapReduce was developed (Dean and Ghemawat, 2010; Dean and Ghemawat, 2004; Somers, 2018).

The MapReduce programming abstraction can be viewed as a specific implementation of a general idea of Divide and Recombine (D&R) (Cleveland and Hafen, 2014), where a large data is first divided, then some operations are performed on each shard, followed by combination of all the results. Divide and recombine paradigm is naturally suited to newer distributed

systems such as Hadoop since it explicitly addresses the division of a large data set into shards that are distributed across the nodes.

In this chapter, we begin by providing a brief overview of divide and recombine. We then outline MapReduce programming abstraction in Section 2.3.1, and its role in distributed computing system such as Hadoop. Section 2.4 explores Hadoop and shows how abstractions such as MapReduce are a naturally fit for the performing analysis on distributed data on systems such as Hadoop. In Section 2.5, we fit logistic regression model using various implementations of divide and recombine.

2.3 Divide and Recombine (D&R) Data Processing Model

A strategy for performing statistical analysis on distributed computing systems is through the use of Divide and Recombine (D&R) (Cleveland and Hafen, 2014), which is particularly suited for very large data sets where the number of observations greatly exceeds the number of variables (Hafen, 2016). In D&R, the data are first *purposefully* divided into *subsets*, with each subset small enough in size to be analysed with standard software on a single computer. After performing analysis on individual subsets, the results are then combined to '*yield a statistically valid—although not always exact—result*' (Hafen, 2016). As an example, consider the linear regression model

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}\,, \tag{2.1}$$

where for n observations, $\mathbf{Y}$ is the $n \times 1$ vector of the responses, $\mathbf{X}$ is the $n \times p$ design matrix with p covariates, $\boldsymbol{\beta}$ is the $p \times 1$ vector containing the regression coefficients, and $\boldsymbol{\epsilon}$ is the $n \times 1$ vector of errors that are from a Gaussian distribution with zero mean and σ^2 variance, i.e., for the ith error $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. The ordinary least square estimate of the regression coefficients, $\hat{\boldsymbol{\beta}}$ can be obtained by

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}\,. \tag{2.2}$$

Many existing implementation of linear regression algorithm assume that the entire design matrix $\mathbf{X}$ is stored in the memory of a single computer. Let's demonstrate how a divide and recombine strategy can be used to estimate the parameters $\widehat{\boldsymbol{\beta}}$. To a large dataset on a distributed computing system, the data are first divided into S subsets. A divide and recombine method does not necessary require estimating the regression coefficients in each subset. Instead, we can view it as calculating some subset specific terms that can be recombined to obtain an overall estimate. One idea is to think in terms of using *the sufficiency principle*. For a parameter θ, Casella and Berger (2002) define sufficient statistics $T(\mathbf{X})$ as "*a* sufficient

statistics for θ if the conditional distribution of the sample $\mathbf{X}$ given the value of $T(\mathbf{X})$ does not depend on θ" (Casella and Berger, 2002, Section 6.2.1). Lee, Brown, and Ryan (2017) performs data reduction by applying the sufficiency principle where a sample-based summary statistic captures all the information about the parameters. As a result, any inference about the parameters depends on the samples only through the sufficient statistics. In cases where sufficient statistics can be used, statistical inference can be performed without the need to process over all the individual observations of a dataset. For the kth subset, denoting $\mathbf{Y}_k$ as the vector of responses and $\mathbf{X}_k$ as the design matrix of the kth subset, if we can obtain an estimate for the kth subset as

$$\widehat{\beta}_k = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{Y}_k \,,$$

we can combine the estimates $\widehat{\boldsymbol{\beta}}_k$ to obtain $\widehat{\boldsymbol{\beta}}$. The calculation inside each subset as shown above yields the Maximum Likelihood equivalent of $\widehat{\boldsymbol{\beta}}$. Sufficient statistics is generally limited to exponential distributions in the GLM case. However, recently Huggins, Adams, and Broderick (2017) developed a polynomial approximation of sufficient statistics. Guha et al. (2012) show one method of combining of the estimates from the subsets

$$\widehat{\boldsymbol{\beta}} = \Big(\sum_{k=1}^{S} \mathbf{X}_k^\top \mathbf{X}_k \Big)^{-1} \sum_{k=1}^{S} \mathbf{X}_k^\top \mathbf{Y}_k \,, \tag{2.3}$$

where, $\mathbf{X}_k^\top \mathbf{X}_k$ and $\mathbf{X}_k^\top \mathbf{Y}_k$ are calculated for each kth subset, and then combined together to a similar form shown in (2.2). Another divide and recombine method is to first estimate the coefficients for each subset $\widehat{\boldsymbol{\beta}}_k$ using least squares method shown in (2.2) such that

$$\widehat{\boldsymbol{\beta}}_k = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{Y}_k \,, \tag{2.4}$$

for $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_S\}$ subsets, then average the subsets coefficients. That is, for the regression estimate of a single variable, $\widehat{\beta}$ is simply the average of the estimators from S subsets (Guha et al., 2012)

$$\widehat{\beta}_{\text{DnR}} = \frac{1}{S} \sum_{k=1}^{S} \widehat{\beta}_k \,. \tag{2.5}$$

In general, the average of sub-set regression coefficient estimates as shown in (2.5) is not equivalent to the coefficient estimate that can be obtained when all the data can be analysed at once (2.2). However, with weak law of large numbers, it can be argued that (2.5) is statistically consistent with (2.2). That is, $\widehat{\beta}_{\text{DnR}} \to \widehat{\beta}$. See Wasserman (2004, Chapter 5) for weak law of large numbers. A more general form of (2.5) is the weighted average of the estimates from

the subsets. The estimate for a single parameter, in the weighted average form is

$$\widehat{\beta}_{\text{DnR}} = \Big(\sum_{k=1}^{S} w_k \hat{\beta}_k \Big) \Big/ \Big(\sum_{k=1}^{S} w_k \Big) , \tag{2.6}$$

where w_k is weight for the kth subset. The estimator shown in (2.5) is a special case with $w_k = 1$. An alternative form of weighting scheme is the inverse of the variance of the estimator in the subset, similar to meta-analysis, i.e., $w_k = 1 \,/\, \text{Var}(\hat{\beta}_k)$. Instead of combining the estimates of regression coefficients from the individual subsets, McMahan et al. (2017) combine the *cost functions* of the subsets in their work on the concept of *federated learning*. They show that assuming the errors terms are Gaussian distributed, the estimate of the parameters shown in (2.2) and (2.5) minimise the same *cost function*, in this case a square loss. See 2.F for details on how the cost functions are combined. The variance for estimator (2.5), as shown in Guha et al. (2012) is

$$\begin{aligned}
\text{Cov}(\hat{\boldsymbol{\beta}}_{\text{DnR}}) &= \text{Cov}\left[\frac{1}{S} \sum_{k=1}^{S} (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{Y}_k \right] \\
&= \frac{1}{S^2} \text{Cov}\Big[\sum_{k=1}^{S} (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{Y}_k \Big] \quad \text{, treating } S \text{ as a constant,} \\
&= \frac{1}{S^2} \sum_{k=1}^{S} \text{Cov}\Big[(\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{Y}_k \Big] \quad \text{, assuming independence in } \mathbf{Y}_k \,.
\end{aligned} \tag{2.7}$$

Using a property of covariance matrix from Ruppert, Wand, and Carroll (2003a, (A.6))

$$\text{Cov}(\mathbf{A}\mathbf{x} + \mathbf{c}) = \mathbf{A} \, \text{Cov}(\mathbf{x}) \, \mathbf{A}^\top ,$$

where $\mathbf{A}$ is a constant matrix, $\mathbf{c}$ is a constant vector, and $\mathbf{x}$ is a random vector. Assigning $\mathbf{A} = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top$, in (2.7),

$$\text{Cov}(\hat{\boldsymbol{\beta}}_{\text{DnR}}) = \frac{1}{S^2} \sum_{k=1}^{S} (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \, \text{Cov}(\mathbf{Y}_k) \, \mathbf{X}_k (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} . \tag{2.8}$$

$\text{Cov}(\mathbf{Y}_k) = \sigma^2 \mathbf{I}$, when we assume σ^2 are all the same for all the observations in the regression model, and are not correlated. In addition assuming $\sigma^2 = 1$ as in Guha et al. (2012), (2.8) then becomes

$$\begin{aligned}\mathrm{Cov}(\hat{\boldsymbol{\beta}}_{\mathrm{DnR}}) &= \frac{1}{S^2}\sum_{k=1}^{S}\sigma^2(\mathbf{X}_k^\top\mathbf{X}_k)^{-1} \\ &= \frac{1}{S^2}\sum_{k=1}^{S}(\mathbf{X}_k^\top\mathbf{X}_k)^{-1}\,. \end{aligned} \tag{2.9}$$

With this (inverse variance) weighting scheme, the general form of variance is

$$\mathrm{Var}(\hat{\beta}_{\mathrm{DnR}}) = \Big[1\Big/\Big(\sum_{k=1}^{S} w_k\Big)^2\Big]\Big[\sum_{k=1}^{S} w_k^2\mathrm{Var}(\hat{\beta}_k)\Big]\,. \tag{2.10}$$

Substituting $w_k = 1$ into (2.10) we get (2.9). If we use the inverse variance weighting scheme substituting the weights $w_k = 1\,/\,\mathrm{Var}(\hat{\beta}_k)$ into (2.10),

$$\begin{aligned}\mathrm{Var}(\hat{\beta}_{\mathrm{DnR}}) &= \Big[1\Big/\Big\{\sum_{k=1}^{S} 1/\mathrm{Var}(\hat{\beta}_k)\Big\}^2\Big]\Big[\sum_{k=1}^{S}\{1/\mathrm{Var}(\hat{\beta}_k)\}^2\,\mathrm{Var}(\hat{\beta}_k)\Big] \\ &= \Big[1\Big/\Big\{\sum_{k=1}^{S} 1/\mathrm{Var}(\hat{\beta}_k)\Big\}^2\Big]\Big[\sum_{k=1}^{S} 1/\mathrm{Var}(\hat{\beta}_k)\Big] \\ &= 1\Big/\Big[\sum_{k=1}^{S} 1/\mathrm{Var}(\hat{\beta}_k)\Big] \\ &= 1\Big/\Big[\sum_{k=1}^{S} w_k\Big]\,. \end{aligned}$$

The divide and recombine strategy is just one of many strategies for developing algorithms to be run on distributed computing systems. Its model of divide and recombine closely matches the operation of distributed computing systems. The data in the each of the subsets can be thought of as the data stored in the nodes of a cluster. Since divide and recombine is a high level strategy, it can be implemented in many contexts. In the context of distributed computing systems such as Hadoop and Spark, one implementation is through MapReduce.

2.3.1 MapReduce

Developing software for distributed computing systems is more difficult compared with developing for a single computer. Rotem-Gal-Oz, Arnon (n.d.) explains the major challenges that are inherited in distributed computing systems, such as network reliability, network latency, security, and transport cost. Handling these issues is non-trivial. For example, to han-

dle failure of the node or network (i.e., the hardware of the node failed), the software, first of all, has some sort of criteria determining what constitutes a failure and mechanisms to verify it. One mechanism to detect failures is to constantly poll the node then waiting for the node to respond within a specific period of time, where the waiting time often depends on the type and amount of traffic on the network. Once a failure of a node has been detected, the software has to manage this failure without stopping the analysis. When performing analysis using very large datasets, it's not desirable for the software to terminate the entire analysis and restart again. One way to handle failure of a node is to find another node to process the data, which there are many ways to do so. A statistical analyst will then have to write software that performs statistical analysis as well as handle issues arising from distributed computing systems. One way to reduce the workload on the analyst is to separate the analysis tasks from the distributed computing related tasks. One such example is the concept of MapReduce.

Inspired by concepts from functional programming languages such as Lisp, Dean and Ghemawat (2008) at Google proposed the *MapReduce* programming abstraction to reduce the complexity of developing applications on distributed computing systems. In MapReduce, computations are performed by two functions: *Map* and *Reduce*. The Map function is executed over individual blocks of the data (shards or subsets), and the Reduce function is used to combine the outputs from the Map function. The concept of the Map function is not new. In R, the Map function is analogous to using the `apply` family of functions (e.g., `lapply`), whereby a function is executed over many elements of a data structure such as a vector or list. Recent packages such as `purrr` (Henry and Wickham, 2018) also has a family of `map` functions. The Reduce part is the function that performs some form of recombination operation on the results from the Map function. The R code in Listing 1 demonstrates a simple MapReduce code by attempting to find the maximum values of a list of vectors. In this code listing, the Map part consists of applying the built-in function `max()` to the individual vectors of a list named `data_list`. The result of Map is a list (`max_inner`) with 2 elements, with each element of the list contains the largest number of each vector. In the Reduce stage, the `max()` function is executed again on the result from the Map stage (`max_inner`) to return the largest value across both vectors.

Listing 1: *Finding maximum values of 2 vectors using lapply() function. The apply() family of functions can be considered as equivalent of the Map function. While the Reduce function is just some form of aggregation that is applied to the results of the Map function.*

```
vec_1 <- c(1,2,3)
vec_2 <- c(4,2,1)
data_list <- list(vec_1, vec_2)

# Our Map function is R's built-in 'max' function
```

```
# The Map function is applied to each element of 'data_list'
max_inner <- lapply(X = data_list, FUN = max)

# This is our Reduce equivalent.
# Combine and process the outputs of lapply()
max(unlist(max_inner))
```

What makes MapReduce combination interesting is that Dean and Ghemawat (2008) applied the Map and Reduce functions to large scale distributed computing system setting. In this setting, instead of a vector or list storing the data, individual nodes store the data, with the nodes representing the individual elements of a vector or list. Executing Map function over a vector or list then becomes getting the nodes to execute the Map function using the data they have. As a result, MapReduce is a natural fit to distributed computing systems.

MapReduce can be seen as one implementation of the divide and recombine paradigm. By providing a consistent way of encapsulating computation for the divide and re-combine phases via Map and Reduce functions, when developing statistical software for distributed computing systems, an analyst can isolate the analysis parts from computing parts (Somers, 2018).

Another aspect of standardisation provided by MapReduce that is not obvious is the format of the data between the Map and Reduce stages. In a single computer setting shown in Listing 1, the the `lapply` function expects a list as its input and outputs a list as well. The data structure MapReduce uses to send data between its two stage is *Key-Value pairs*.

If MapReduce standardises the computational workflow of a distributed computing system, key-value pairs standardise the data structures to pass data across nodes of a distributed computing system. As the name suggests, a key-value pair data structure has two components, the *Key*, which is '*the part of a group of data by which it [data] is sorted, indexed, cross referenced, etc.*' (Black and Pieterse, 2004), and the *Value*, which is the data indexed by the Key. An example of a key-value pair to model a patient in a hospital might use the Patient ID number as the *Key*, with the *Value* holding all the data regarding that particular patient. As in the case with MapReduce, although key-value pair data structure is not unique to MapReduce (the *named list* data structure in R has the same structure), it is a data structure that can be applied to large scale distributed computing system.

As shown in Figure 2.1, key-value pairs can be further qualified according to their usage in a MapReduce workflow to emphasis their roles. For instance, the key-value pairs containing outputs from Map function can be called *intermediate key-value pairs*, to emphasise that they are not the final result of the workflow. While the final output, which is the result of Reduce stage can be called *output key-value pair*.

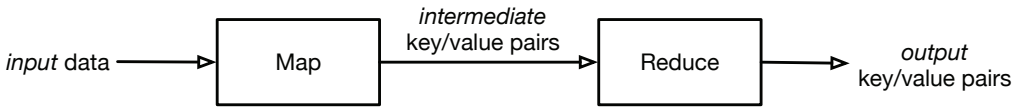


Figure 2.1: *Key-value pairs can be qualified depending on their roles in a MapReduce data flow.*

2.3.2 Example of MapReduce workflow

We now make the MapReduce concept more concrete through an example. In this example, suppose a very large data set containing the customer number and their payment for a good or service, has been distributed across two nodes, as illustrated in Figure 2.2. Suppose we want to calculate the amount paid by each customer. The first aspect to note is that each node holds some portion of the data from each customer. In this example we assume the original data set was divided arbitrarily. Even though Figure 2.2 looks complicated, we only needs to perform three tasks: determine our key-value pairs, develop the Map and Reduce functions.

In this example, let's assume the key-value pair is in the form `(Cust_Num, Amount)`. Since we are interested in obtaining the total amount paid by each customer, our goal with the Map stage is to output intermediate key-value pairs (the data that will go to the Reduce stage) containing the total amount for each customer. As shown in Figure 2.2, our Map function simply groups all the data by customer number. In this particular design, we are moving the majority of the computation to the Reduce function. Sometimes we can reduce the workload of the Reduce function by performing some rudimentary tasks in the *Combine* function. In Figure 2.2, instead of getting Reduce function to aggregate all the observations of a particular customer, we reduce the data also summing up each customer's total payment amount in the *Combine* function. The Combine function is useful because sometimes the Map function may not be able to perform all the operations we want since it needs to deal with user input data which are not homogeneous. In addition, by performing some aggregation in Combine, we reduce the amount of data that needs to be transferred for the Reduce function.

We then create the Reduce function, which simply performs an aggregation. Note that the Reduce function only receives data that are grouped by one unique Key. In Figure 2.2, the Reduce function in node 1 only receives key-value pairs related to customer 1, and likewise node 2 only receives key-value pairs related to customer 2. In distributed computing systems that supports MapReduce, a step called *Shuffle and Sort/Merge* is executed which ensures each Reduce function receives key-value pair data that are indexed by a single unique Key.

We can see how MapReduce implements the Divide and Recombine paradigm. In the

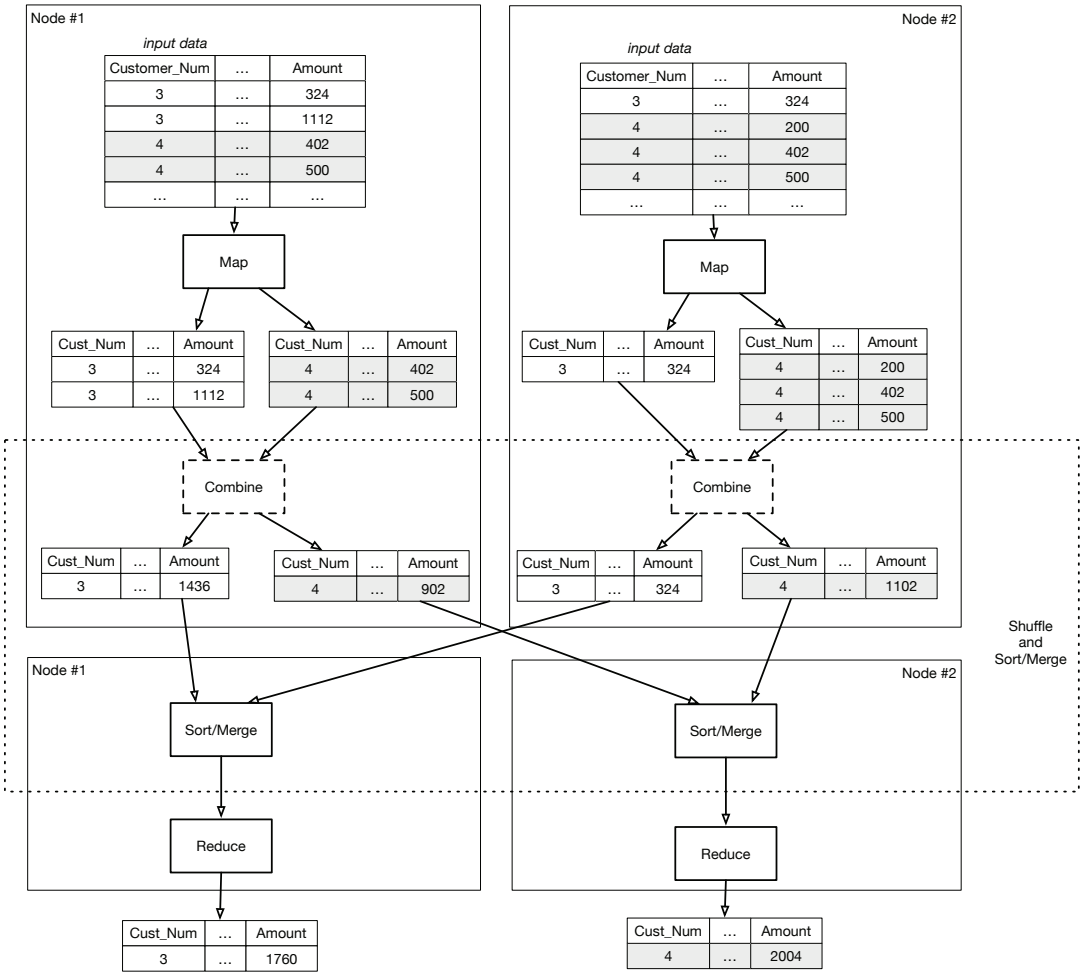


Figure 2.2: *Detailed MapReduce workflow to calculate a sum*

Divide stage, the Map function simply groups and sums the total amount for each customer within each shard (subset) of the distributed data. In the Recombine stage, the Reduce simply aggregate all the amounts per customer.

By standardising the computational (Map and Reduce functions) and data structure (Key-Value pairs), the MapReduce programming abstraction can be implemented across a variety of distributed computing architectures to carry out Divide and Recombine. For example, in the *Mars* system, MapReduce is implemented on GPUs (He et al., 2008). Likewise, in the Phoenix system, MapReduce is implemented to utilise the multi-core capabilities of a single machine (Yoo, Romano, and Kozyrakis, 2009).

2.4 Using Hadoop to execute the distributed algorithm

A popular distributed computing systems that implements the MapReduce programming abstraction is Hadoop (The Apache Software Foundation, n.d.[a]). Hadoop is a distributed computing system that is designed to execute software written using MapReduce programming abstraction. Unlike more traditional distributed computing systems such as High Performance Computing (HPC) clusters, where the nodes consists of server class computers that are computational centric, the nodes of a Hadoop cluster can consist of commodity desktop computers that perform dual role of computation and storage.

Historically, Hadoop began as an open source implementation of Google's distributed storage system called Google File System (GFS) (Ghemawat, Gobioff, and Leung, 2003). Unlike other large scale distributed storage systems at that time, the nodes that make up GFS system consisted of commodity desktop computers. Since GFS was a file system, it could not perform computation on the data. Google later proposed the MapReduce programming abstraction as a method to perform computation on large scale distributed storage system like GFS (White, 2015).

We now explore how Hadoop implements GFS and MapReduce. Hadoop's open source implementation of GFS is the Hadoop Distributed File System (HDFS). HDF stores data by first dividing a data set into *blocks* (in general, we also have called these units of distributed data as *shards*), which are then distributed across the nodes of the Hadoop cluster. These blocks are also replicated across the nodes so that there are multiple copies of a block, which allows data to be available when the node storing them experience a hardware failure. If we consider Hadoop solely by its storage component, the only major difference it has over traditional storage system is the use of commodity hardware, since traditional storage systems also can provide high availability large scale storage.

The way computation is performed on Hadoop is how it sets apart from traditional distributed systems. Hadoop uses MapReduce programming paradigm to perform computation across its nodes. In a MapReduce workflow, the nodes in the Hadoop cluster execute the Map function on their block of data. Since a large data set can be divided into many blocks, a single Map function can be executed in parallel across many nodes. Upon completion of the Map function, Hadoop first selects nodes to execute the Reduce function, then transfer the results of the Map function (intermediate key-value pairs) to these Reducer nodes.

The computation model of Hadoop described above is to move code to data. In contrast, in many traditional distributed computing systems, data move to the computational centric nodes, since the nodes do not store data. By moving computational code, which are generally be much smaller in size compared to data (The Apache Software Foundation, n.d.[e]),

Hadoop greatly reduce the data movement between the nodes, which is a major source of performance bottleneck.

To illustrate and contrast the data movement between a traditional distributed system and Hadoop, consider Scenario B we outlined in the Introduction of the thesis. This is the scenario where the data are too large to fit in the memory of a single computer, but can be stored in the computer's non-memory storage, such as hard drive or solid state drive. An organisation typically will store this data in its specialised data storage system and make the data available via a network shared drive. When we run any analysis using this data in R for example, this data will be copied across the network into the memory of our computer. This is an example of moving data to code. In this computational model, we can encounter two problems. First, copying the data across the network can be time consuming, and also takes up the network resource. Second, since the data is too large, our computer will run out of memory (or with swapping, we incur more performance penalty).

In contrast, if the data is stored on a Hadoop cluster, we only need to send our Map and Reduce functions to the cluster. This situation has been shown in Figure 2.2, where since the data for the customers are already stored on the nodes, we can get the nodes to execute code on their blocks of customer data, then combine the results. An additional benefit of Hadoop's model is that there is a separation between the computation for analysis and the management of the distributed system. In Hadoop, analysis code only needs to be written in MapReduce abstraction. The analyst is not required to write code that performs tasks such as scheduling nodes to become the Reducer, or what to do in the event of a node failure. Hadoop software takes care of all the distributed computing aspects.

Since Hadoop is open source there are many *distributions* of Hadoop that can be used. This is similar to GNU/Linux where there are many distributions such as Red Hat, Debian, Ubuntu, etc. The major distribution open source distribution of Hadoop from The Apache Software Foundation is simply called Apache Hadoop distribution. Commercial Hadoop distributions include Cloudera (Cloudera Inc., n.d.), Hortonworks (*Horton Works*,), and MapR (*MapR*,). The commercial distributions provide additional tools and enterprise support for Hadoop, such as tools to process log files from Hadoop. A very popular option to trial Hadoop or to create large Hadoop cluster is by using cloud computing services. A *service* is a computation capability provided by a third party organisation. A example of a service is an Internet Service Provider (ISP) providing access to the Internet. Two popular cloud services for Hadoop are Amazon Web Services (AWS) Elastic MapReduce (EMR), and Microsoft Azure. An example of using cloud computing is The New York Times which used AWS to convert its archived articles between 1851–1922 to PDF format (Gottfrid, 2007).

Since Hadoop was developed using Java programming language, MapReduce software

written in other languages such as R, use the the generic *Hadoop Streaming* (The Apache Software Foundation, n.d.[f]), which is available in most Hadoop distributions. In Hadoop Streaming environment, MapReduce software are executed as though they are command line programs. As a result, any command line programs that can act as Map and Reduce functions can be execute in Hadoop Streaming environment. The downside to using Hadoop Streaming is that programs need to act like a command line application, which can have structure compared to analysis code written in programming language such as R. As an alternative, software libraries such *DeltaRho*(The Department of Statistics, n.d.) can be used. DeltaRho allows analysis code to be developed in the same manner as other analysis code written in R, and it also hides Hadoop Streaming from its users. For example, the *distributed data frame* (DDF) object in DeltaRho is very similar to the `data.frame` data structure, which is familiar to many users of R. Using DeltaRho, users are no longer restricted to Hadoop Streaming model of text-only input and output. Users can instead accept and return distributed data frames as if they are working on an ordinary `data.frame` objects. Other alternatives to work with Hadoop include Hadoop specific scripting languages such as Apache Pig (The Apache Software Foundation, n.d.[c]) and Apache Hive (The Apache Software Foundation, n.d.[b]). The former allows users to perform data manipulation without writing raw MapReduce code. The latter allows queries on the data to be made in a similar manner to Structured Query Language (SQL) for relational databases. Scripting language such as Apache Hive allows database users to use Hadoop without the need to know details such as MapReduce.

Commercial statistical software that are widely used today such as SAS and MATLAB supports Hadoop (SAS Institute Inc, n.d.; The MathWorks, Inc., n.d.). MATLAB provides a number of ways to work with Hadoop. One mode of operation is to perform swapping and execute MapReduce programs on a single computer through data structure such as `Tall Array` (The MathWorks, n.d.). Another mode is to run MATLAB code using Hadoop Streaming (The MathWorks, Inc., n.d.).

2.4.1 Conditional Data Division

As discussed in previously, the Hadoop Distributed File System (HDFS) divides a large data set into data *blocks*. In Hadoop, data are divided based on the size of an individual data block (e.g., 64MB), as opposed to division based on the nature of the dataset, such as the number of rows. Although this this method of division is natural for a computer file system, it is arbitrary and not very useful from the viewpoint of statistical analysis. To provide more user friendly methods of data division, software packages such as DeltaRho provides functions to perform different ways of dividing the data which are more statistically relevant. These include ran-

domised data division, and *Conditional Data Division*. The latter involves dividing a large data set conditional on some variables of the data set (i.e., data are grouped together based on that variable). For example, our analyst might divide a large data set conditional on the year it was collected.

Note that the method of data division involves a trade-off between computational efficiency and user friendliness. Hadoop's default method of data division, based on the file size, is more computationally efficient since data can be divided as it's being loaded into the HDFS. In contrast, with DeltaRho's data division, an extra step is performed after the data has been loaded (divided using file size). This extra step typically involves executing a MapReduce computational job.

2.5 Application

2.6 Results

The codes representing the various ways of fitting the logistic regression models are shown in Table 2.1. In this table, 'Single Machine' denotes fitting the regression model on a single computer. 'Ordinary logistic regression' refers to the model (**??**) where log-likelihoods for each individual observation are calculated then aggregated. 'Group variable' refers to the model (**??**) whereby log-likelihood is calculated for unique combination of the categorical covariates. 'DnR' denotes Divide and Recombine approach implemented using `DeltaRho` package. Logistic regression models are fitted using `glm` function of R. The hardware for the 'Single Machine' implementation consists of a server class computer that has a 3.47GHz Intel Xeon W3690 (6 Cores) CPU, 12GB of memory, and 2 × 1TB 7,200 RPM SATA II hard drives configured as an RAID array for non-memory (secondary) storage. For Hadoop implementation, the cluster consists of one *master node* and two *core nodes*, with the core nodes performing the actual storage and computation. The master node and the core nodes are physically located on the same server rack. The master node is a server with 2×2.6GHz Intel Xeon E5-2697 v3 (14 Cores), 128GB memory and 2× 900GB 10,000 RPM SAS II Hard Drives in RAID array configuration. The two core nodes have the same specification as the hardware for the single machine. All the computers ran Red Hat Linux Enterprise Edition (64-bit) with R version 3.2.0.

Table 2.2 and 2.3 illustrate the estimates of the coefficients using the data set for the year 2013, using estimator shown in (2.5). Since the hardware we used is a server class computer with large amount of memory, the entire data set can be stored in memory. The objective of these table is then to demonstrate that estimates of the coefficients using divide and recombine approach are similar to that performing ordinary logistic regression.

Implementation	Notation
Single Machine. Ordinary logistic regression	T1
Single Machine. Grouped variable	T2
Single Machine. DnR.	T3
Single Machine. DnR Grouped Variable	T4
Hadoop. DnR	T5
Hadoop. DnR Grouped Variable	T6

Table 2.1: *Different implementations of logistic regression.*

	Intercept		$\hat{\beta}_1$		$\hat{\beta}_2$	
Impl.	Est.	Std.Err	Est.	Std.Err	Est.	Std.Err
T1	2.666	0.00662	-0.505	0.00894	-0.717	0.00758
T2	2.666	0.007	-0.505	0.009	-0.717	0.008
T3	2.666	0.00662	-0.505	0.00895	-0.717	0.00759
T4	2.666	0.007	-0.505	0.009	-0.717	0.008
T5	2.667	0.00662	-0.505	0.00895	-0.718	0.00759
T6	2.666	0.007	-0.505	0.009	-0.717	0.008

Table 2.2: *Estimate of regression coefficients for 2013 dataset.*

	$\hat{\beta}_3$		$\hat{\beta}_4$		$\hat{\beta}_5$	
Impl.	Est.	Std.Err	Est.	Std.Err	Est.	Std.Err
T1	-1.195	0.00700	-1.465	0.00692	-0.284	0.00346
T2	-1.195	0.007	-1.465	0.007	-0.284	0.003
T3	-1.196	0.00701	-1.465	0.00692	-0.284	0.00346
T4	-1.195	0.007	-1.465	0.007	-0.284	0.003
T5	-1.196	0.00701	-1.466	0.00692	-0.284	0.00346
T6	-1.195	0.007	-1.465	0.007	-0.284	0.003

Table 2.3: *Estimate of regression coefficients for 2013 dataset (continued).*

2.6.1 Execution Timing for Different Implementations of Logistic Regression

To demonstrate the limitations of using single computer to perform analysis on very large datasets, we attempt to fit logistic regression with different computation setup using increasingly large data sizes. There are five datasets in total, starting with with NY State SPARC dataset from the year 2010 to 2012. To simulate very large data set, the next two datasets consist of the combined data from year 2010 to 2012 resampled at two and four times.

Figure 2.3 shows the execution times (transformed to log base 10) of fitting two different logistic regression models on increasingly large data sets. The solid lines (——) represent the execution times of implementations that used the grouped variable method, likelihood from

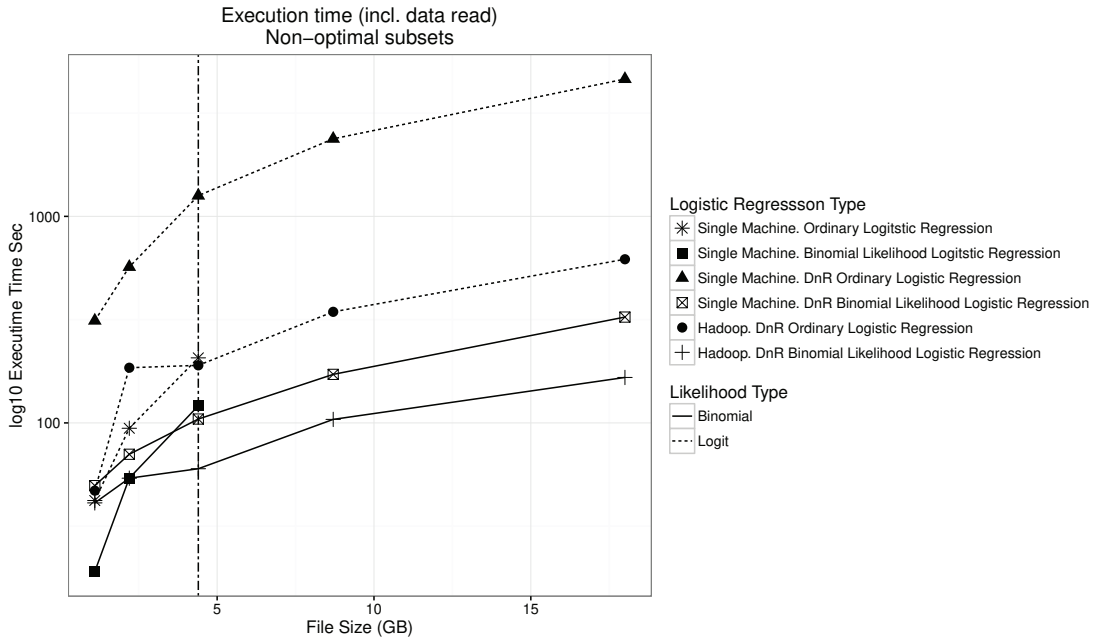


Figure 2.3: *Execution times of fitting logistic regression with different computation setup across increasingly large datasets.*

(**??**). The dotted lines (·····) represent execution times for implementations with likelihood in the form (**??**), where calculation is performed over individual observations.

There are some interesting features shown in Figure 2.3. Firstly, fitting logistic regression using the likelihood form in (**??**) generally require lower execution times compared to fitting logistic regression using the model in (**??**). By calculating the likelihood over the unique combination of the levels of categorical covariates, the amount of data that is transferred between the nodes is reduced. In addition within each node, the data required for the calculation may be small enough to be all stored in memory. Since analysing data that is stored in memory is faster than analysing data stored in non-memory storage, the execution times are expected to be lower. An interesting note is that it may be possible to group the levels of categorical variables to the point where all the data can be analysed on a single computer.

The second interesting feature in Figure 2.3, is the vertical line indicating the data size limit for the single computer setup. Beyond this data size, the computer we use no longer is able to fit the regression model. All hope is not lost entirely however.

The third interesting feature is the benefit of computational technique such as swapping in allowing single computer setup to perform analysis on very large data sets. In Figure 2.3, the triangle line shows how processing large data sets that cannot be stored in the memory of a single machine is possible with the divide and recombine approach. Since data is divided

into subsets in the divide and recombine approach, analysis is performed on subsets that are much smaller than the entire data set. The trade-off however, is performance penalty for swapping the data of each subset between the memory and disk. In Figure 2.3, this is shown by long execution times of the model.

Another feature of note is the appropriate data size when using a distributed computing system such as Hadoop. Distributed computing systems such as Hadoop was designed to process very large data sets. As a result, attempting to perform analysis on data sizes that can be store on a single computer setting can result in performance degradation since the distributed system need to move results of Map to Reduce stage. This is shown in the variability of performance time of Hadoop implementation with smaller data sizes.

2.7 Discussion

In this chapter we show how performing statistical analysis on very large data is possible through the use of distributed computing systems. New type of distributed computing system such as Hadoop which leverages on MapReduce programming abstraction allows software to for distributed computing systems to be developed with ease. The divide and recombine paradigm provides a way perform statistical analysis on new distributed systems such as Hadoop via MapReduce. By dividing a very large data set into smaller manageable subsets, performing operation on these subsets, then combining their results, divide and recombine paradigm can be naturally implemented using MapReduce on new distributed computing systems such as Hadoop. To make these ideas more concrete, we demonstrate different ways of fitting logistic regression across different computing setup and data sizes.

In our demonstration we show the limit a single computer setup when fitting a regression model. At a certain data size, the software simply ran out of memory. Nevertheless we can use the divide and recombine implementation of software such as `DeltaRho` to fit regression model on larger data sizes. We demonstrate performing statistical analysis using divide and recombine approach on distributed computing system such as Hadoop have no problem with regards to the data size.

Since divide and recombine is a generic concept, we can leverage existing statistical techniques to perform analysis on large data sets. In our demonstration, by calculating the likelihoods based on unique combination of the levels of categorical covariates, we reduced the original data size which enables faster computation. Since using data that are entirely stored in a computer memory is faster, and performing computation on a single computer does not require data transfer across computer networks, we can take the data reduction idea further by attempting to reduce the large data set to a size where the analysis is entirely possible on a

single computer. There are a number of advantages to this data reduction approach. Firstly, the data reduction we have used is fairly to implement on a distributed computing system since we are obtaining counts of successes and failures for each unique combination of levels of categorical variables. This can be implemented in MapReduce where the Map stage involves performing the count on each shard of distributed data, and the Reduce stage aggregating the results. The second advantage, apart from computational performance is that by reducing the data to be able to store on a single computer, we have satisfied the implicit assumption made by existing statistical software that all the data being available at one location. Hence we are able to use many existing software.

There are existing statistical techniques that are similar to divide and recombine approach. One such technique is meta-analysis, where results from many studies are combined to obtain a combined estimate. In Chapter 5, meta-analysis is used to estimate a combined effect, using data from studies with different covariate characteristics. By regarding the shards of distributed data as studies, we can apply techniques such as random effect meta-analysis in the context of distributed computing systems.

2.7.1 Special Divide and Recombine Algorithms

Recently there are statistical techniques that uses the divide and recombine approach to analyse distributed data. Examples of these new techniques are Bags of Little Bootstraps (Kleiner et al., 2014), and Consensus Monte Carlo (Scott and Blocker, 2013). The first example, Bags of Little Bootstrap technique attempts to resolve the problem of performing bootstrapping on very large datasets. The bagging part of this technique creates classification model by averaging (or through majority vote) the results from many classifiers that are trained using bootstrap samples. Traditionally when generating the bootstrap samples, each set of bootstrap sample have the same number of observations (denoted n) as the original data set. This approach is however not feasible when the original data set is very large. The memory and network bandwidth requirements of these large bootstrap samples can become too large. In the Bags of Little Bootstraps technique, instead of directly bootstrapping from the original data set, we bootstrap from subsamples of the original data set. This technique effectively divide the traditional bootstrap procedure into two steps consisting of subsampling and sampling. Instead of obtaining bootstrap samples from the original data set of size n, s subsamples with size b, where $b < n$ are first obtained. For sth subsample, bootstrapping via sampling with replacement n times is then performed. The quality assessment of an estimator of interest is then calculated using these newly bootstrapped samples within sth subsample. Bagging is then performed on quality assessments from *all* the subsamples. In a distributed comput-

ing system that supports MapReduce abstraction, if the data are distributed randomly, we can treat the shards (units of distributed data) as a subsample. Our Map function then performs the bootstrapping and calculation of the quality assessment for the particular shard (subsample). The Reduce function than performs bagging procedure by combining the quality assessments from all the shards (subsamples).

Another example of divide and recombine approach is the Consensus Monte Carlo (Scott and Blocker, 2013), where a large data set is first divided into shards. Inference using Monte Carlo method is then performed on each shard, resulting in posterior distribution for each shard. Assuming these posterior distributions from the shards are Gaussian, they are them combined as a weighted average.

2.7.2 Stochastic Gradient Descent with Divide and Recombine

Stochastic gradient descent (SGD) (Kiefer and Wolfowitz, 1952; Robbins and Monro, 1951) is an optimisation algorithm commonly used in machine learning environments (Dean et al., 2012). In stochastic gradient descent, gradient of a function is calculated, at a minimum, with one observation. This is in contrast to traditional gradient calculation based on Newton's method, where all the observations are used to calculate the gradient for a single step. Even though stochastic gradient descent is used in many areas due to its online nature (online in the sense of processing one observation at a time), with very large data sets, stochastic gradient descent become impractical (Dean et al., 2012). As a result, there have been attempts to apply stochastic gradient descent to distributed data settings. One such example is Downpour SGD by Google (Dean et al., 2012). In Downpour SGD, each node of a distributed computing system calculates the gradient for its assigned parameters and data. A central machine called the *Parameter Server* receives gradient updates from the nodes and a new iteration of the parameters is calculated. The new parameters are then fetch by the nodes before they begin their calculation. One notably feature is that the Parameter Server does not wait for all the nodes to complete their calculations before starting the next iteration of the algorithm. The major reason for not waiting for all the nodes to complete their calculations is to avoid the Parameter Server from becoming delayed when the nodes experience a failure (Dean et al., 2012). This approach is called Asynchronous Stochastic Gradient Descent (Async-SGD) (Chen et al., 2016). One major downside with asynchronous approach is that since nodes can complete their calculations at different times, in each iterations, it is possible for some nodes to be begin their calculations based on out of date parameters value (Chen et al., 2016).

An alternative approach is Synchronous Stochastic Gradient Descent (Sync-SGD) (Chen

et al., 2016), where iterations of the parameter updates are moved forward only after the estimates from *all* the nodes have been collected by a central machine. To overcome the problem of node failures, extra nodes are used (with some performing the same calculation as other nodes), but their results are discarded at each iteration (Chen et al., 2016). The main down side is the inefficiency due to the use of extra nodes, as well as performance cost resulting from copying data to the extra nodes.

2.7.3 Apache Spark

In recent years, Apache Spark(The Apache Software Foundation, n.d.[d]) has become the preferred distributed computing system, chiefly due to its computational performance. Similar to Hadoop, Apache Spark is a computational engine (Karau et al., 2015, p. 2) that performs analyses on distributed data. However, unlike Hadoop, the nodes in an Apache Spark cluster can store results of their analyses in memory. In contrast, in a Hadoop cluster, the nodes store their results into to the hadoop distributed file system. As mentioned in previous sections, performing analysis on data that are stored in the memory of a computer is faster than with data that are stored on disk. As a result, iterative algorithms such as stochastic gradient descent benefits from running on a system like Apache Spark since the parameter estimates of each iteration can be stored in memory rather than on disks of the distributed file system. Since stochastic gradient descent is widely used in machine learning algorithms, distributed computing systems with high performance stochastic gradient descent are highly desirable.

In addition to computational performance, Apache Spark has native support more modern computing languages such as R that are data analysis centric. Hadoop in contrast was developed before R became popular, hence it only support Java natively. In Hadoop, MapReduce programs written in programming languages other than Java are executed through hadoop streaming environment. Although hadoop streaming environment is flexible in the sense that it allows MapReduce programs to be written in many different types of programming languages, it severely constrains the type of data types represented in the code. For example, when working with R, the main data type that holds data is `data.frame`. Typically in R programs, data from data source such as comma separated values (CSV) file are first loaded into a `data.frame` before analysis is carried out. R MapReduce programs written for hadoop streaming environment, on the other hand, does not have a data structure to represent the shards of the data stored on the node; data are instead treated as a stream of text input, which the R program is expected to read into a `data.frame`. In order to use some form of abstraction that represent the shards of data, external libraries such as `DeltaRho` has to be used. Apache Spark, on the other hand, natively have data abstraction for languages such as R. As a result,

Apache Spark is more accessible to the statistical community. The combined result is that the analyst can develop algorithms much more easily in Apache Spark compared with Hadoop. This in turn has lead to more libraries being developed for Apache Spark. The `sparklyr` software package (Luraschi et al., 2019) for R is such an example.

Regardless, note that since Apache Spark performs analyses on distributed data, analysis approaches that are applicable to Hadoop, such as divide and recombine, and programming abstractions such as MapReduce are also applicable to Apache Spark.

2.7.4 Future research

This chapter explores how statistical inference in regression models can be performed on distributed computing systems via D&R paradigm. Since linear regression model shown in (2.1) has assumptions with regards to the errors $\boldsymbol{\epsilon}$, regression diagnostic is typically performed to check how well the fitted regression model follow the assumptions. A key component of regression diagnostic is visual examination of plots based on the residuals $e_i = y_i - \widehat{y}_i$. A popular method of checking homoscedasticity, $\mathrm{E}(\boldsymbol{\epsilon}) = \mathbf{0}$ and $\mathrm{var}(\epsilon_i) = \sigma^2$, for $1 \leqslant i \leqslant n$ for n is visual examination of a residual plot. A residual plot consists of standardised residuals on the y-axis and fitted values $\widehat{y}_i$, on the x-axis. The check for homoscedasticity is performed by visually examining the spread of the standardised residuals. Although the statistics for the residual plot can be calculated in a big data setting, there may be too many data points on the residual plot which can make visual inspection difficult.

Likewise, to check the assumption of the independence of errors terms, or they are uncorrelated, the autocorrelation between the residuals between different lags is calculated and plotted (Ruppert, Wand, and Carroll, 2003b, Sec 2.3.2). As before, although it is possible to calculate the autocorrelation for large datasets in a distributed computing system, the autocorrelation plot may contain too many data points.

Performing more complex statistical analysis using large datasets remains a challenge, hence there are many opportunities for further research. One such area is fittingă longitudinal models (sometimes referred to as hierarchical or mixed models) on large datasets. Unlike the regression model shown in this chapter where the observations y_i are independent, a longitudinal model allows repeated observations of a subject over time. Longitudinal model also allows for the incorporation of *random effects* that are specific to the individual subjects. These random effects are modelled as additional regression coefficients in the longitudinal model. Other regression coefficients are then considered as fixed effects, which are shared by all the subjects Fitzmaurice, Laird, and Ware, 2011, Sec 1.5. Using existing method, assuming all the data can be stored on the memory of a single computer, the fixed effects are estimated

using generalised least squares (GLS) method (Fitzmaurice, Laird, and Ware, 2011, Section 4.2; Ruppert, Wand, and Carroll, 2003b, Section 4.5), and the random effects are predicted using the best unbiased linear predictor (BLUP) (Fitzmaurice, Laird, and Ware, 2011, Section 8.6; Ruppert, Wand, and Carroll, 2003b, Section 4.3). The variances are estimated using the restricted maximum likelihood (REML) method(Ruppert, Wand, and Carroll, 2003b, Section 4.5.4). It can be seen that a dataset that has a large number of subjects and many measurements on each subject, can become quite large, such that it cannot be stored in the memory of a single computer. For instance, a fitness device company that collects health data such as heart rate of its users. In this kind of scenario, it is unclear how the methods for parameter estimation and prediction, such as BLUP, GLS and REML can work when they do not have all the data. What is also unclear is how these methods can work when the data are distributed, such as when they are stored in a system such as Hadoop.

One possible method is to utilise the 'two-stage' random effects modelling technique, a variant of it is known as 'NIH method'(Fitzmaurice, Laird, and Ware, 2011). Although this technique has some restrictions and was used before the advent of linear mixed model, it nevertheless provide a way to analyse distributed longitudinal datasets. In the two-stage technique, the first stage consists of regressing the outcome variable on time-varying covariates for individual subjects. In the second stage, the subject-specific coefficients (and the intercept term) from the first stage are modelled as random variables. Each subject-specific coefficient is then regressed on the time-invariant covariates. We can see how the two-stage approach can work on distributed data with divide and recombine. For the first stage, we fist divide the data by subjects, then fit the subject-specific regression model. For the second stage, we then perform a regression of subject-specific coefficients on time-invariant covariates in divide and recombine fashion.

Appendix 2.A Introduction to Hadoop Streaming

Since Hadoop was written in the Java programming language, MapReduce code written in Java can seamlessly work with Hadoop. However, MapReduce code written in other programming languages such as R or Python has to be written to use *Hadoop Streaming* mode. In Hadoop Streaming mode, the Map and Reduce functions are treated as command line scripts that Hadoop will run as part of its workflow. As a result, the Map and Reduce functions have to interact with the 3 Standard *Input/Output (I/O) streams* which are communication channels for command line programs. These three streams are

Input Stream The channel which Map and Reduce functions receive their inputs. For command line programs, the default input stream is the input from the command line.

Output Stream Map and Reduce functions write their outputs to this channel. For command line programs, the default output stream is the command line itself.

Error Stream Similar to the Output Stream, the Map and Reduce functions can write error messages to this channel, although doing so is not mandatory however. In many cases, the default Error Stream is the Output Stream.

Although this sounds complex, by treating non-Java MapReduce functions as command line scripts, Hadoop can accept code written in many programming languages. An analyst can also write her MapReduce code in Python, or any other language, including scripting languages such as Perl, as long as her code are command line programs that can communicate using three Standard Input/Output (I/O) streams described above.

Appendix 2.B Minimal Map function

Listing 3 in Appendix 2.E.2 shows the minimal code for a Map function written in R, for Hadoop Streaming mode. As mentioned earlier, in Hadoop Streaming mode, the Map and Reduce functions are treated as command line programs. So we'll begin by setting up the Map function as a command line program. Line 1 specifies the `Rscript` program to be the command line *interpreter*. `RScript` is a program included in default installation of R. It allows R code to be treated as *non-interactive* (e.g., no user interactions) command line scripts that can be executed directly from command line (without needing to launch R) (Smith, 2009). Line 1 indicate that the Map function as a command line program that can be interpreted using (executed by) `RScript`. Line 4 suppresses warning messages because Hadoop expects the Map function script to only output key-value pairs.

After setting up the interpreter for the Map function, we'll now set up the Standard I/O streams so that our Map function can read and write data. We'll accept the default Standard Output and Standard Error streams, which are the command line itself. That is, any output from our Map function goes to the command line. For Standard Input Stream, we need to make our Map function uses it for input because Hadoop will supply the input data through the Standard Input Stream. Line 8 accomplishes this by creating a connection to the Standard Input Stream. This set-up is different to traditional R analysis code where we explicitly read in the data from a file (e.g., using `read.csv()` function). In the case of Hadoop, instead of reading all the data in a single operation, the Map function reads lines of text data from the Standard Input Stream.

After the initialisation of the Map function, we now read data from the Standard Input Stream *read line by line* and process them. The Map function expects each line of data from the Standard Input Stream to be a valid datum. For example, the header line of a CSV dataset cannot be passed to the Map function since it doesn't contain data. The `while` loop enclosing line 15 and line 32 performs the data reading. The code between line 27 and line 29 stop reading the data when there are no more data left to receive. Analysis on the data begins after all the data have been received.

After the completion of the analysis, we'll write out the results to the Standard Output Stream as shown on line 43. Since the output of the Map function will become input to the Reduce function, the result is written out as key-value pairs. In this template, the Key and Values are separated by the `TAB` character. The Reduce function will use the `TAB` character to separate the Key and Value.

Appendix 2.C Minimal Reduce function

Due to the nature of Hadoop Streaming, the Reduce function is more complex compared to the Map function. Listing 4 in Appendix 2.E.3 shows the minimal R code for a Reduce function written to use Hadoop Streaming.

Just like the Map function, the first few lines (lines 1-5) perform initialisation such as setting `Rscript` as the interpreter and creating a connection to the Standard Input Stream. Also like the Map function, the Reduce function also receives data line by line. The code inside the `while` loop, starting from line 19 performs the data reading process. Unlike the Map function, the Reduce function receives data in Key-Value pair format. In addition, Hadoop have sorted the intermediate key-value pairs before sending them the Reduce function (Section 2.D).

As a consequence of using Hadoop Streaming, the Reduce function is more complex. In particular in Hadoop Streaming, the Reduce function receives *all* the key-value pairs from

Map functions. Whereas when developing MapReduce function in Hadoop native programming language such as Java, the Reduce function only receives data indexed by a single key. Since the Reduce function receives all the key-value pairs, it has to decide which key-value pair to process. Part of this process is performing *key boundary detection.*

Key boundary detection involves detecting when data from a new key arrives from the Standard Input Stream. For example, in Figure 2.4, if the Reduce function is only needed to process data for `key_1`, it needs to detect the change in the Key starting from the 4th line.

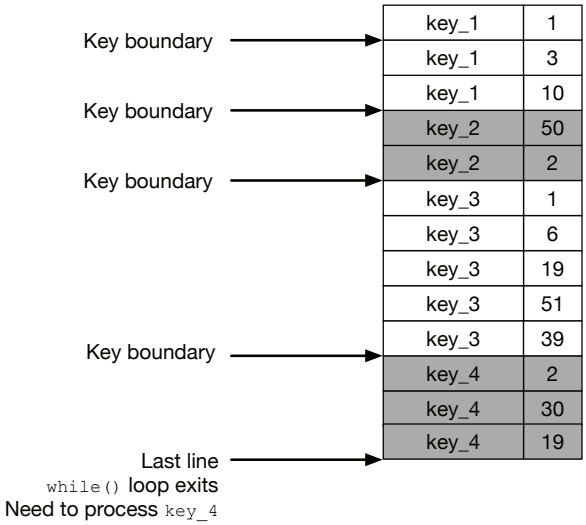


Figure 2.4: *Hadoop Streaming Key Boundary Detection*

To detect a key boundary, we'll first store the *last* key processed in a variable, as shown on line 10. Later in the Reduce function (line 42), this is compared against the latest key read from its Standard Input Stream. If the keys are not the same, then we assume we have received all the data related to the last key. We can then proceed to run our analysis on the data from last key.

We also need to implement a variation of the key boundary detection process near the end of the Reduce function. Instead of detecting a new Key, this time the Reduce function needs to detect the end of input data and then process data for its last key. For example, in Figure 2.4, the key-values for `key_4` constitute the final set of input data. As a result, the values collected for `key_4`, which are `(2, 30, 19)` need to be processed before the termination of the Reduce function. If the Reduce function does not perform this, it'll never process the data of the last key. The code between lines 73-79 perform this crucial step.

After we've set up the data reading loop, we can proceed to perform analysis on the data and output the results. The minimal Reduce function in Listing 4 outputs lines of text with the word `result`. Unlike the output from the Map function, there are no technical restrictions on the output format of the Reduce function. However, it is generally a good idea to output key-value pairs like the Map function because the output might be used as inputs to another Hadoop job.

It needs to be emphasised that the Reduce function is much more complex because we are using Hadoop Streaming. We need to use Hadoop Streaming because R is not a native language of Hadoop. If the Reduce function is written the native language of Hadoop, Java, tasks such as key boundary detection are not necessary. In addition, the Reduce function written in Java receives data for a specific Key, as opposed to all the data. This complexity motivates the use of software libraries such as DeltaRho (The Department of Statistics, n.d.).

Appendix 2.D Combine, shuffle and sort

When writing the Reduce function that we have made an assumption about the order of key-value pairs. Specifically, the keys are ordered as illustrated in Figure 2.4. As mentioned in Section 2.3.2, Hadoop performs two important steps after the execution of the Map functions, *Combine*, and *Shuffle and Sort*.

The *Combine* step performs an additional aggregation after the Map function. The purpose of Combine is to further reduce the amount of data to send to the the Reduce function. Hadoop does allows its users to supply their custom functions to perform this aggregation. In Figure 2.2, the `Amount` values for each Key (`Cust_Num`) are further aggregated in the Combine stage before being passed to the Reduce function. *Shuffle and Sort* is a crucial step that the make sure the Reduce function receives data sorted by Key. In Figure 2.2, the Reduce function at the bottom left panel processes data for `Cust_Num` 3, while the other Reduce function processes data for `Cust_Num` 4.

Apart from making sure that each Reducer obtains the correct data (as discussed earlier, this is only for MapReduce code written in Java), from Figure 2.2, it can be seen how computational parallelism is achieved due to the distributed nature of the input data. The input data are divided and stored on 2 different nodes. Since Map function applies to all the nodes, Hadoop Hadoop will execute two Map functions *in parallel*. For the Reduce function, since there are two *unique* Keys (`Cust_Num` 3 and `Cust_Num` 4), Hadoop will also schedule two Reduce functions to execute in parallel.

Note that we only need to write our Map and Reduce functions. The task of scheduling nodes to execute the MapReduce functions and transfer of outputs from Map functions to

the Reduce functions, are all handled by Hadoop. The separation and abstraction allows us to concentrate on our analysis code.

Appendix 2.E Hadoop Streaming Code Listing

2.E.1 Data for Testing

Listing 2: *test_data.csv*

```
3       , 1         , 324
3       , 1         , 1112
4       , 2         , 402
4       , 1         , 500
3       , 2         , 324
4       , 1         , 200
4       , 2         , 402
4       , 3         , 500
```

2.E.2 Minimal Map Function

Listing 3: *min_mapper.R*

```
#! /usr/bin/env Rscript

# Ignore warning messages. The mapper should only output key-value parirs.
options(warn=-1)

# Initialising a variable that can read from Standard Input Stream
# Input to mapper will come from the Standard Input stream
input <- file("stdin", "r")

# Variable that stores the number of lines read in the last iteration
# of while() loop.
# Starting from value of '1' to start the loop
numLinesReadBefore <- 1

while(numLinesReadBefore > 0) {
        # Reading a line of text from Standard Input Stream
        currentLine <- readLines(input, n=1, warn=FALSE)

        # Gets the number of lines read
        numLinesRead <- length(currentLine)

```

```
    # Keep track of the number of lines that was just read
        numLinesReadBefore <- numLinesRead

    # If we have read in one line, process it.
    # Do not process the line if it is end of input.
    if (numLinesRead < 1) {
        break
    }

        # Perform some processing here.
}

# Completed reading data. Closing the Standard Input Stream.
close(input)

# If you have not perform the analysis, do it here.


# Analysis completed.
# Output Key-Value pairs for Reduce.
# In this case, Key and Value are separate by a special TAB character
cat("key", "\t", "value", "\n", sep="")
```

2.E.3 Minimal Reduce Function

Listing 4: *min_reducer.R*

```
#! /usr/bin/env Rscript

# Initialising a variable to read from Standard Input Stream, just like
# the Map function.
input <- file("stdin", "r")

# Variable to track the current 'key'
# Initialsing to a known value that is not used in the data. This allows
# the Reduce function to know when it is reading data for a new 'key'.
mapper.key.prev <- -1

# Loop to process data one line at a time.
# We'll stop the loop where we read a blank/empty line.

# Variable that tracks the length of the current line.
# It starts from 1 to allow the while() loop below to run once.
```

```
numLinesReadBefore <- 1

while(numLinesReadBefore > 0) {

  # Reading a line from Standard Input Stream
  currentLine <- readLines(input, n=1, warn=FALSE)

  # Updating the length of data just read.
  numLinesRead <- length(currentLine)

  numLinesReadBefore <- numLinesRead

  if (numLinesRead < 1) {
    break
  }

  # Extracting the 'key' and 'value' parts of 'currentLine'.
  # The Map function separated
  # the 'key' and 'value' with a TAB character.
  inter.kv <- unlist(strsplit(currentLine, "\t"))
  inter.key <- inter.kv[1]
  inter.val <- inter.kv[2]

  # Performs Reduce function, there are 2 possibilities at this point.
  # Either the code just read data for a new 'key', or it just read
  # data for an existing 'key'.
  if (identical(mapper.key.prev, -1) ||
      identical(mapper.key.prev, inter.key)) {

    # Collecting data so they can be put into a data structure such as
    # 'data.frame'.

  } else {

    # The if() statement returned FALSE, which means data for a new 'key'
    # was just read. Executes the Reduce function on data for current
    # 'key', and output the results.

    # Execute Reduce function here...

    # Output the result.
    cat("result", "\n", sep="")

    # Resetting any variables used for collecting data earlier.
```

```
61
62    }
63
64    # Stores the current key for processing of next line.
65    mapper.key.prev <- inter.key
66
67  }
68
69  # Processing the data for the last 'key'. This is necessary because the
70  # code above executes the Reduce function only when a new 'key' is
71  # detected. Since all the data has been read, there won't be any new
72  # 'key'. As a result, the Reduce function was not executed. The
73  # following code executes the Reduce function for the last 'key' that
74  # was read.
75  if(!identical(mapper.key.prev, -1)) {
76
77      # Execute Reduce function here...
78
79      # Output the result.
80      cat("result", "\n", sep="")
81  }
82
83  # Closeing the Standard Input Stream.
84  close(input)
```

Appendix 2.F Combining of cost function in federated learning

Instead of combining the estimates of regression coefficients McMahan et al. (2017) combine the score equations from the distributed data. First, recognise that as the name (least square estimate) suggests, the parameter estimate in matrix notation shown in (2.2), is the solution that minimises a square loss function. In the non-divide and recombine case, assume there are N observations. For the ith observation, y_i is the outcome variable for the ith observation. With a single covariate x_i is the covariate for the ith observation. For the linear model $y_i = \beta_0 + \beta_1 x_i$, the loss function for the ith observation is

$$\mathcal{L}_i\{y_i, f(x_i; \beta_0, \beta_1)\} = \{y_i - f(x_i; \beta_0, \beta_1)\}^2 , \tag{2.11}$$

where $f(x_i; \beta_0, \beta_1) = \beta_0 + \beta_1 x_i$. The loss function for N observations is then

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i\{y_i, f(x_i; \beta_0, \beta_1)\} . \tag{2.12}$$

The parameters β_0, β_1 are obtained by minimising (2.12)

$$\underset{\beta_0,\beta_1}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i\{y_i, f(x_i; \beta_0, \beta_1)\} \ . \tag{2.13}$$

If N observations are then divided into S subsets randomly, with n_s observations per sth subset such that $\sum_{j \in S} n_j = N$, the loss function for the sth subset is (2.11) and (2.12) applied to the subset

$$\begin{aligned} \mathcal{L}_s &= \frac{1}{n_s} \sum_{j=1}^{n_s} \mathcal{L}_j\{y_j, f(x_j; \beta_{0,j}, \beta_{1,j})\} \\ &= \frac{1}{n_s} \sum_{j=1}^{n_s} \{y_j - f(x_j; \beta_{0,j}, \beta_{1,j})\}^2 \ . \end{aligned} \tag{2.14}$$

McMahan et al. (2017) combine the loss functions of the subsets to match the non-D&R loss function as follows

$$\begin{aligned} \sum_{k=1}^{S} \left(\frac{n_k}{N} \mathcal{L}_k\right) &= \sum_{k=1}^{S} \left\{\frac{n_k}{N} \left(\frac{1}{n_k} \sum_{j=1}^{n_k} \mathcal{L}_j\right)\right\} \\ &= \sum_{k=1}^{S} \left\{\frac{1}{N} \sum_{j=1}^{n_k} \mathcal{L}_j\right\} \\ &= \frac{1}{N} \sum_{k=1}^{S} \left\{\sum_{j=1}^{n_k} \mathcal{L}_j\right\} \\ &= \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i \quad \text{since} \sum_{K} n_k = N \ . \end{aligned} \tag{2.15}$$

As a result, the same loss function is minimised, as in (2.13), to obtain the estimates of the parameters β_0, β_1. Note that (2.15) is a specific form of weighted combination, a topic that is discussed later.

3 Covariate discretisation for the analysis of Big Data[8]

3.1 Summary

Distributed computing systems are often used to store and process large datasets. By combining the resources of many computers, such systems alleviate the storage and computational limit of a single computer. Although distributed computing systems such as Hadoop and Spark allow statistical analysis to be performed on arbitrarily large datasets, the communication between the computers that make up these systems, called nodes, can become a source of performance bottleneck. In this work, we outline a novel combination of statistical and computational techniques that enables logistic regression to be performed using a large data set on a standard desktop computer. We achieve this firstly by coarsening a large data set then aggregating the coarsened data so that they are small enough to be used in statistical analysis on a single desktop computer. However, performing analysis using coarsened data can introduce biases in the results of the analysis. To address this, we use the Expectation-Maximisation (EM) algorithm to recover the complete (non-coarsened) data model. Our work draws on methods for the analysis of data involving coarsened covariates using EM by methods of weights. We explore different coarsening strategies (e.g., rounding and quintile) and discuss how our methods can scale to very large data through simulations, we find our method works especially well when data are coarsened to cover a wide interval, where there are more loss of information. Compared with naïvely using the coarsened data, our method is able to estimate regression coefficients with less bias.

3.2 Introduction

Data that are too large for analysis on a desktop computer are often stored and analysed using distributed computing systems such as Hadoop (White, 2012). In these systems, data are divided into smaller data sets, which are then stored on the individual computers that make up the distributed computing system. Since the data are spread across different computers, performing statistical analysis using existing techniques and software can become a challenge, because these techniques implicitly assume that the data are available at a single location (typically in the memory of the computer). Even when using analysis techniques that work on distributed data such as Divide and Recombine (Guha et al., 2012), there is a performance bottleneck resulting from the data transfer between the nodes of the computer system (Scott et al., 2016, p79). To improve the performance time of performing statistical

[8] Dr. Stephen T. Wright, formerly from the Australia Red Cross Blood service provided advice on the statistical analysis as part of his research fellow position at UTS.

analysis on distributed data, we propose a statistical data reduction and information recovery method based on missing data methods such as coarsening and Expectation Maximisation (EM) algorithm (Dempster, Laird, and Rubin, 1977). We provide an overview of our method in this section and defer the details to later sections. We reduce the large data by coarsening and aggregating the large data set so that the reduced data small enough for regression analysis to be performed using a desktop computer. However, using the reduced data set can introduce bias in the estimates of the regression coefficients. To reduce this bias, we use the EM algorithm to recover the complete data model. Our method is a novel combination of distributed computing system and statistical methods. Computationally, the coarsening and aggregation operations are not difficult to implement on a distributed computing system. By performing regression analysis outside of the distributed computing system, we do not need to write specialised regression software. Statistically, in our approach we *deliberately* coarsen the data, as opposed to treating coarsened data as a problem. We can do this because we utilise techniques from the field of missing data, such as EM algorithm, to recover the complete data model. The method in this chapter is applicable to cases where there are many more observations than the covariates. We assume that data division are based on observations (row-based), as opposed to division by covariate (column-based) which can occur in scenarios involving 'wide-data'.

Heitjan and Rubin (1991) refer to coarsened data as "data are neither entirely missing nor perfectly present. Instead, we observe only a subset of the complete-data sample space in which the true, unobservable data lie" (Heitjan and Rubin, 1991). For example, if we perform rounding operation on the values of a covariate, we observe the rounded value; the value of the covariate itself is not missing but we only observe the rounded value. In medical and health applications, coarsened data is often referred to as *interval-censored* data (Lipsitz et al., 2004, p280). An example of interval-censored data is recording the age ranges of subjects as instead of the actual ages. Similar ideas in computing are bin aggregation in data in visualisation of large data sets (Scheidegger, 2016), and the use of summary data known as *synopses* in computer database systems (Cormode, 2011; Liu, Jiang, and Heer, 2013). In this work, coarsening refers to the definition by Heitjan and Rubin (1991).

After coarsening, the second part of our data reduction involves aggregating the covariates, which we can perform if the covariates and the outcome variable are all categorical. In this situation, we can aggregate the data by counting the number of observations belonging to each unique combination of the observed data and the outcome variable. We can naïvely use the coarsened (and aggregated) data directly in our regression model. Karmel and Polasek (1970) is an example of a common approach to naïvely using the coarsened data, which treats the mid-points of the coarsened intervals as a proxy to the complete data (as cited by Kim and

Hong (2012)). However, naïvely using the coarsened and aggregated data in statistical analysis such as regression can introduce bias into the estimates of the parameters. Hsiao (1983) shown that using the mid-points of coarsened intervals result in biased estimates (as cited by Kim and Hong (2012)). One way to avoid using the coarsened data directly is to use a method such as such as the Expectation Maximisation (EM) algorithm, where we impute the plausible true data within the coarsened intervals. By using the EM algorithm, we attempt to reduce the bias in our estimates of the regression coefficients. In this work, our coarsening mechanism is a deterministic function, such as the ceiling function. The general case is where the coarsening mechanism is stochastic, i.e., where the coarsened values are random variables. Kim and Hong (2012) show how statistical inference can be performed in the general case.

We present our work in the following way. In Section 3.3 we first outline the notation and key terminologies. We use a very small sample of NYC Yellow Taxi data from New York City's Taxi and Limousine Commission (TLC), to illustrate key operations such as coarsening and aggregation. The data contain variables related to taxi trips taken in NYC such as pick up date and time, drop off date and time, trip distance, passenger count, fees and tips received by the driver. In this work we perform logistic regression with our outcome variable being whether the driver received a tip or not, and the two covariates are trip distance and the payment type. The covariate we coarsen is trip distance. We use data for December 2010, which as a CSV data file, contain 13,819,322 observations and occupies approximately 2.5 GB of disk space. In addition to notation, in Section 3.3 we also present our logistic regression model and the estimation of the parameters, beginning with the complete data case, then proceeding to coarsened and aggregated data. In section 3.4 we show how to estimate the standard errors for the estimates from the EM algorithm. After the presentation of the logistic regression model, we perform simulations in Section 3.5. We then apply our method to the NYC taxi data in Section 3.6, followed by a discussion in Section 3.7.

3.3 Statistical model

In this work, vectors are *column* vectors and they are shown with bold lower case Roman font, e.g., $\boldsymbol{y}$. For our logistic regression with n responses (observations) and m predictors (covariates), $\boldsymbol{y}$ is the $n \times 1$ vector of outcome variables, $\boldsymbol{x}$ is the $n \times 1$ vector for the continuous covariate we coarsen, and $\boldsymbol{Z}$ is a $n \times p$ matrix contains all the other covariates that we do not coarsen, which also are not $\boldsymbol{x}$. The total number of covariates is $p + 1 = m$.

With the above notation, our large dataset $\mathbb{D}$ is

$$\mathbb{D} = \{\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{Z}\} ,$$

with dimension of $n \times (m+1)$. For a single observation of data,

$$\mathbb{D}_i = \{\, y_i, x_i, z_i \,\}.$$

In our work, we use simple random sampling to set aside n_s observations from the full data set $\mathbb{D}$ into a sub-sample, denoted as $\mathbb{D}_s$. The role of $\mathbb{D}_s$ is explained in a later section. After setting aside n_s observations, we also have $\mathbb{D}_r$, which is all the other data from $\mathbb{D}$ that are not in the sub-sample $\mathbb{D}_s$. Note that $\mathbb{D}_r$ has $n_r = n - n_s$ observations. We assume that n is very large, while n_s is relatively small such that $\mathbb{D}_s$ can be easily stored and processed on a desktop computer.For convenience, we can follow the convention used in machine learning, where 20% of the data is used for training, and set aside 20% of the full data as a sub-sample.

We now illustrate the notations relevant to coarsening and aggregation functions using a very small sample of NYC Yellow Taxi data from The City of New York (*New York City's Taxi and Limousine Commission (TLC) Trip Record Data*), which is shown in Table 3.1.

ReceivedTip	TripDistance	PaymentType
0	4.13	CRD
0	3.16	CRD
0	3.30	CRD
0	2.16	CRD
0	1.00	CRD
1	0.69	CSH
1	0.80	CSH
0	1.11	CSH
0	1.39	CSH
1	2.33	CRD
1	4.79	CRD
1	2.70	CRD
1	2.50	CRD

Table 3.1: *An example of NYC yellow taxi data*

For our logistic regression, we only use three variables from this data set. We use `TripDistance` and `PaymentType` as the covariates, and derive the outcome `ReceivedTip` from the amount of tip a driver received. Since we coarsen `TripDistance`, using our notation, it maps to $\boldsymbol{x}$. Likewise, `PaymentType` corresponds to our non-coarsen covariate z_1. We generate the binary outcome `ReceivedTip` based on whether a driver received a tip. Since this is the outcome variable in our logistic regression model, it maps to the vector of responses $\boldsymbol{y}$. We denote the coarsening function as c such that

$$c(x_i) = x_i^*$$

where x_i^* is the coarsened version of x_i. For example, if we coarsen by using the ceiling function

$$c(x_i) = \lceil x_i \rceil \ ,$$

and apply this to the data in Table 3.1, we obtain the coarsened values in `CoarsenedTripDistance` column as shown in Table 3.2. Note that in general, x_i^* is restricted to the range of values within the coarsened domain. For this example, $x_i^* \in \{1, 2, \cdots, \kappa\}$ where κ is the largest value of $\lceil \boldsymbol{x} \rceil$, which is $\kappa = 15$. The coarsening notation can also apply to the dataset level, i.e.,

ReceivedTip	TripDistance	PaymentType	CoarsenedTripDistance
0	4.13	CRD	5
0	3.16	CRD	4
0	3.30	CRD	4
0	2.16	CRD	3
0	1.00	CRD	1
1	0.69	CSH	1
1	0.80	CSH	1
0	1.11	CSH	2
0	1.39	CSH	2
1	2.33	CRD	3
1	4.79	CRD	5
1	2.70	CRD	3
1	2.50	CRD	3

Table 3.2: *Data after coarsening*

$$\begin{aligned} c(\mathbb{D}) &= c\Big(\big\{\boldsymbol{y}, c(\boldsymbol{x}), \boldsymbol{Z}\big\}\Big) \\ &= \big\{\boldsymbol{y}, \boldsymbol{x}^*, \boldsymbol{Z}\big\} \\ &= \mathbb{D}^* \ . \end{aligned}$$

For the aggregation, we denote a as the aggregation function. Assuming that all the covariates and the outcome variable are all categorical, the aggregation function counts the number of observations in each unique combination of the outcome and covariates. This form of counting is possible because the coarsening function c, has restricted the possible values of x_i. If we are performing logistic regression where the outcome variable is binary, when we coarsen then aggregate the dataset not in the sub-sample $\mathbb{D}_r^*$, we obtain a compressed form of it with some loss of information:

$$a(\mathbb{D}_r^*) = \Big\{n_j, y_j, x_j^*, \boldsymbol{Z}_j\Big\}_{j=1}^{J} \ ,$$

where J is the total number of unique combinations of, $\boldsymbol{x}^*$, response $\boldsymbol{y}$, and $\boldsymbol{Z}$. In addition,

n_j is the number of observations in the jth unique combination. When we apply our aggregation to the data in Table 3.2, the result would be in the form of Table 3.3, where the `CoarsenedTripDistance` column contains the coarsened covariate $\boldsymbol{x}^*$, `NumObs` column contains the number of observations within the jth unique combination such that $n = \sum_{j=1}^{J} n_j$, and `NumSuccesses` column contains the number of 'successes'.

j	NumObs	NumSuccesses	NumFails	CoarsenedTripDistance	PaymentType
1	1	0	1	1	CRD
2	2	2	0	1	CSH
3	0	0	0	2	CRD
4	2	0	2	2	CSH
5	4	3	1	3	CRD
6	0	0	0	3	CSH
7	2	0	2	4	CRD
8	0	0	0	4	CSH
9	2	1	1	5	CRD
10	0	0	0	5	CSH

Table 3.3: *Coarsened and Aggregated Data*

Table 3.3 demonstrates the level of data reduction we can achieve through coarsening and aggregation. Instead of using n observations of $\mathbb{D}$, we scale J observations to n. In this example, the NYC Taxi data could have observations in the order of millions whereas after we apply coarsening and aggregation, the dataset we only use $J = 10$ observations. Although we coarsen one covariate in our logistic regression, in principle, we can coarsen multiple covariates.

3.3.1 Complete Data Likelihood

We now proceed to explain the regression model in detail. We begin by modelling the complete data case where the data are not coarsened or aggregated. Lipsitz et al. (2004) re-expressed the joint density of the outcome and the coarsened covariate by conditioning on covariates that we always observe (i.e., not coarsened). Using Bayes Theorem and Conditional Total Probability, the joint density of the complete data can be expressed as a product of two density functions

$$f(y_i, x_i \mid \mathbf{z}_i\,;\,\boldsymbol{\beta}, \boldsymbol{\alpha}) = f(y_i|x_i, \mathbf{z}_i\,;\,\boldsymbol{\beta})\, f(x_i|\mathbf{z}_i\,;\,\boldsymbol{\alpha})\,, \tag{3.1}$$

where y is the outcome variable, x is the coarsen covariate, $\mathbf{z}$ is the covariates that are not coarsened, and $i \in \{1, 2, \ldots, n\}$ is the index for the observations. The first density function on the right-hand side $f(y_i|x_i, \mathbf{z}_i\,;\,\boldsymbol{\beta})$, parameterised by $\boldsymbol{\beta}$ has the outcome as conditional on all

the covariates, including ones that we are going to coarsen (x_i). The second part on the right-hand side $f(x_i|\mathbf{z}_i\,;\,\boldsymbol{\alpha})$, is the density function of the covariate to be coarsened, conditional on the observed, parameterised by $\boldsymbol{\alpha}$. In the complete data case, where there are no coarsening, the second density function adds no additional information. However, when data are coarsened, this density is used as part of the EM algorithm to recovery the information lost due to coarsening.

In a logistic regression, if we assume $\boldsymbol{y}$ is binary, to estimate the parameters $\boldsymbol{\beta}$ of density function $f(y_i|x_i,\mathbf{z}_i)$ from (3.1), we model y_i as

$$y_i \sim \text{Bern}(\pi_i)\ ,$$

where π_i is the probability of a *success* outcome ($y_i = 1$). We then model the relationship between $\boldsymbol{y}$ to $\boldsymbol{x}$ and $\boldsymbol{z}$ Generalized Linear Models (GLM). The linear predictors for the for ith observation, $\eta_i^{(y)}$ can be expressed as

$$\eta_i^{(y)} = \beta_0 + \beta_x x_i + \beta_{z1} z_{1i} + \ldots + \beta_{zp} z_{pi}\ .$$

The mean of the ith observation, $\mu_i^{(y)}$ is modelled as

$$\mu_i^{(y)} = \text{E}(y_i \mid x_i, \mathbf{z}_i)\ .$$

The mean of the observations relates to the linear terms through the link function, g such that

$$g(\mu_i^{(y)}) = \eta_i^{(y)}\ .$$

If $g(\mu_i^{(y)})$ is the *canonical* link function, we have

$$\begin{aligned} g(\mu_i^{(y)}) &= \eta_i^{(y)} \\ \text{logit}(\mu_i^{(y)}) &= \eta_i^{(y)} \\ \mu_i^{(y)} &= \frac{\exp(\eta_i^{(y)})}{1+\exp(\eta_i^{(y)})}\ . \end{aligned}$$

The vector of GLM score equations, for the ith complete data observation is

$$\begin{aligned} \boldsymbol{s}_{1i} &= \begin{bmatrix} 1 \\ x_i \\ \mathbf{z}_i \end{bmatrix} \left\{ y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i) \right\} \\ &= \begin{bmatrix} y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i) \\ [y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i)]x_i \\ [y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i)]\mathbf{z}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_{1i(1)} \\ \boldsymbol{s}_{1i(2)} \\ \boldsymbol{s}_{1i(3)} \end{bmatrix} . \end{aligned} \tag{3.2}$$

We solve (3.2) to obtain $\hat{\boldsymbol{\beta}}$, the estimates of parameters $\boldsymbol{\beta}$.

We can model the coarsened covariate x_i with different density functions. In this work, we begin by assuming that x_i comes from the Exponential distribution. This choice is motivated by the NYC Taxi trip data we are going to use in simulation section. As an alternative, we model x_i using LogNormal distribution in our application section. From the second part of (3.1), to estimate the vector of parameters $\boldsymbol{\alpha}$, we begin by modelling x_i as

$$x_i \sim \text{Exp}(\lambda_i) ,$$

where λ is the *rate* parameterisation of Exponential distribution. Following the GLM notation, the combination of linear predictors is

$$\eta_i^{(x)} = \alpha_0 + \alpha_1 z_{1i} + \alpha_2 z_{2i} + \ldots + \alpha_p z_{pi} .$$

In this work, note that the parameters of interest are those in the linear predictor, i.e., $\boldsymbol{\alpha}$, and λ is a nuisance parameter. To relate the mean of the outcome with the linear predictors, instead of using the canonical link (inverse function), we use the log link function since this relaxes the positivity requirements of the linear predictors $\eta_i^{(x)}$ (which requires $\eta_i^{(x)}$ to be positive since the mean of the Exponential distribution is positive). We relate $\mu_i^{(x)}$ to the linear predictor as follows

$$\begin{aligned} g(\mu_i^{(x)}) &= \eta_i^{(x)} \\ \ln(\mu_i^{(x)}) &= \eta_i^{(x)} \\ \mu_i^{(x)} &= \exp(\eta_i^{(x)}) . \end{aligned}$$

The GLM score equations, for the i^{th} complete data observation are

$$\begin{aligned} \boldsymbol{s}_{2i} &= \begin{bmatrix} 1 \\ \boldsymbol{z}_i \end{bmatrix} \left\{ \frac{1}{\mu_i^{(x)}(\boldsymbol{z}_i)} \left\{ x_i - \mu_i^{(x)}(\boldsymbol{z}_i) \right\} \right\} \\ &= \begin{bmatrix} \frac{x_i - \mu_i^{(x)}(\boldsymbol{z}_i)}{\mu_i^{(x)}(\boldsymbol{z}_i)} \\ \frac{\boldsymbol{z}_i (x_i - \mu_i^{(x)}(\boldsymbol{z}_i))}{\mu_i^{(x)}(\boldsymbol{z}_i)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_{2i(1)} \\ \boldsymbol{s}_{2i(2)} \end{bmatrix} \end{aligned} \tag{3.3}$$

More generally, we can also express the Exponential distribution as a Gamma distribution with *shape* ($k = 1$) and *scale* (v) parameterisation. We if model using the Gamma distribution, the GLM score equations become

$$\boldsymbol{s}_{2i} = \begin{bmatrix} 1 \\ \mathbf{z}_i \end{bmatrix} \frac{v}{\mu_i^{(x)}(\mathbf{z}_i)} \left\{ x_i - \mu_i^{(x)}(\mathbf{z}_i) \right\} .$$

When we estimate the standard errors in a later section, we combine the score equations, (3.2) and (3.3), to form a combined score equation denoted as $\mathbb{S}$.

3.3.2 Estimating from sub-sample

In our work, before we apply coarsening and aggregation, we divide the complete data $\mathbb{D}$ into a random sub-sample, denoted as $\mathbb{D}_s$, and another data set contains the remaining data, denoted as $\mathbb{D}_r$. The number of observations in these data sets are n_s and n_r respectively. Since the random sub-sample $\mathbb{D}_s$ is smaller than $\mathbb{D}$ (e.g., 20% of $\mathbb{D}$), $n_s < n_r$. We coarsen and aggregate the data in $\mathbb{D}_r$, but not the random sub-sample $\mathbb{D}_s$ because the sub-sample has different roles in our analysis. The first role is to allow us to estimate the distribution of the coarsened covariate through exploratory data analysis. For example, by plotting a frequency histogram of the covariate to coarsen, we gain an understanding of the possible density functions the covariate can be modelled. Secondly, since $\mathbb{D}_s$ has complete data, we can re-incorporate it into the EM algorithm when estimating the parameters. By doing this, we speed up the convergence of the EM algorithm.

3.3.3 Observed Likelihood

We now begin the process of modelling using $\mathbb{D}_r$ (data not in the sub-sample), which when we coarsen and aggregate becomes $\mathbb{D}_r^*$. When we coarsen a covariate, the joint density of the data is no longer in the form of (3.1). To take into account the coarsened covariate x^*, we utilise the Missing Information Principle (Orchard and Woodbury, 1972) and the EM algorithm (Demp-

ster, Laird, and Rubin, 1977). In the case of x that is continuous, (3.1) becomes the integral over all possible value of x that will result in the coarsened value x^*.

$$\begin{aligned} f(y_i, x_i^* \mid \mathbf{z}_i\,;\,\boldsymbol{\beta},\boldsymbol{\alpha}) &= \int\limits_{x:c(x)=x_i^*} f(y_i, x \mid \mathbf{z}_i\,;\,\boldsymbol{\beta},\boldsymbol{\alpha})\,\mathrm{d}x \\ &= \int\limits_{x:c(x)=x_i^*} \Big\{ f(y_i|x,\mathbf{z}_i\,;\,\boldsymbol{\beta})\, f(x|\mathbf{z}_i\,;\,\boldsymbol{\alpha}) \Big\}\,\mathrm{d}x\,. \end{aligned} \tag{3.4}$$

For instance, if c is the ceiling function and we observe $x^* = 3$, the integration is taken across $2 < x \leqslant 3$.

Recall that we divide the original data set $\mathbb{D}$ into a random sub-sample $\mathbb{D}_s$ and the remaining data $\mathbb{D}_r$, and only apply coarsening and aggregation to $\mathbb{D}_r$. As a result, our observed data is a combination of the complete data in $\mathbb{D}_s$, and the coarsened and aggregated data in $\mathbb{D}_r^*$. That is, the observed data likelihood L_{obs} is

$$L_{obs} = \underbrace{\prod_{i=1}^{n_s} \Big[f(y_i|x_i,\mathbf{z}_i) f(x_i|\mathbf{z}_i) \Big]}_{\text{likelihood from } \mathbb{D}_s} \underbrace{\prod_{i=n_s+1}^{n} \Bigg[\int\limits_{x:c(x)=x_i^*} \Big\{ f(y_i|x,\mathbf{z}_i)\, f(x|\mathbf{z}_i) \Big\}\,\mathrm{d}x \Bigg]}_{\text{likelihood from } \mathbb{D}_r^*} \quad . \tag{3.5}$$

3.3.4 Estimating the Parameters using $\mathbb{D}_r^*$

As discussed in the Introduction section, if we use the coarsened data $\mathbb{D}_r^*$ directly in our regression, there will be bias to the estimates of the regression coefficients. In addition, naïvely using the coarsened data this way can underestimate the standard errors, since the coarsening function collapses observations within a coarsen interval into a single value. For example, if the ceiling coarsening function is applied to values between 2 and 3, all the coarsened values are 3. Instead of using the coarsened data, we use the plausible values within the coarsened interval. We generate these plausible values using the Expectation Maximization (EM) algorithm (Dempster, Laird, and Rubin, 1977). An example of using the EM algorithm this way in the medical setting is Brumback, Cook, and Ryan (2000), where the observed data are interval-censored age data. Lipsitz et al. (2004) also use EM algorithm this way to demonstrate the general case where x is discrete.

In this work, the E-Step of the EM algorithm involves specifying the conditional expectation of the vector of combined score equations $\mathbb{S}$ (consisting both $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$.), and the M-Step involves solving these to estimate the parameters. The conditional expectation is taken with

respect to the *the complete data, conditioned on the observed data.* Our observed data in this work is $\mathbb{D}_r^*$, which consists of $(y_i, x_i^*, \boldsymbol{z}_i)$. We abbreviate the observed data as 'observed' in a lot of the equations for brevity. However, in some equations, we write out the observed data fully to aid clarity. Using the notation just outlined, the conditional expectation of the combined score equation for the ith observation is

$$\mathrm{E}_{x|\text{observed}}[\,\mathbb{S}\,] \tag{3.6}$$

In the M-step of the EM algorithm, we then solve the conditional expectation of these score equations, to obtain new estimates of the parameters. The new estimates are then used in the E-Step again. We stop the iterations of the EM algorithm when the estimates of the parameters converge. Expanding the expectations from (3.6) results in integrated score equations:

$$\begin{aligned}\mathrm{E}_{x|\text{observed}}[\,\mathbb{S}\,] &= \int_{x:c(x)=x_i^*} \mathbb{S}\, f(x|\text{observed})\, \mathrm{d}x \\ &= \int_{x:c(x)=x_i^*} \mathbb{S}\, f(x|y_i, x_i^*, \boldsymbol{z}_i)\, \mathrm{d}x\ .\end{aligned} \tag{3.7}$$

Importantly, observations with the same observed values of contribute equally to the conditional expectation of the score equations. As such, we can aggregate the conditional expectations over unique combinations of the observed data. Applying the aggregation function transforms the data as follows

$$\left\{\begin{pmatrix} y_i \\ x_i^* \\ \boldsymbol{z}_i \end{pmatrix}\right\}_{i=n_s+1}^{n} \longrightarrow \left\{\begin{pmatrix} y_j \\ x_j^* \\ \boldsymbol{z}_j \\ n_j \end{pmatrix}\right\}_{j=1}^{J} \tag{3.8}$$

where J is the total number of unique combination of observed values, indexed by j. As such x_j^* and $\boldsymbol{z}_j$ are the values of the covariates, and y_j is the outcome, for the jth unique combination. More formally, consider logistic regression in our work, where we have an outcome variable that is binary (y), one covariate that we coarsen (x), and one non-coarsened covariate (z). If z is categorical, after we coarse x, the outcome variable y and both the covariates x^* and z are categorical. We define the set of possible values the outcome and covariates can

take as

$$\begin{aligned}\psi &:= \{0, 1\},\\ \phi &:= \{1, 2, \ldots, \kappa\},\\ \omega &:= \{1, 2, \ldots, \psi\}\,.\end{aligned}$$

The outcome and covariates belong to the sets as follows

$$\begin{aligned}y_i &\in \psi,\\ x_i^* &\in \phi,\\ z_i &\in \omega\,.\end{aligned}$$

The result of the *cartesian product* of the above sets ψ, ϕ, and ω is the set Γ, whose elements are 3-tuple that contains the unique combinations of the levels of outcome and the covariates. That is

$$\begin{aligned}\Gamma &= \psi \times \phi \times \omega\\ &= \big\{\{0, 1, 1\},\ \{0, 1, 2\}, \ldots, \{1, \kappa, \psi\}\big\}\,.\end{aligned}$$

For example, if $\kappa = 2$ (the coarsened covariate has 2 levels) and $\psi = 2$ (the non-coarsened covariate has 2 levels), then

$$\Gamma = \big\{\{0, 1, 1\},\ \{0, 1, 2\}, \{0, 2, 1\}, \{0, 2, 2\}, \{1, 1, 1\},\ \{1, 1, 2\}, \{1, 2, 1\}, \{1, 2, 2\}\big\}\,.$$

We denote the total number of possible combinations as J, which is the *cardinality* of Γ, i.e., $J = |\Gamma|$. For the previous example, $J = 8$. Note that J can also be calculated as $2 \times \kappa \times \psi$. Each element of the set Γ is indexed by j such that

$$\begin{aligned}j &\in \{1, 2, \ldots, |\Gamma|\}\\ &= \{1, 2, \ldots, J\}\end{aligned}$$

Since the observations in each jth contribute equally to the score equation, (3.6) can be aggregated as

$$\sum_{j=1}^{J} \left\{ n_j\, \mathbb{E}_{x|y_j, x_j^*, \mathbf{z}_j} \big[\mathbb{S}_j(\cdot)\big] \right\},$$

where n_j is the number of observations in each jth combination. In this work, we refer to

covariates and score equations in the context of aggregation, we implicitly use j indexing.

3.3.5 Parameter Estimation using EM algorithm

If we can obtain a closed form solution to the conditional expectations of the score equations in (3.7), we can solve them in the M-Step. However, obtaining a closed form solution to the integral is not possible where y_i follows a logistic regression model. We can however, approximate the integral using techniques such as numerical integration or Monte Carlo EM (Ibrahim, Chen, and Lipsitz, 1999; Wei and Tanner, 1990), where we sample from the distribution of $f(x \mid \text{observed}) = f(x \mid y_i, x_i^*, \boldsymbol{z}_i)$. In both cases, the approximation to the integral in (3.7) has a *weighted* sum form (See Appendix 3.E). The weighted sum formulation has the advantage that in the M-Step, we can use existing methods such as weighted GLM (Venables and Ripley, 2002) to estimate our parameters. This is the core idea behind EM by methods of weights introduced by Ibrahim (1990), and Ibrahim, Chen, and Lipsitz (1999). In this work, we use *importance sampling*, which is a specific Monte Carlo Integration technique to approximate the integral, and use the importance samples and their normalised weights. As mentioned earlier, Kim and Hong (2012) outline how to perform statistical inference on coarsened data in the general case, where the coarsening function can be stochastic. In their work, the function that generates the plausible values of un-coarsened values x depends on the coarsening function. To reduce the computational load when performing MCMC with Metropolis-Hastings algorithm, Kim and Hong (2012) used a technique by Kim (2011), which reduces the function that generates plausible values of un-coarsened values to a weighted sum formulation. These weights are different from weights in this work (described a bit later) since they account for the coarsen mechanism.

The Importance Sample approximation to (3.7) is

$$\frac{1}{m}\sum_{k=1}^{m}\frac{f(x = u_k \mid \text{observed})}{h(u_k)}\,\mathbb{S}_j(u_k)\,, \tag{3.9}$$

where m is the number of *importance samples* (in most part of this work, $m = 20$ unless stated otherwise), h is the *importance function* (in this case is Uniform) which we sample from to obtain $u_1, u_2, \ldots, u_m$. $\mathbb{S}_j()$ is our combined score equation, with j parameterisation where j is the jth combination of observed values. The fraction part is also known as the *weight* component of the approximation. For a brief outline of Importance Sampling see Appendix 3.E.

The conditional density $f(x = u_k \mid \text{observed})$ in (3.9) can be can be un-normalised. In these situations, instead of using the weights directly, we use the *self normalising weights*. One

form of self-normalising weights proposed by Robert and Casella (2010) is

$$w_j = \left[f(u_k)/g(u_k)\right] \Big/ \sum_{k=1}^{m} \left[f(u_k)/g(u_k)\right] . \tag{3.10}$$

If we incorporate the self normalising weights, the approximation to the integral in (3.7) has the weighted sum form

$$\sum_{k=1}^{m} w_k \, \mathbb{S}_j(u_k) \, . \tag{3.11}$$

When we implement (3.11), the importance samples are sampled from a uniform distribution with its lower and upper bounds corresponding to the coarsened interval of x_j^*. In our work, we generate these importance samples *outside* the EM algorithm such that they are repeatedly used across all iterations of the EM algorithm.

Recall that we divide the original data set $\mathbb{D}$ into a random sub-sample $\mathbb{D}_s$ and remaining data $\mathbb{D}_r$, and the coarsening and aggregation is applied to $\mathbb{D}_r$. To recover the complete data model, we specify the conditional expectation of $\mathbb{D}_r^*$. Before we solve the score equations in the M-Step, we re-incorporate the random sub-sample $\mathbb{D}_s$. Since $\mathbb{D}_s$ is not coarsened, we re-incorporate it by assigning weight value of one to the data in $\mathbb{D}_s$. As an illustration, table 3.5 contains a small example of data from the sub-sample. When it is incorporated, as shown in Table 3.4, each item in the sub-sample has weight value of one. In addition, they are individual items in the new table since importance sampling is not applied.

j	ReceivedTip	TripDistance	PaymentType	NumSuccesses	NumFails	ImportanceWeight
4	0	0.11	0	0	1	1
5	1	3.34	0	1	0	1
6	0	0.29	0	0	1	1
7	1	0.60	0	1	0	1

Table 3.4: *Sub-sample data transformed to coarsened-data format*

ReceivedTip	TripDistance	PaymentType
0	0.11	0
1	3.34	0
0	0.29	0
1	0.60	0

Table 3.5: *An example of a sub-sample*

In the M-Step, we solve the score equations using combined data consisting of $\mathbb{D}_s$ and importance samples for $\mathbb{D}_r^*$. Within each iteration of the EM algorithm, we only update the

weights (and their normalised counterparts) w_k. These updated weights and the importance samples are then used in GLM to obtain the next set of estimates of the parameters. We stop the EM iterations when the estimates of the parameters converge.

3.4 Estimating Standard Errors

In addition to obtaining the estimates of the parameters from the EM algorithm, we performed additional work to obtain the standard errors of the estimates. We can apply the method by Louis (1982), which involves adjusting the complete data Information Matrix to take into account the uncertainty from using the observed (i.e,. not complete) data. However, the method by Louis (1982) can become tedious because we need to specify all the elements of the information matrix, as well as their conditional expectations. As an alternative, we can use the method by Meilijson (1989) which is applicable when the observed data i.i.d. In the method by Meilijson (1989), we sum the conditional expectations of the *outer product* of the score equations for *each observation* to obtain the *empirical Fisher information matrix*. We then obtain the standard errors by taking the inverse of the empirical Fisher information matrix. In our case, the empirical Fisher information matrix is calculated as

$$\boldsymbol{I} = \sum_{i=1}^{n_{obs}} \left\{ \mathrm{E}_{x|\text{observed}}[\mathbb{S}_i] \times \mathrm{E}_{x|\text{observed}}[\mathbb{S}_i]^\top \right\} \quad , \tag{3.12}$$

where n_{obs} is the number of *observed* data (e.g., can be data within the jth combination, or it can be $n_r + n_s = n$), and recall $\mathbb{S}_i$ is the combined score equation for the ith observation. Expanding the conditional expectation of the combined score equation by incorporating the score equations from (3.2) and (3.3), we have

$$\begin{aligned} \mathrm{E}_{x|\text{observed}}[\mathbb{S}_i(\cdot)] &= \mathrm{E}_{x|\text{observed}} \begin{bmatrix} y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i) \\ \{y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i)\} x_i \\ \{y_i - \mu_i^{(y)}(x_i, \mathbf{z}_i)\} \mathbf{z}_i \\ \{x_i - \mu_i^{(x)}(\boldsymbol{z}_i)\} \,/\, \{\mu_i^{(x)}(\boldsymbol{z}_i)\} \\ \{\boldsymbol{z}_i (x_i - \mu_i^{(x)}(\boldsymbol{z}_i))\} \,/\, \{\mu_i^{(x)}(\boldsymbol{z}_i)\} \end{bmatrix} \\ &= \begin{bmatrix} y_i - \mathrm{E}_{x|\text{observed}}(\mu_i^{(y)}(x_i, \boldsymbol{z}_i)) \\ \mathrm{E}_{x|\text{observed}}\big(\{y_i - \mu_i^{(y)}(x_i, \boldsymbol{z}_i)\}\, x_i\big) \\ \big\{ y_i - \mathrm{E}_{x|\text{observed}}\big(\mu_i^{(y)}(x_i, \boldsymbol{z}_i)\big) \big\}\, \boldsymbol{z}_i \\ \frac{1}{\mu_i^{(x)}(\boldsymbol{z}_i)} \left(\mathrm{E}_{x|\text{observed}}(x_i) - \mu_i^{(x)}(\boldsymbol{z}_i) \right) \\ \frac{\boldsymbol{z}_i}{\mu_i^{(x)}(\boldsymbol{z}_i)} \left(\mathrm{E}_{x|\text{observed}}(x_i) - \mu_i^{(x)}(\boldsymbol{z}_i) \right) \end{bmatrix} . \end{aligned} \tag{3.13}$$

Since the coarsened covariate ${x_i}^*$ is continuous, the conditional expectations in (3.13) expand to integrals. For brevity, we show the integral form of the first row of (3.13). Also, we will assume there is only one uncoarsened covariate z_{1i}. The integral form is

$$\begin{aligned} \mathrm{E}_{x|\text{observed}}\big(y_i - \mu_i^{(y)}(x_i, z_{1i})\big) &= y_i - \mathrm{E}_{x|\text{observed}}\big(\mu_i^{(y)}(x_i, z_{1i})\big) \\ &= y_i - \int_{\mathcal{X}} \mu_i^{(y)}(x_i, z_{1i})\, f(x_i \mid \text{observed})\, \mathrm{d}x \\ &= y_i - \int_{\mathcal{X}} \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 z_{1i})}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 z_{1i})} f(x_i|\text{observed})\, \mathrm{d}x \ . \end{aligned} \tag{3.14}$$

We then approximate the conditional expansion such as (3.14) using importance sampling as shown in Section 3.3.5. Using $(u_1, u_2, \ldots, u_m)$ importance samples, generated from h importance function, the approximation to the first conditional expectation in (3.13) is

$$\mathrm{E}_{x|\text{observed}}\big(y_i - \mu_i^{(y)}(x_i, z_{1i})\big) \approx y_i - \left[\frac{1}{m}\sum_{k=1}^{m}\left\{\mu_i^{(y)}(u_k, z_{1i})\frac{f(x = u_k \mid \text{observed})}{h(u_k)}\right\}\right] \tag{3.15}$$

If we denote w as the normalised weights, as shown in Section 3.3.5, the approximation shown in (3.15) becomes

$$\begin{aligned} \mathrm{E}_{x|\text{observed}}\big(y_i - \mu_i^{(y)}(x_i, z_{1i})\big) &= y_i - \mathrm{E}_{x|\text{observed}}\big(\mu_i^{(y)}(x_i, z_{1i})\big) \\ &\approx y_i - \left[\sum_{k=1}^{m}\left\{\mu_i^{(y)}(u_k, z_{1i})\, w_k\right\}\right] . \end{aligned}$$

To obtain the empirical Fisher information matrix using technique by Meilijson (1989), we use the approximations in the form shown above to all the conditional expectation of score equations in (3.13). Since we have aggregate the data, we no longer have n observations from $\mathbb{D}$, instead we have J *blocks* of *unique combination of the observed values.* We incorporate aggregation by first calculating the vector outer products (3.12) using observed data from the jth block. Since the data in each jth block block contribute equally to the score equations, we can scale the outer product by n_j, the number of observations in the jth combination. The empirical Fisher information matrix using aggregated data is then

$$\boldsymbol{I} = \sum_{j=1}^{J}\left[n_j\left\{\mathrm{E}_{x|\text{observed}}[\mathbb{S}j] \times \mathrm{E}_{x|\text{observed}}[\mathbb{S}j]^{\top}\right\}\right] \quad . \tag{3.16}$$

Since we calculate the standard error when the EM algorithm has converged, in addition to using the data from $\mathbb{D}_r^*$, we also use the data from the random sub-sample $\mathbb{D}_s$. In our work, for

coarsened and aggregated data, we follow the scaling of the outer product as shown in (3.16). For data from the random sub-sample, we evaluate the conditional expectation of the score equations without scaling. The empirical Fisher information matrix we use is the combination of the two. Once we have the combined empirical Fisher information matrix, we invert it to obtain the variance-covariance matrix. Taking the square root of it provides the standard error for the estimates. Appendix 3.C contains details of obtaining the conditional expectations of the combined score equations. In addition, Appendix 3.D has a simple example that illustrates the aggregation of vector outer product that is used in (3.16).

3.5 Simulations

To examine the effectiveness of our method, we perform 100 simulations. In each simulation, we first generate the complete data $\mathbb{D}$. We then randomly sample 20% of $\mathbb{D}$ to create the sub-sample $\mathbb{D}_s$; with the other 80% of data becoming $\mathbb{D}_r$. The number of observations for the complete data n in each simulation is 20,000. This value is chosen for convenience as it is large enough for the estimates of the regression coefficients using the complete data to be stable across simulations, and also not too large that to hinder the computational speed of the simulation. The values for covariate to coarsen x_i are sampled from the density $f(x_i|z_i;\boldsymbol{\alpha})$, with the non-coarsen covariate z_i generated from a binomial distribution. After generating x_i, we generate the outcome variable $\boldsymbol{y}$ using density $f(y_i|x_i,z_i;\boldsymbol{\beta})$. Note that β_1 is the coefficient for x and β_2 is the coefficient for z. To approximate the condition expectation of the score equations in the EM algorithm, we used $m = 20$ importance samples per jth observed value combination.

3.5.1 Varying the degree of coarsening

The panel of boxplots in Figure 3.1 shows the estimated coefficient of the coarsened covariate. Each boxplot in the panel represents the estimates of the coefficient for the coarsened covariate from 100 simulations. The main purpose is to illustrate how the estimates of the coefficients are affected by varying the degree of coarsening. The left most panel illustrates coarsening x using across 20 frequency intervals (ventile), while at the right hand side, the coarsening is done across 3 intervals (tertile). Within each panel, there are 4 boxplots, with each boxplot showing the estimates of the regression coefficients from different methods across 100 simulations. One of these methods include naïvely using the mid-points of the coarsened intervals.

The first feature to note in Figure 3.1 is how using the mid-points of the coarsened intervals consistently have more biased compared to other methods. This is consistent with the

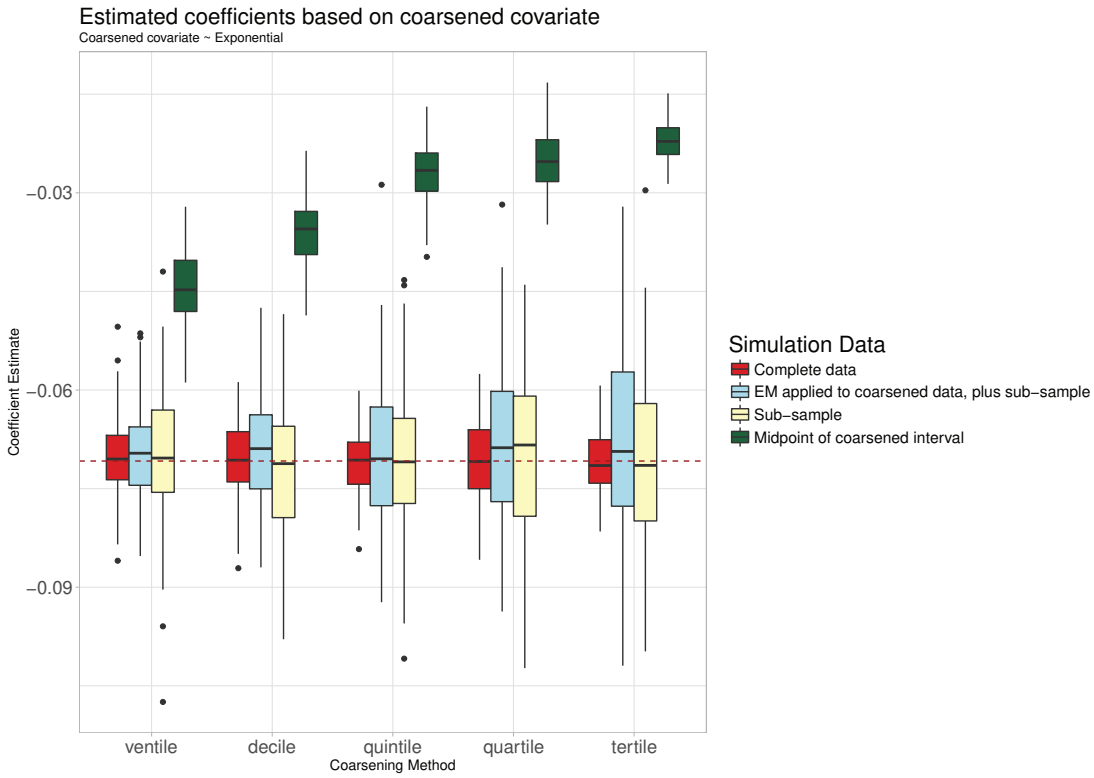


Figure 3.1: *Estimates of coefficients when varying coarsening methods*

reasoning behind using the EM algorithm, as outlined in the Introduction section. As well, the bias increases as as the coarsen interval becomes larger. The second feature to note is that since the sub-sample contains random sub-sample of the complete data, we expect its estimate to have less bias. The third feature is when we know the distribution of the coarsened covariate, the estimates of regression coefficients from our method have less bias, compared with naïvely using the coarsened interval. In addition, as the coarsening become less fine, the variation of the co-efficient estimates increase as well.

3.5.2 Varying the Number of Monte Carlo Importance Samples

Another element we vary is the number of importance samples per unique combination of observed values. Figure 3.2 shows the estimate of the parameter of the coarsened covariate as we vary the coarsening method and the number of importance samples. As expected, as we increase the number of importance samples, we obtain better approximation to the density $f(\boldsymbol{x}|\text{observed})$, which then translates into less biased estimation of the co-efficient.

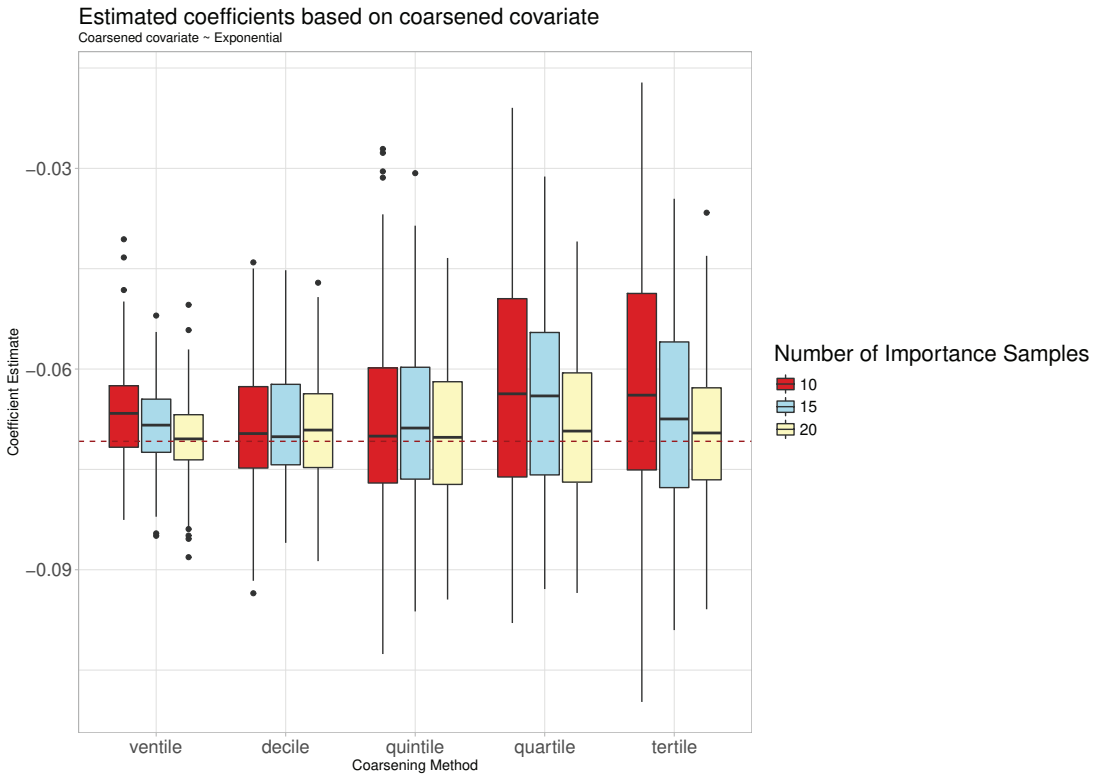


Figure 3.2: *Estimates of coefficients with varying coarsening methods and importance samples*

3.5.3 Variation in the estimated Standard Errors

Figure 3.3 shows the variation of the standard errors of $\hat{\beta}_x$. In the cases where complete data are used, these standard errors are obtained directly from the `glm` function. In the case of observed data with EM algorithm, the standard errors are calculated using methods described in Section 3.4 based on Meilijson (1989). In this figure, we can see the standard errors from the random sub-sample are higher compared to standard errors obtained from using our method. When using the observed data, as the coarsening become more coarsen, the median and the variation of the standard errors increases; as the coarsen interval becomes wider, more and more information are lost.

3.6 Application to NYC Taxi Data

We applied our method to one month of NYC Taxi data for December 2010, which we have described in earlier sections. Our analysis was performed using `R` version 3.3.2 on a computer that with Intel Xeon E5-2687W v2 3.4 GHz CPU, and 32GB of ECC DDR3 memory. We used the suite of software packages from `tidyverse` (version 1.2.1) for a variety of purposes, such as

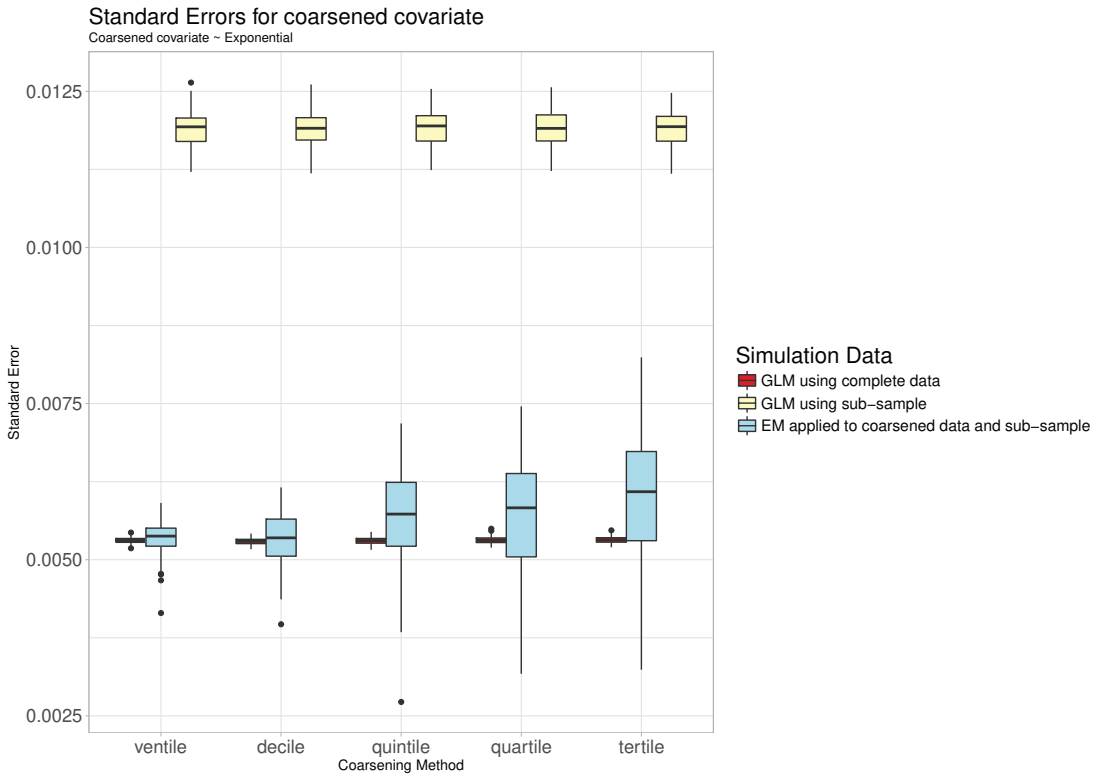


Figure 3.3: *Standard Errors with varying coarsening*

`readr` (version 1.1.1) to load CSV files, `tibble` to store our data, `dplyr` for data manipulation, and `ggplot` for plots. In addition, we used `pryr` (version 0.1.4) to measure the amount of memory various data objects in `R` occupied.

When we restricted the NYC Taxi data to only include observations that have valid trip distances (those that have values larger than zero), and also restricted the type of payment to either cash or credit card, there were 13,748,536 observations. The CSV file containing the data occupied 2.4 GB of disk space on the file system. When we loaded the CSV file into `R`, the resulting `tibble` data structure occupied 1.88 GB in memory.

The NYC Taxi data has 18 variables in its original form, which we only use three of them for our logistic regression. These are trip distance, payment type and tip amount. For the logistic regression, we constructed the binary outcome variable, received tip, from the tip amount variable. The data we use in the regression, consisting of the outcome variable we constructed, and the two covariates, occupied 275 MB in memory. Even though the memory usage of the regression data may seem small, keep in mind that the data contained trips for a single month. If we wanted to perform the analysis using data for an entire year, we would

require approximately 2.4 GB $\times$ 12 $=$ 28.8 GB of disk space to store the data in their original form as CSV files. Similarly, we would need 275MB $\times$ 12 $=$ 3.3 GB of memory to store the filtered dataset which was tailored for our logistic regression. Since R also keeps data related to GLM, such as residuals, we expect we would need a lot more computer memory. In addition to disk space and memory usage, we also need to consider the duration of the analysis. On our computer, loading the 2010 December CSV file from disk into R took approximately four minutes (our computer had 2 $\times$ 2TB 7,200 RPM SATA III Hard Drives in RAID configuration). If we assume the data load time increases linearly as more data from additional months are added, the time required to read one year of data would be 4minutes $\times$ 12 $=$ 48 minutes. As for duration the GLM operation, it took approximately approximately 2 minutes. If we again assume a linear increase in time, this means it would take 24 minutes to perform one logistic regression model using data for a single year (assuming the computer has enough memory to store both the data and the GLM result). Note that in our case, the logistic regression using GLM was performed on a computer that is much powerful than a typical desktop computer; Intel Xeon CPUs are usually found in workstation class or server class computers, and we also have a large memory at 32GB. As a result, in reality, it may not be even possible to perform the regression using the complete data.

By coarsening and aggregating, we allow regression to be performed on a large dataset using a desktop computer in a timely manner, with the trade-off being loss of information due to coarsening. Since the size of the coarsened and aggregated data depend on the the unique levels of the observed data and outcome variable (the value J discussed in earlier sections), it is possible to perform regression on a desktop computer. For example, if we coarsened the data into quintiles (5 levels), the coarsening and aggregation functions would reduce the data size to $2 \times 2 \times 5 = 20$ levels (2 levels for the outcome, 2 levels for the non-coarsened covariate, and 5 levels for the coarsened covariate) If we use $m = 20$ importance samples per level, we only need $20 \times 20 = 400$ observations to use in EM by weights.

Although it may not be feasible to use the large complete data for regression, it is possible in most situations to work with a smaller random sub-sample. For the NYC Taxi data, our random sub-sample consisted of 20% of the full dataset, which we also perform logistic regression on. Logistic regression using EM by method of weights was then performed on the coarsened and aggregated data $\mathbb{D}_r^*$. As mentioned before, the covariate that we coarsened is trip distance and our coarsening method involved a number quantile based methods (i.e., diving the data into intervals by frequency). We also perform logistic regression directly using the coarsened data, which we expect the estimates of the coefficients to have a large bias. This is shown in Figure 3.4 which illustrates the result of using the mid-points of coarsened interval, when the covariate was coarsened using quintile method. When the coarsened data is

used directly this way, the data are now grouped together in such a way that when we perform regression modelling, all the observations belonging to a coarsened interval are now take on the same value. That is, all the trip distances in the last interval are treated as if there are many trips with the same trip distance. We also expect that due to this grouping, the standard errors will be much smaller as well.

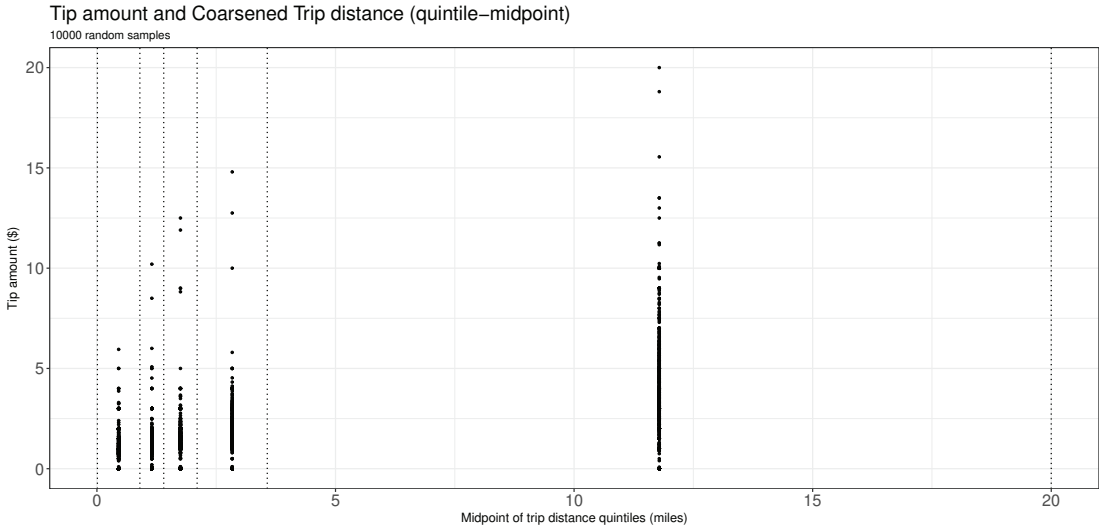


Figure 3.4: *Mid-points of coarsened trip distance covariate from NYC Taxi data for 2010 December. Since the last coarsened interval covers a large range of trip distance, naïvely using the mid-points as data in regression setting will introduce bias to the coefficient estimates.*

For the EM algorithm, we approximate the conditional expectation using $m = 100$ importance samples drawn from a uniform distribution, and their *normalised* 'weights'. In addition, as discussed earlier, in the M-Step, we re-incorporate back our random sub-sample (with each observation from the sub-sample having weight of one).

The taxi trip data exhibits characteristics that make the application of our method challenging. The trip distance covariate, which we coarsen, exhibits a long tail (some trips were recorded to be over 100 miles, approximately 160.9 km) with most of the trips were short distances. The frequency distribution of the trip distances for short trips is not monotonic either; they are groups of trip distances that are more frequent as shown in Figure 3.5. The long tail nature of the data also also posed some challenges when we specify a distribution for the trip distance covariate for the Expectation step of the EM algorithm. As we did in our simulation, we initially modelled the distribution of trip distances with an exponential distribution. Figure 3.6 shows the density histogram of the taxi trip data overlaid with density of an exponential distribution. Although the exponential distribution has a high density for very short trips, it does not capture the distribution of the trip distances in the middle (1 to 3 miles range)

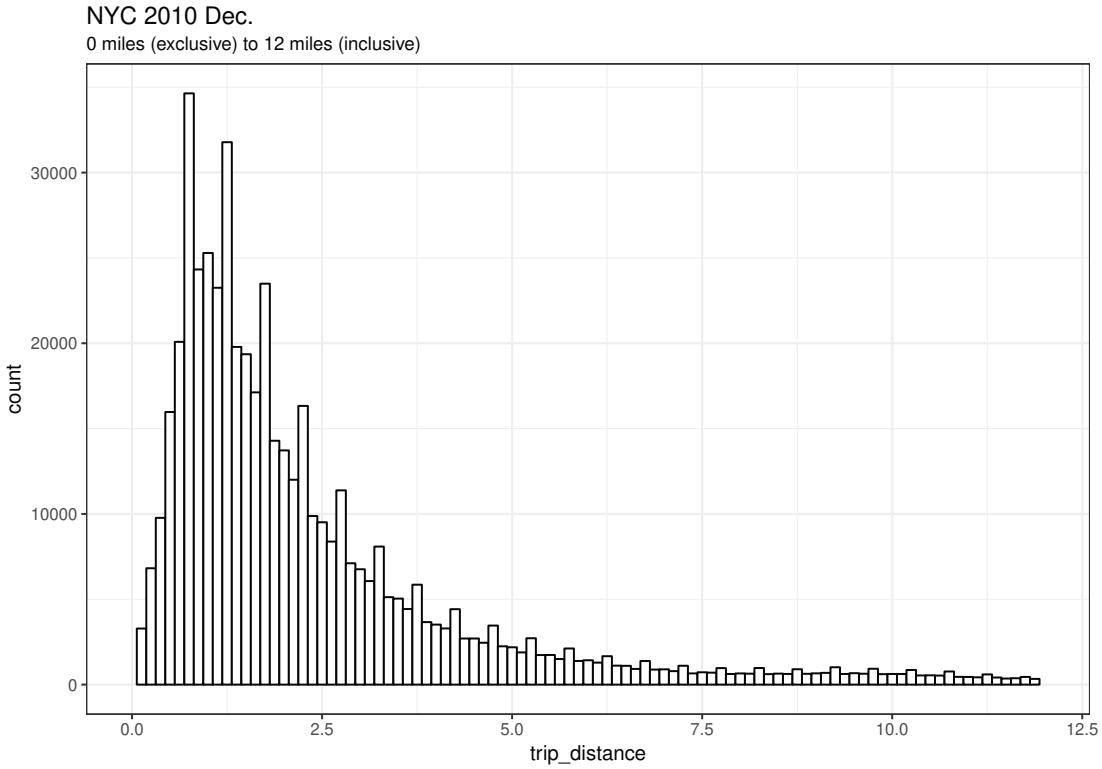


Figure 3.5: *Histogram of NYC Taxi data 2010 Dec. limited at 12 miles.*

adequately. To make the exponential distribution more suitable, we shifted the trip distance data so that the shorter distances have a higher frequency, as shown in Figure 3.7. However, even with this transformation, the frequency histogram from the exponential distribution at short distances (up to 1.5 miles) does not match the frequency of the data. In addition, at the higher trip distances, we have the opposite characteristic where there are more data from exponential distribution than the NYC Taxi trip data. The estimates of the coefficient for the trip distance covariate are shown from Figure 3.8 to Figure 3.12. Each figure illustrates the estimate of the coefficient trip distance, which we coarsened, and its standard errors. Each bar illustrates the result of logistic regression using different datasets. The first and second datasets are self-explanatory. The third bar in each figure is the result of performing regression using mid-point of each coarsened interval, and the last bar is the result from using EM by method of weights. For the coefficient estimate from using EM by method of weights, the standard error are calculated using method from Meilijson (1989). Since we have used weighted importance samples to perform the logistic regression, we need to adjust for the increase in uncertainty in the estimate. As the coarsen interval widened, from coarsening by ventile (20

groups) to coarsening by tertile (3 groups), the bias from using the mid-point, as well as from EM by method of weights increased. Even though the estimate from EM by method of weights has much less bias compared to using the mid-point.

We then used LogNormal distribution to model trip distance covariate. We do this because the distribution of the natural log of the trip distance covariate resembles the Gaussian distribution. Figure 3.13 shows the frequency distribution of trip distances overlaid with the frequency distribution of a LogNormal distribution. As with the case with exponential distribution, the LogNormal distribution cannot perfectly capture the distribution of the trip distance covariate. In particular, it has a higher density for trips less than 3 miles, but has lower density for trips after 4 miles. The natural log of trip distance can be modelled with Gaussian distribution by taking the natural log of the LogNormal distribution, but not perfectly, as illustrated in Figure 3.14. Results from logistic regression using this model are shown from Figure 3.15 to Figure 3.19. When we used this model, we did not shift the trip distance data, but only limit it to be less than 10 miles. Similar to the estimates using the exponential distribution, the coefficient estimates we obtained from EM by method of weights did include some bias. However, when coarsen interval is larger, such as quintile and quartile, the bias was smaller than compared to using exponential distribution. Even though the estimates from using EM by method of weights still have some bias, the biases are less than the estimates from naively using the mid-points of the coarsened intervals.

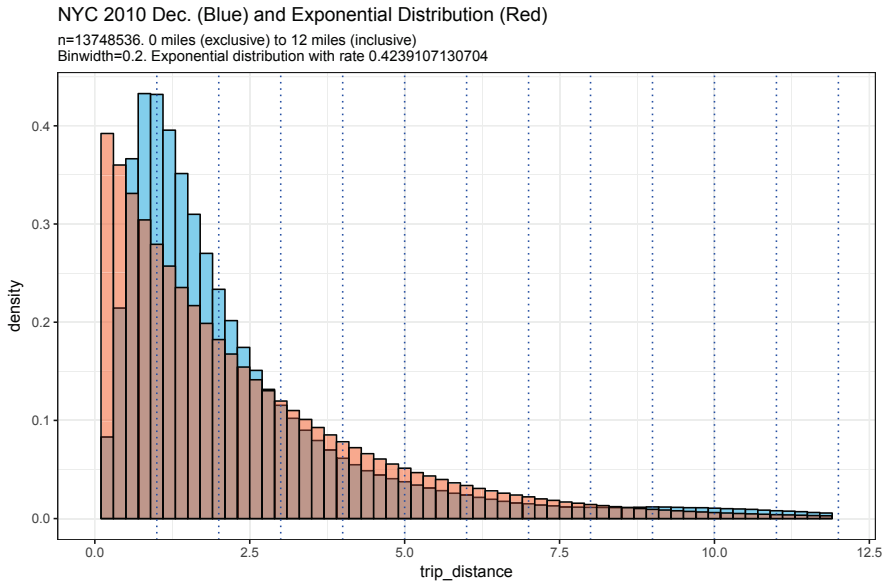


Figure 3.6: *Modelling trip distance covariate with an exponential distribution. The blue histogram is the density of the trip distance covariate, while the red histogram is the density of random samples from an exponential distribution with parameter estimated from the complete data. The areas where the densities overlap are shown in purple like colour. Data generated from exponential distribution fail to capture the density of trip between 1 to 2.5 miles of the complete data. In the tail area, exponential distribution generated more data than what is in NYC taxi data.*

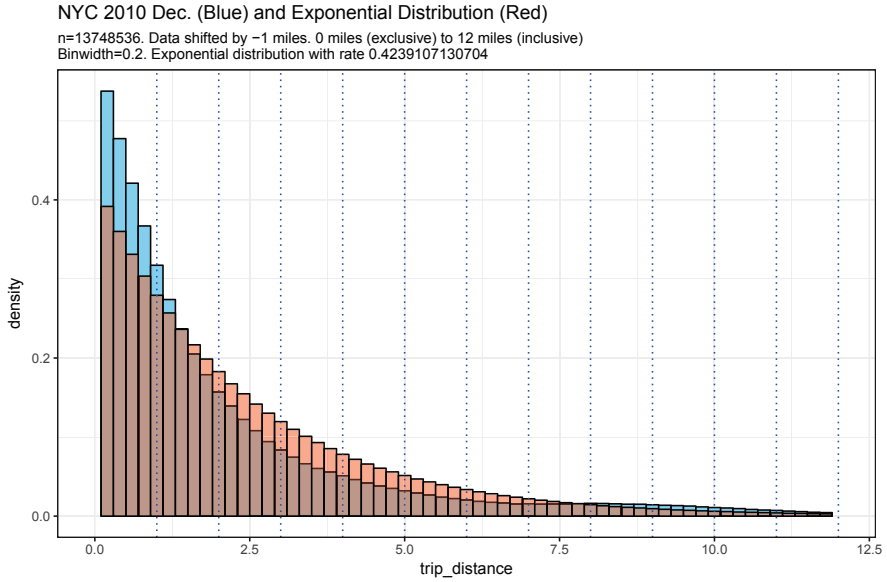


Figure 3.7: *Modelling shifted trip distance variable with an exponential distribution. The densities are as described in Figure 3.6. An attempt to match the trip distance density with an exponential distribution is to shift the trip distance variable. Even though there is a better match at the beginning, the exponential distribution now generates too many data for the tail area.*

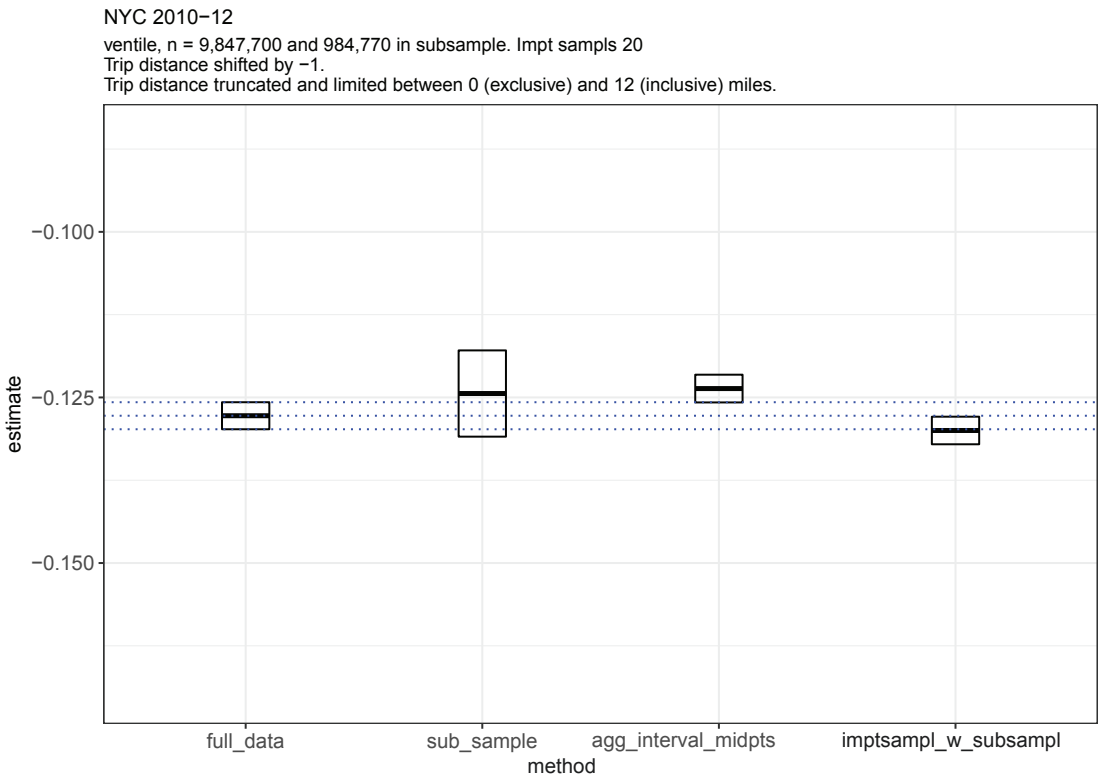


Figure 3.8: *Coefficient estimate. Ventile. Exponential.*

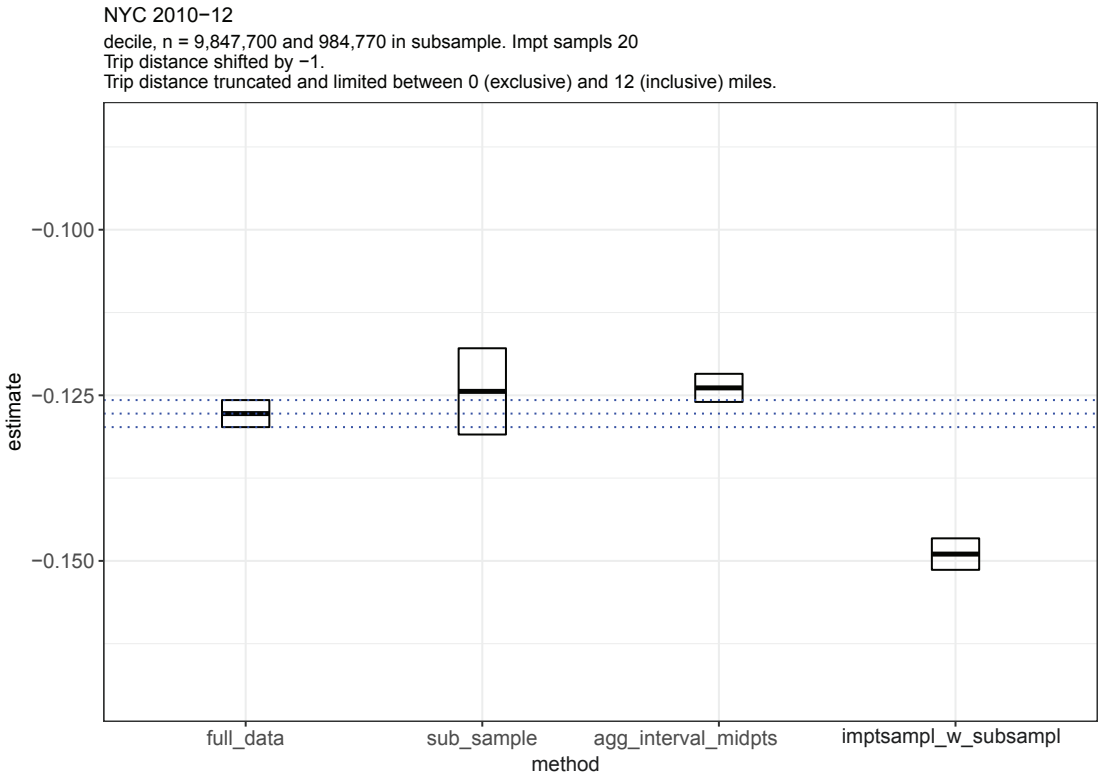


Figure 3.9: *Coefficient estimate. Decile. Exponential.*

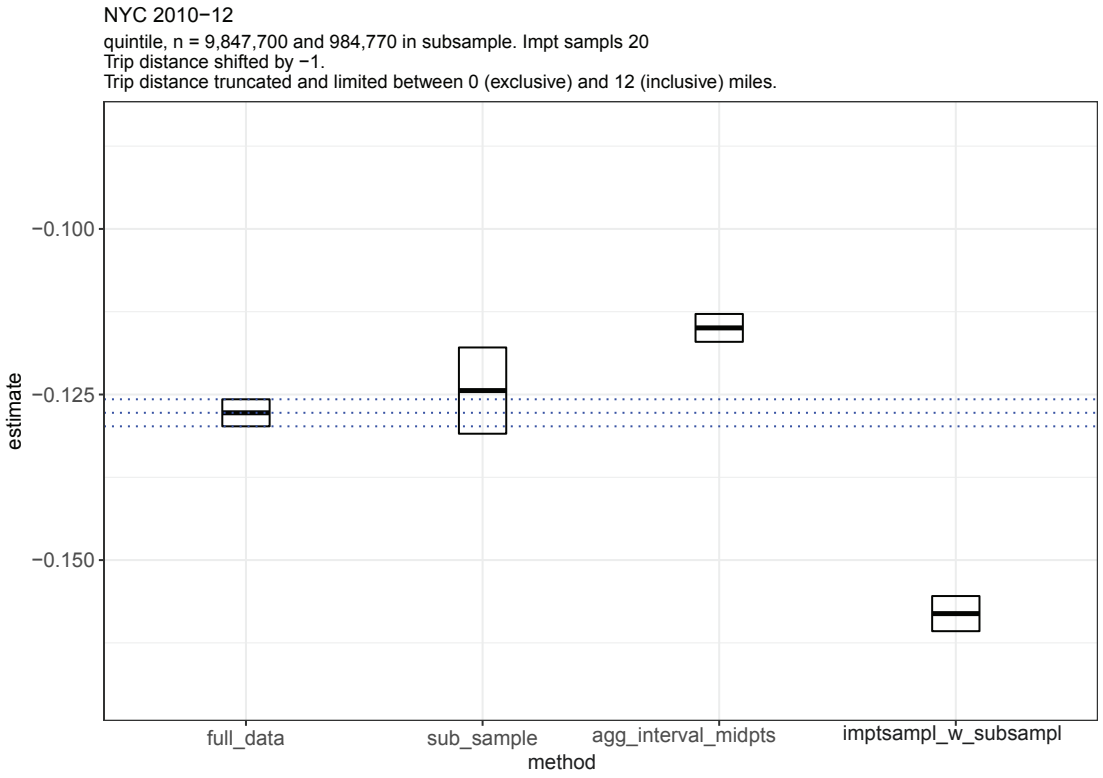


Figure 3.10: *Coefficient estimate. Quintile. Exponential.*

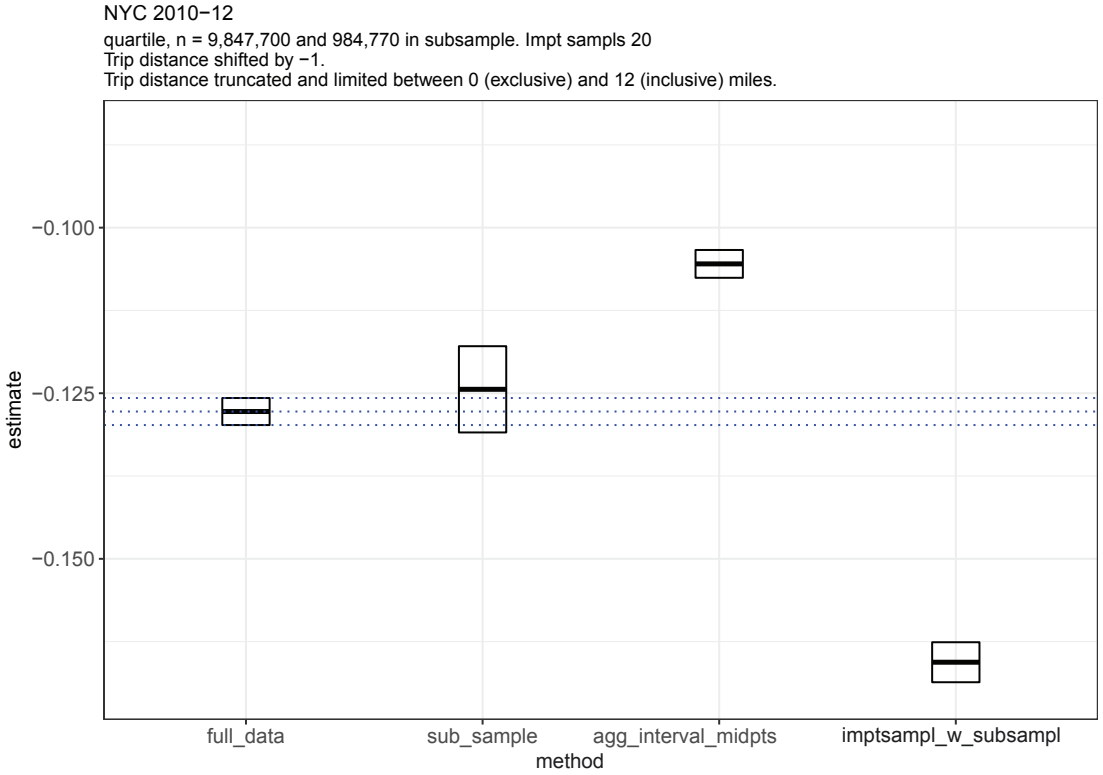


Figure 3.11: *Coefficient estimate. Quartile. Exponential.*

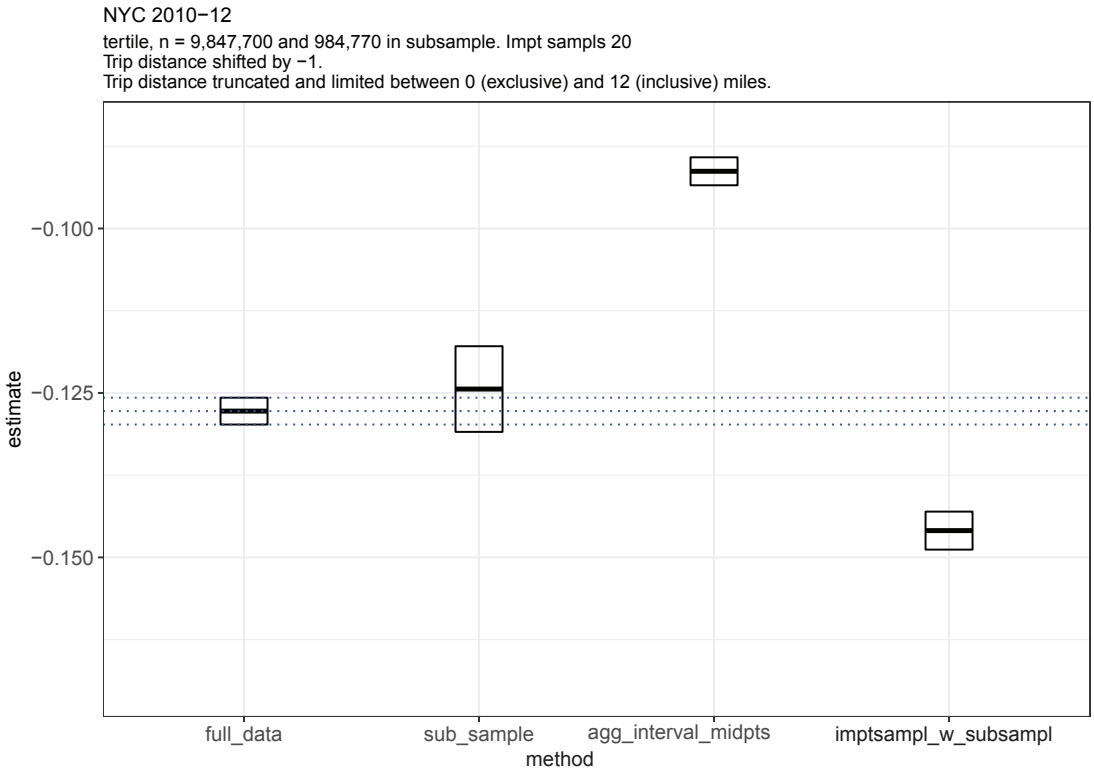


Figure 3.12: *Coefficient estimate. Tertile. Exponential.*

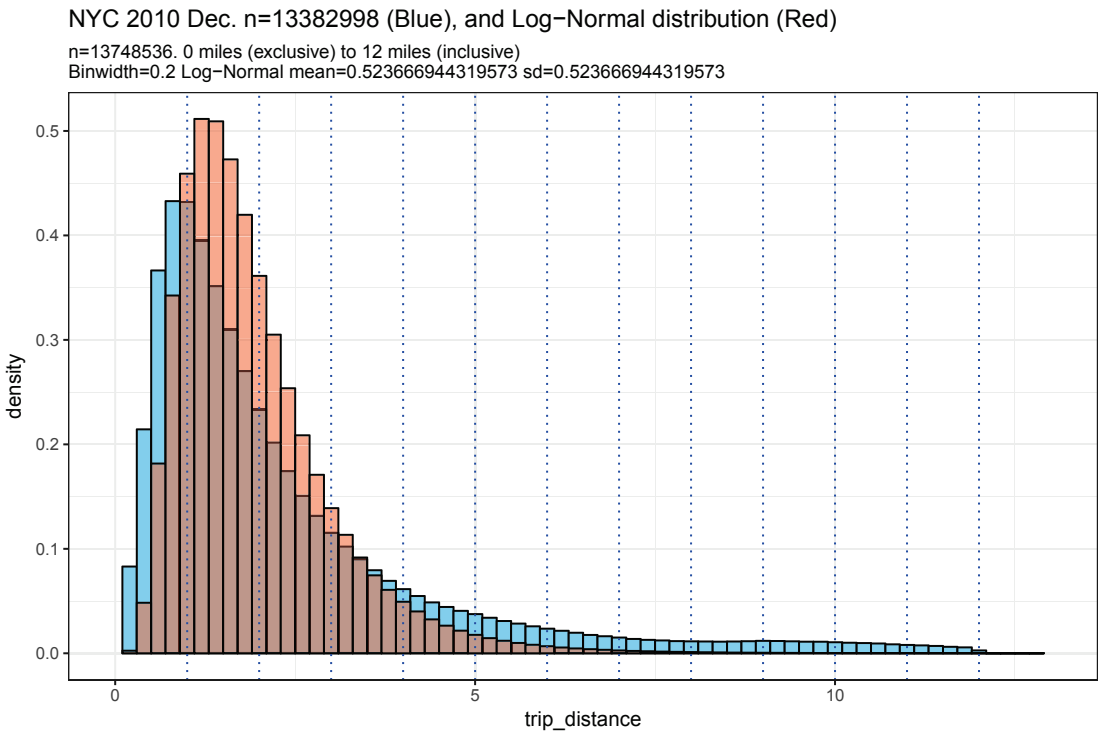


Figure 3.13: *Modelling trip distance with Log-Normal distribution*

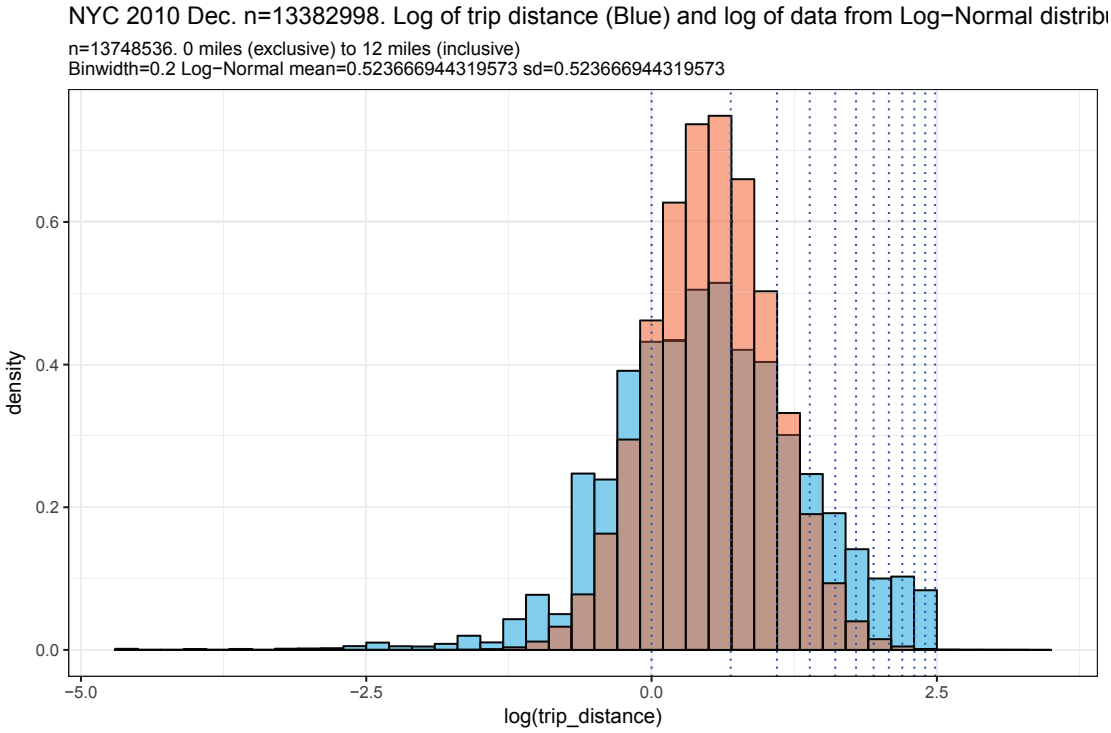


Figure 3.14: *Modelling log of trip distance*

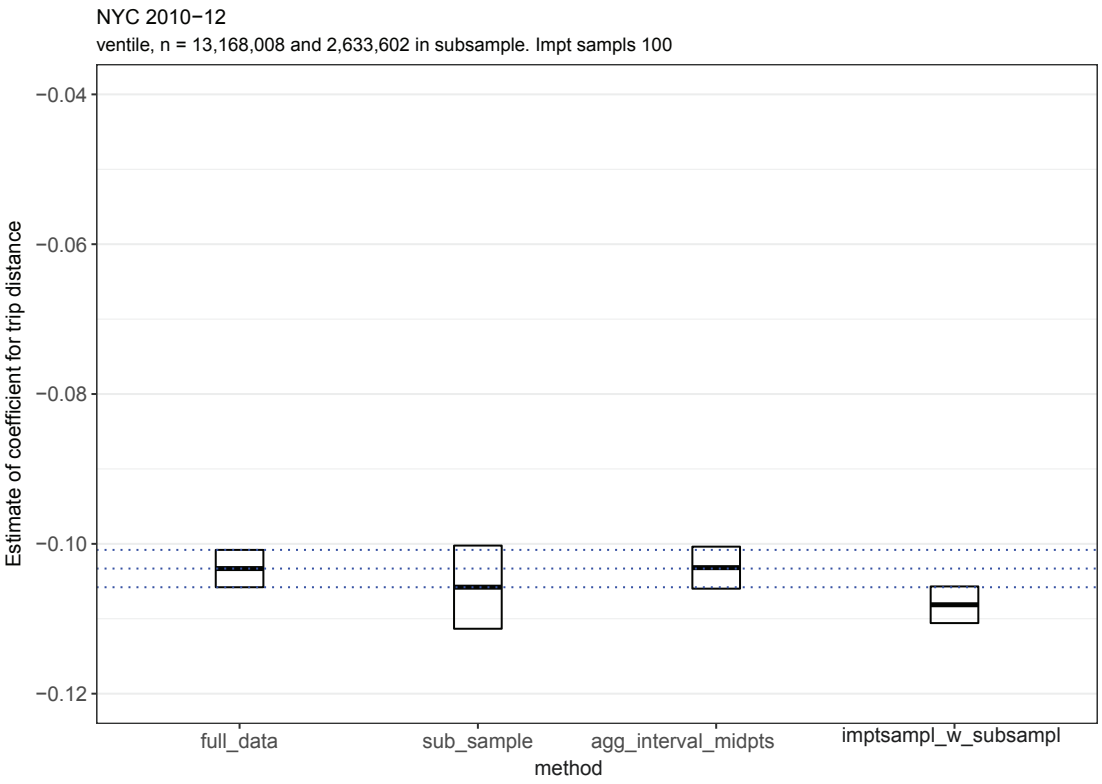


Figure 3.15: *Coefficient estimate. Ventile. Log Normal.*

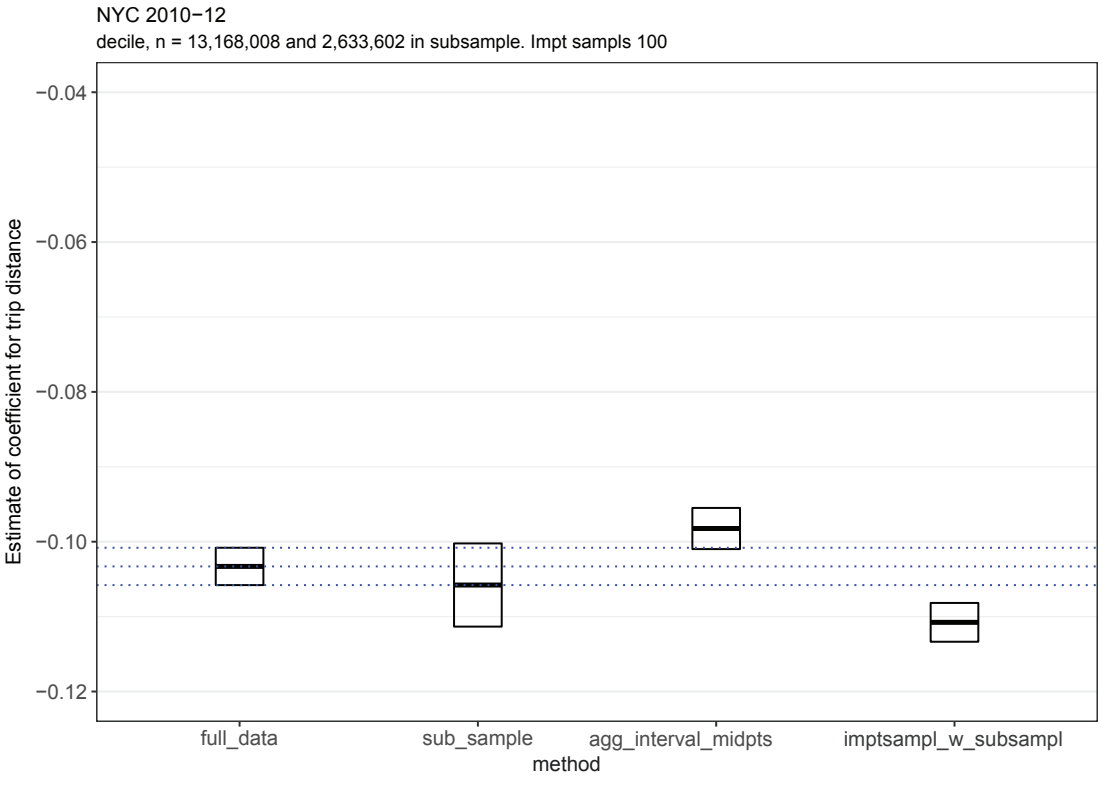


Figure 3.16: *Coefficient estimate. Decile. Log Normal.*

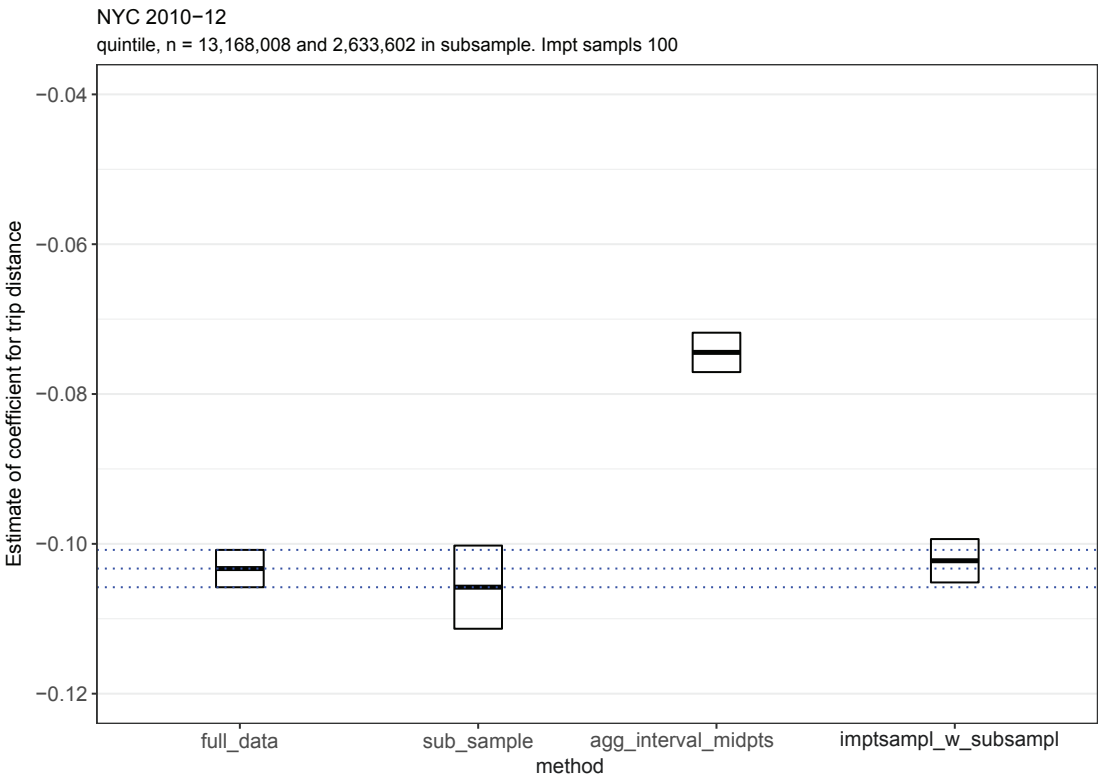


Figure 3.17: *Coefficient estimate. Quintile. Log Normal.*

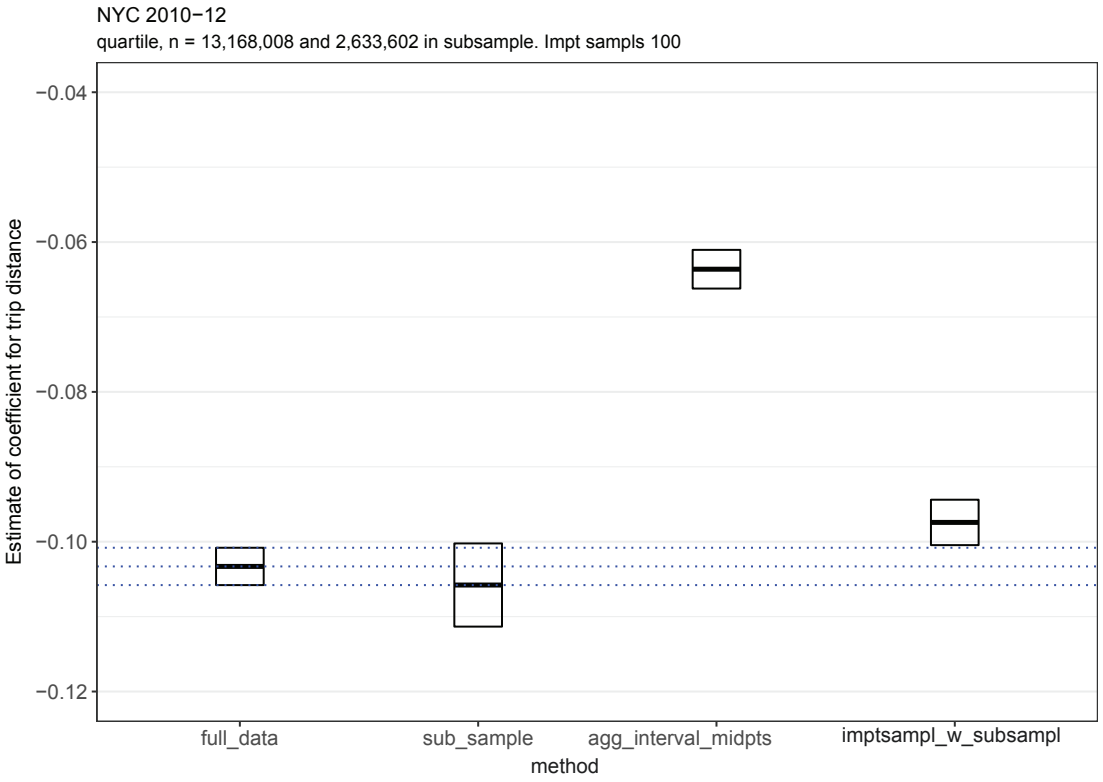


Figure 3.18: *Coefficient estimate. Quartile. Log Normal.*

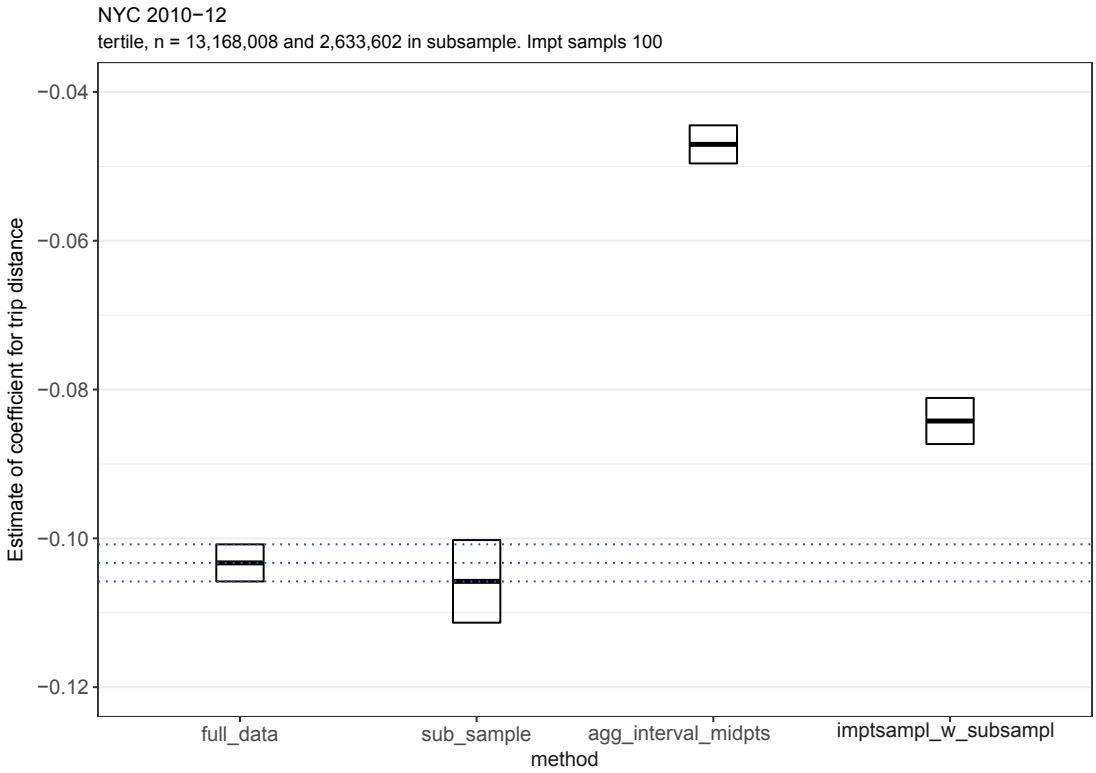


Figure 3.19: *Coefficient estimate. Tertile. Log Normal.*

3.7 Discussion

In this chapter, we have outlined how a method of coarsening and aggregating the data, followed by application of EM algorithm by the method of weights to allow logistic regression model to be fitted using large datasets in a timely manner on a typical desktop computer. We also show that due to coarsening and aggregation, even with complete data model approximation through EM algorithm, we expect the coefficient estimates to be biased. As we have used a relatively simple model to explore the technique, there are many areas of future work.

The regression model in the simulation is simple in the sense that it only include two covariates, including the covariate we coarsen. The regression model is simple in order to allow us to examine the techniques on a small scale. An obvious extension is to include additional covariates in the regression model. Changes to the regression model can impact many aspects of the work in this chapter from the coarsening to the importance sampling. When we coarsen multiple covariates, the the number of unique combination of the coarsened covariates can increase. For instance, suppose we coarsen two covariates into five levels (quintiles).

With this setup, we will have $5 \times 5 = 25$ different unique levels. If we coarsen another covariate, the number of unique combination then becomes $5 \times 5 \times 5 = 125$ levels. If we generate $m = 100$ importance samples per combination, the number of observations we use in the EM algorithm will increase from $25 \times 20 = 500$ to $125 \times 100 = 12,500$ observations. The number of coarsened covariates can also impact the convergence of the EM algorithm. As the number of covariates increases, the conditional expectation with respect to the observed data get more complicated to derive. We also have seen from the application section that selecting a correct parametric distribution for the coarsened covariate is quite challenging. Related to this is the conditional expectation itself. In this work, we only coarsen one covariate that we coarsen, if we coarsen multiple covariates, the joint density of the coarsened covariates can become quite complex. For instance if we coarsen two covariates, the conditional density becomes $f(x, u|\text{observed}) = f(x, u \mid x^*, u^*, y, \boldsymbol{z})$. In addition, coarsening multiple covariates can also introduce computational complexity to the importance sampling. To sample a more complex integral form of the conditional expectation, we will need to reconsider the use of Uniform distribution as the importance function.

As for the importance sampling procedure, a future work is to further examine the effect of the number of importance samples to the convergence of the EM algorithm. Increasing the the number of importance samples, from 100 used in the application section, may result in better approximation to the density of the trip distance covariate. We can generate many more importance samples without encountering issues regarding storage, with the tradeoff of possibly longer computational time. Another aspect concerning importance sampling is the importance function. In this chapter, the uniform distribution we have used may not be the optimal distribution. An optimal importance function is desirable because it can reduce the variance of the approximation. However, finding the optimal importance function is difficult in our case because we are also guessing the density of the coarsened covariate (we used the Exponential and LogNormal distribution). As such if we want an optimal importance function, we need to find one for each distribution of coarsened covariate that we try.

We can of course improve our approximation to the true density of the coarsened covariate. In this work, we used the Exponential and LogNormal distributions to approximate the density of coarsened covariate, trip distance. The long tail nature of this covariate means it was difficult to approximate its true density. Similar challenge exists in the medical field where the long tail nature of length of stay in hospitals of patients poses challenges. Similar to this work, Marazzi et al. (1998) used a parametric modelling approach by using different distributions to model the distribution of length of stay at hospital. However such approach assumes that regardless of the diagnoses, all patients have the similar length of stay in hospital (Atienza et al., 2008). To overcome this assumption, finite mixture of densities are used to model the

length of stay, such as work by Atienza et al. (2008). In a similar manner, we can also model along similar lines for the NYC Taxi data used in the application section by not assuming that all the passengers have similar trip distances. The downsides of using mixture models include complexity of the model and the increase in the number of parameters to estimate.

Apart from the long tail nature of the distribution, logistic regression using the 2010 December NYC Taxi trip data set had the issue of *separation* (Albert and Anderson, 1984), where the outcomes can be perfectly separated by one or more covariates (Heinze and Schemper, 2002; Zorn, 2005). In our case, from examining the random sub-sample, we find that when passengers do tip the driver, they predominately used credit card. For example, out of 1,639,788 trips where the driver received a tip, only 909 (0.055%) of those were paid by cash. As a result, the coefficient estimate of cash payment type has an extremely low odds ratio value (0.0000167017). To see the how separation affects the estimate of coefficients from EM by method of weights, we simulated a dataset with less extreme proportion of payment types (e.g., 10% of tips were paid by cash) then applied our method. The results from Figure 3.20 to Figure 3.24 illustrate that when separation do not occur in the logistic regression, the estimates of coefficients from our method match closely to their complete data counterparts. The works by Zorn (2005) and Heinze and Schemper (2002) illustrate the various methods to address the issue of separation in the areas of social sciences and medicine respectively. The first option is to remove the covariate from the analysis, which is not always possible, such as in your application section. The second approach is to perform some data manipulation by adding 'artificial' data, but it is inferior to other approaches in dealing with separation (Zorn, 2005). The third approach, exact logistic regression (Cox and Snell, 1989; Mehta and Patel, 1995) is only applicable if all the covariates in the regression model are categorical (this method might work for our model). Instead of removing or manipulating the covariates, both Zorn (2005) and Heinze and Schemper (2002) favour a penalized likelihood method by Firth (1993). In this method, a bias term is added to the likelihood function which counteracts the effect of separation on the likelihood (Zorn, 2005). In GLM context with canonical link, this bias term has Bayesian interpretation of having a Jeffrey invariant prior (Jeffreys, 1946) in the model (Heinze and Schemper, 2002; Zorn, 2005). Das, Maiti, and Pradhan (2010) applied this likelihood penalty method by Firth (1993) to GLM with missing covariate. We have not applied the likelihood penalty because the issue of separation arises from the specific data set we used in the application section, rather than a general issue with covariate discretisation and EM by method of weights.

In this chapter, we put aside 20% of the full data as the sub-sample, adopting the convention of in machine learning applications, where 20% of the data is used as training set. Although we simulated the effect of varying coarsening and importance sample size, we have

not explored the effect of varying sub-sample size. From our simulation the sub-sample is important as it reduces the bias in our method, as well as improving convergence of the EM algorithm. An exploration of changing sample size can potentially show the level of influence the sub-sample has on the estimates, especially when multiple covariates are coarsened. Another area of influence of the sub-sample is the effect on the computational speed. The EM algorithm can be slow to converge in situations where there are many missing data. Since we are using the sub-sample to speed up the convergence, a sensitivity analysis can also show the level of mitigation on the computation limitation of the EM algorithm.

We can also change the way we obtain the sub-sample $\mathbb{D}_s$, using other techniques other than random sampling. We can use techniques from sample survey to obtain more representative samples of the complete full data. This is useful for two reasons. Firstly, examining Figure 3.2, we can see that in cases of very fine coarsening, the estimate of the coefficient obtained from the sub-sample is comparable to that of EM by method of weights; although there are more variations. This suggested that in some cases, using data from a representative sub-sample to perform regression might be enough. E.g., using other sub-sampling scheme or other sampling technique such as *coresets* (Campbell and Broderick, 2017), which are a weighted subset of the data, where the weighted are calculated such that the weighted log-likelihood approximates the the log-likelihood of the original dataset. Secondly, when the outcome variable we are looking for has low probability of success, using other methods than random sub-sampling may become necessary because in these cases, random sampling may fail to capture enough samples with success outcome, which means our $\mathbb{D}_s$ become less useful. We can obtain the sub-sample by using sampling techniques from the medical field such as case control sampling. By not relying on simple random sampling to obtain the sub-sample $\mathbb{D}_s$, we can perform regression analysis when the outcomes have low probabilities. For example, Wright, Ryan, and Pham (2018) use case control sampling to obtain representative samples with low probability outcomes.

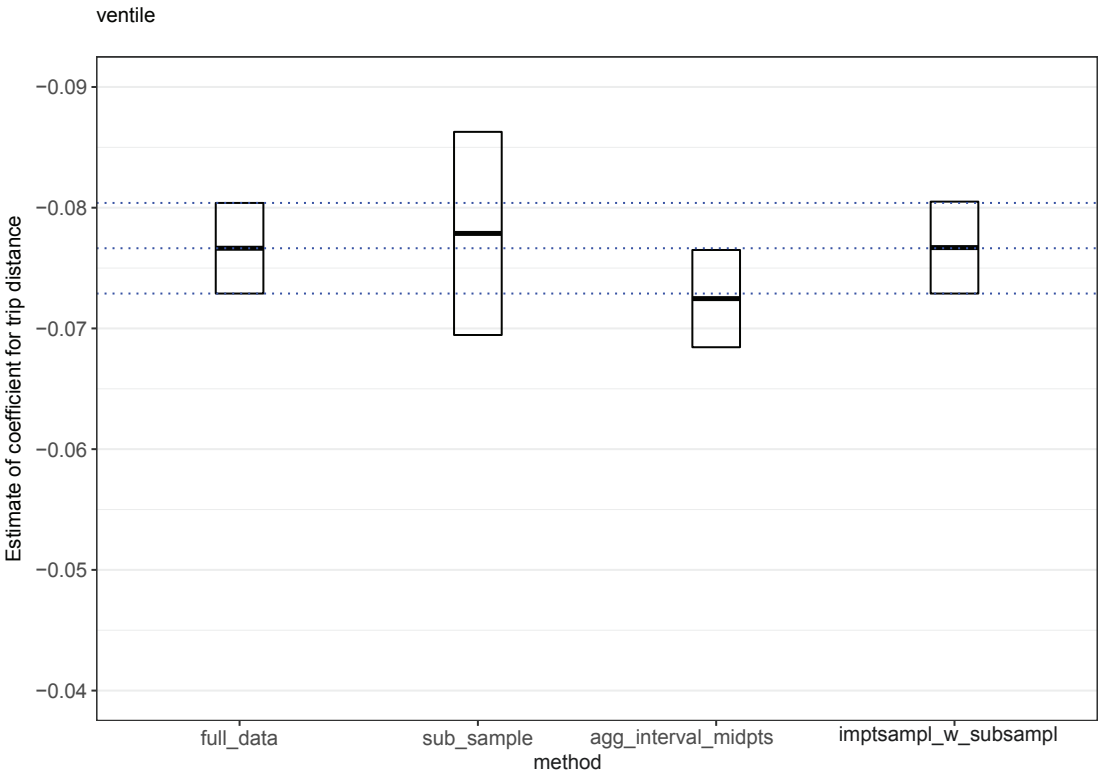


Figure 3.20: *Estimate of coefficient from simulation of 500,000 observations with a less extreme coefficient for cash payment type. Trip distance covariate is coarsened into ventiles (20 quantile groups). 10% of simulated trips that resulted in a tip to the driver paid by cash. We expect with very fine coarsening method, the estimate from EM by method of weights should match the estimate from using the complete data.*

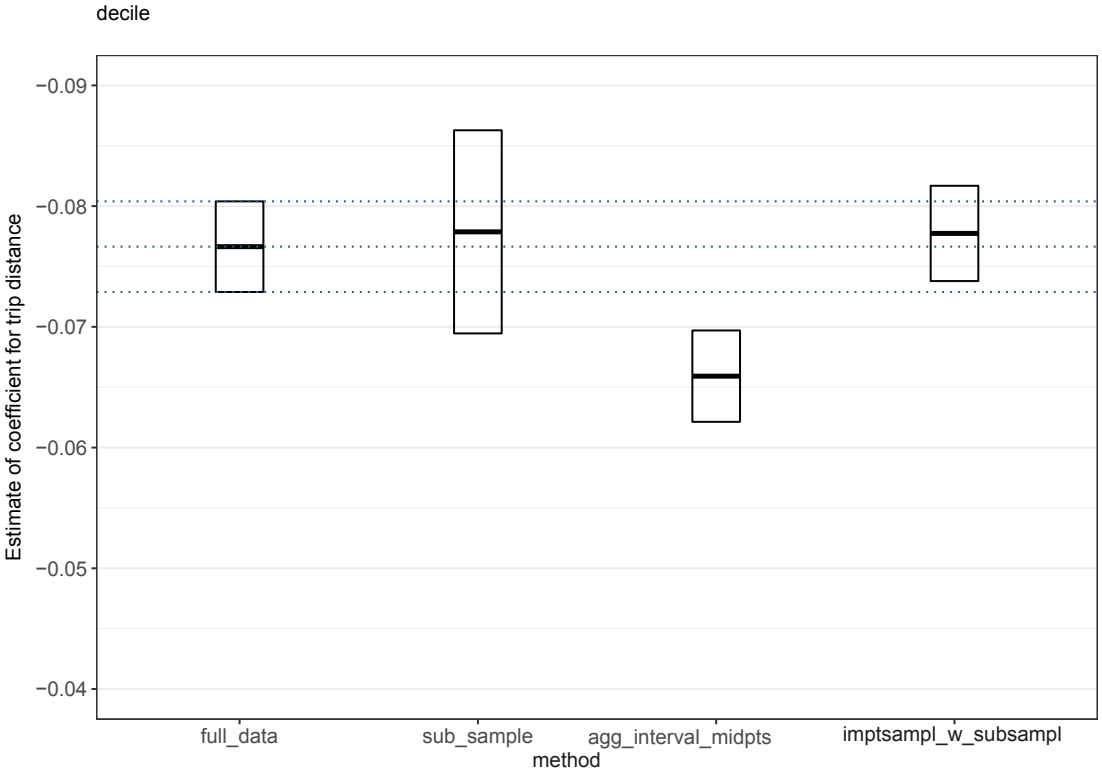


Figure 3.21: *Estimate of coefficient from simulation of 500,000 observations with a less extreme coefficient for cash payment type. Trip distance covariate is coarsened into deciles (10 quantile groups). 10% of simulated trips that resulted in a tip to the driver paid by cash.*

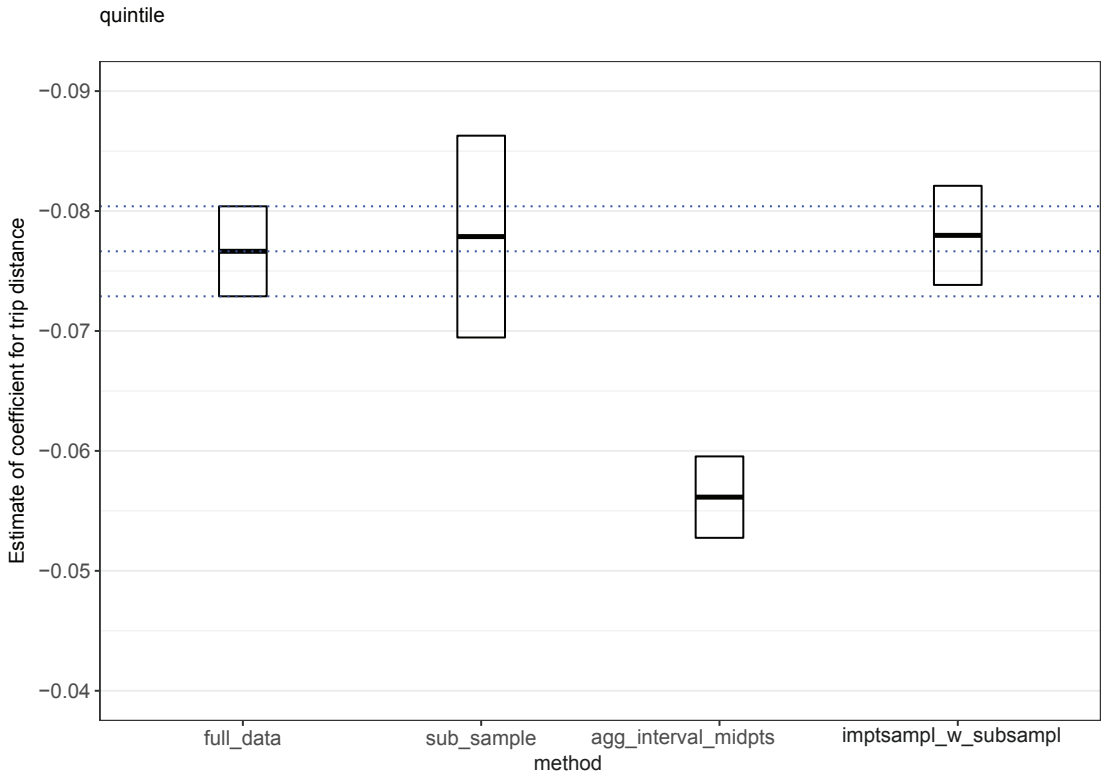


Figure 3.22: *Estimate of coefficient from simulation of 500,000 observations with a less extreme coefficient for cash payment type. Trip distance covariate is coarsened into quintile (5 quantile groups). 10% of simulated trips that resulted in a tip to the driver paid by cash. Notice that as the coarsening intervals get wider, more bias can be seen from estimate that use the mid-point of the coarsened interval.*

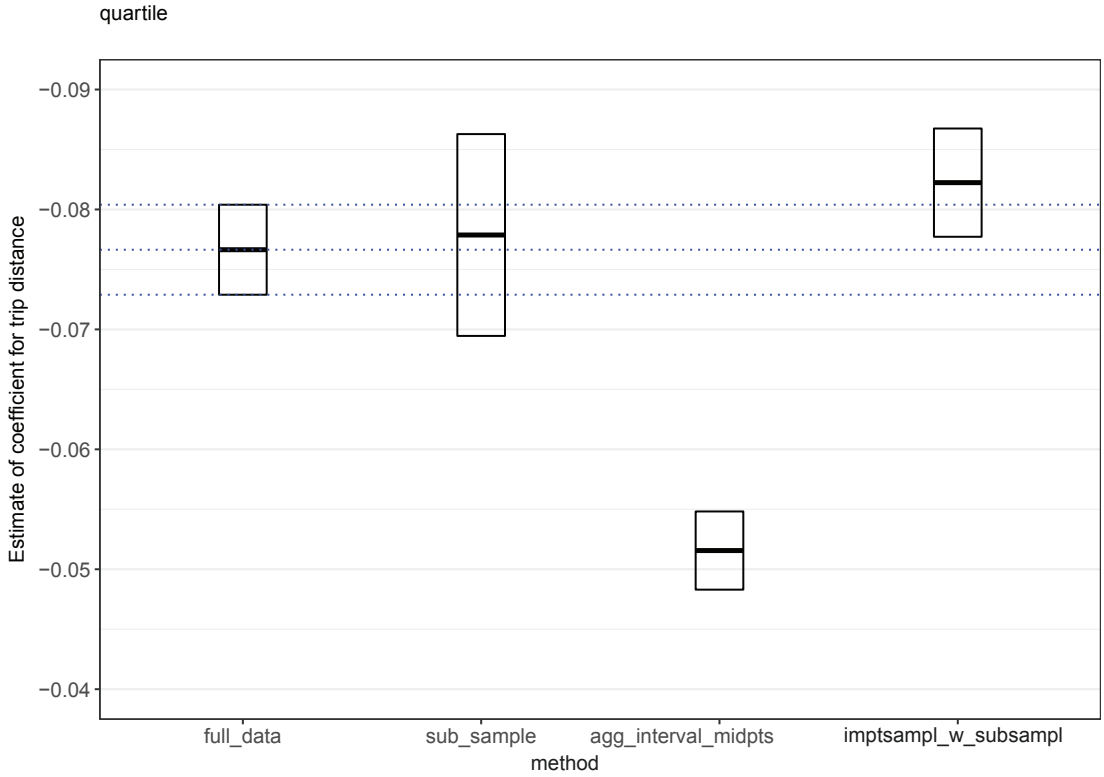


Figure 3.23: *Estimate of coefficient from simulation of 500,000 observations with a less extreme coefficient for cash payment type. Trip distance covariate is coarsened into quartile (4 quantile groups). 10% of simulated trips that resulted in a tip to the driver paid by cash.*

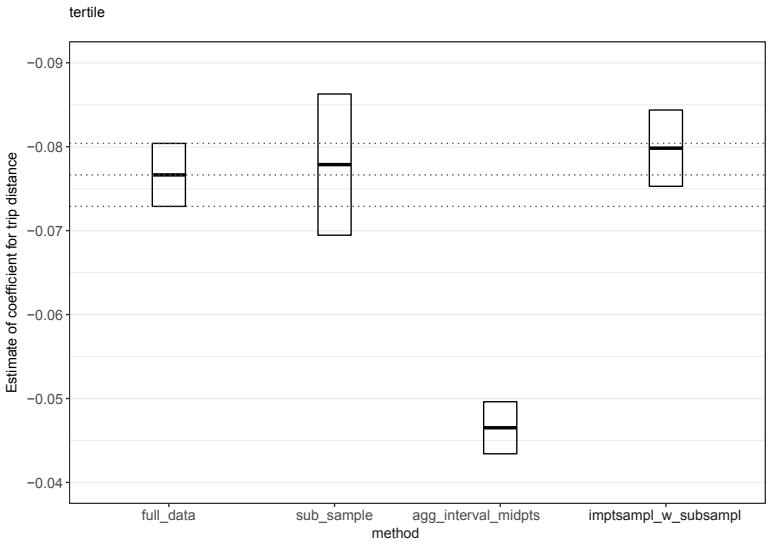


Figure 3.24: *Estimate of coefficient from simulation of 500,000 observations with a less extreme coefficient for cash payment type. Trip distance covariate is coarsened into tertile (4 quantile groups). 10% of simulated trips that resulted in a tip to the driver paid by cash.*

Appendix 3.A Derivation of Score Equations to Estimate β

The model for Y is

$$Y \sim \text{Bern}(\pi) \,,$$

where π is the probability of a *success* outcome. μ_i is defined as

$$\text{E}(Y_i) = \mu_i^{(y)}$$

Let the linear term, η, for the ith observation be:

$$\eta_i^{(y)} = \beta_0 + \beta_1 x_{i1} + \beta_2 z_{i1} + \ldots + \beta_p z_{ip}$$

Note that we'll coarsen x later on, and the intercept term belongs with covariates $\mathbf{z}$. Let $g(\mu_i)$ be the link function. Using the canonical logit link. We'll relate μ_i to the linear terms as follows:

$$\begin{aligned} g(\mu_i^{(y)}) &= \eta_i^{(y)} \\ \text{logit}(\mu_i^{(y)}) &= \eta_i^{(y)} \\ \mu_i^{(y)} = g^{-1}(\mu_i^{(y)}) &= \frac{\exp(\eta_i^{(y)})}{1 + \exp(\eta_i^{(y)})} \end{aligned}$$

The general form of the GLM Score Equation for the jth covariate, according to Dobson and Barnett (2008, p.65) is

$$\text{S}_j = \sum_{i=1}^{n} \left[\frac{y_i - \mu_i^{(y)}}{\text{Var}(Y_i)} \; x_{ij} \left(\frac{\partial \mu_i^{(y)}}{\partial \eta_i^{(y)}} \right) \right]$$

In the *complete data case*, we have x_i and we've shown μ_i above, so we'll need $\text{Var}(Y_i)$ and $\Big(\partial \mu_i\Big) \Big/ \Big(\partial \eta_i\Big)$. From the properties of the Bernoulli Distribution, we have

$$\begin{aligned} \text{Var}(Y_i) &= \pi(1 - \pi) \\ &= \text{E}(Y_i)[1 - \text{E}(Y_i)] \\ &= \mu_i^{(y)}(1 - \mu_i^{(y)}) \end{aligned}$$

since

$$\mu_i = \frac{\exp(\eta_i^{(y)})}{1 + \exp(\eta_i^{(y)}))} \,,$$

the variance is

$$\begin{aligned}
\text{Var}(Y_i) &= \mu_i^{(y)}(1-\mu_i^{(y)}) \\
&= \left[\frac{\exp(\eta_i^{(y)})}{1+\exp(\eta_i^{(y)})}\right]\left[1-\frac{\exp(\eta_i^{(y)})}{1+\exp(\eta_i^{(y)})}\right] \\
&= \left[\frac{\exp(\eta_i^{(y)})}{1+\exp(\eta_i^{(y)})}\right]\left[\frac{1}{1+\exp(\eta_i^{(y)})}\right] \\
&= \frac{\exp(\eta_i^{(y)})}{\left[1+\exp(\eta_i^{(y)})\right]^2} \quad .
\end{aligned}$$

Also

$$\begin{aligned}
\frac{\partial \mu_i^{(y)}}{\partial \eta_i^{(y)}} &= \frac{\exp(\eta_i^{(y)})}{\left[1+\exp(\eta_i^{(y)})\right]^2} \\
&= \text{Var}(Y_i) \ ,
\end{aligned}$$

which will greatly help when we substitute these into the general Score Equation for the jth covariate, which takes the form

$$\begin{aligned}
\text{S}_j &= \sum_{i=1}^{n}\left[\frac{y_i-\mu_i^{(y)}}{\text{Var}(Y_i)}\ x_{ij}\ \left(\frac{\partial \mu_i^{(y)}}{\partial \eta_i^{(y)}}\right)\right] \\
&= \sum_{i=1}^{n}\left[\frac{y_i-\mu_i^{(y)}}{\text{Var}(Y_i)}\ x_{ij}\ \text{Var}(Y_i)\right] \\
&= \sum_{i=1}^{n}\left[(y_i-\mu_i^{(y)})x_{ij}\right] \ .
\end{aligned}$$

We have an intercept term and one covariate that we'll coarsen (x), and one covariate we *won't* coarsen, z_1. So we'll have three Score Equations in this model, where the x_{ij} term in the general Score Equation takes on three values

$$\begin{aligned}
x_{i,j=1} &= 1 \\
x_{i,j=2} &= x_i \\
x_{i,j=3} &= z_{i1}
\end{aligned}$$

That is,

$$S_1 = \sum_{i=1}^{n} \left[(x_i - \mu_i^{(y)}) \right] ,$$

$$S_2 = \sum_{i=1}^{n} \left[(x_i - \mu_i^{(y)}) x_i \right] ,$$

and

$$S_3 = \sum_{i=1}^{n} \left[(x_i - \mu_i^{(y)}) z_{i1} \right] .$$

If we factor out the intercept and the covariates, then place them in a vector, the *vector* of Score Equations, for the ith observation becomes

$$\mathbf{S}_i = \begin{bmatrix} 1 \\ x_i \\ z_{i1} \end{bmatrix} \{ x_i - \mu_i^{(y)} \} .$$

We can emphasis $\mu_i^{(y)}$ as a function of $\eta_i^{(y)}$:

$$\mathbf{S}_i = \begin{bmatrix} 1 \\ x_i \\ z_{i1} \end{bmatrix} \{ x_i - \mu_i^{(y)}(\eta_i^{(y)}) \}$$

$$\mathbf{S}_i = \begin{bmatrix} 1 \\ x_i \\ z_{i1} \end{bmatrix} \{ x_i - \mu_i^{(y)}(x_i, z_{i1}) \}$$

$$\mathbf{S}_i = \begin{bmatrix} 1 \\ x_i \\ z_{i1} \end{bmatrix} \left\{ x_i - \left[\frac{\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 z_{i1})}{1 + \exp(\beta_0 + \beta_1 x_{i1} + \beta_2 z_{1i})} \right] \right\}$$

Appendix 3.B Derivation of Score Equations to Estimate α

The model is

$$X \sim \text{Exp}(\lambda) ,$$

where λ is the rate parameter for Exponential Distribution. $\mu_i^{(x)}$ is then

$$\text{E}(X_i) = \mu_i^{(x)}$$

We'll assume there is only one z. Following the GLM notation, the linear term is

$$\eta_i^{(x)} = \alpha_0 + \alpha_1 z_{1i} \ .$$

Let $g(\mu_i^{(x)})$ be the link function. For this model, we'll *not* use the canonical link (inverse function). We'll use the log link instead. We'll relate $\mu_i^{(x)}$ to the linear terms as follows:

$$\begin{aligned} g(\mu_i^{(x)}) &= \eta_i^{(x)} \\ \ln(\mu_i^{(x)}) &= \eta_i^{(x)} \\ \mu_i^{(x)} &= g^{-1}(\mu_i^{(x)}) = \exp(\eta_i^{(x)}) \end{aligned}$$

Note that since $\mu_i^{(x)}$ is a function of $\eta_i^{(x)}$, $\mu_i^{(x)}$ is a function of z and $\boldsymbol{\alpha}$.

The general form of the GLM Score Equation for the jth covariate, according to Dobson and Barnett (2008, p.65) is

$$\mathrm{S}_j = \sum_{i=1}^{n} \left[\frac{x_i - \mu_i^{(x)}}{\mathrm{Var}(X_i)} \ z_{ij} \left(\frac{\partial \mu_i^{(x)}}{\partial \eta_i^{(x)}} \right) \right]$$

In the *complete data case*, we have z_i and we've shown $\mu_i^{(x)}$ above, so we'll need $\mathrm{Var}(X_i)$ and $\left(\partial \mu_i^{(x)}\right) \Big/ \left(\partial \eta_i^{(x)}\right)$. Earlier in this section we state that we are modelling $X \sim \mathrm{Exp}(\lambda)$, where λ is the *rate* parameter. Therefore, from the property of the Exponential distribution, we have

$$\mathrm{Var}(X_i) = \frac{1}{\lambda^2} \ .$$

Let's re-express $\mathrm{Var}(X_i)$ using $\mathrm{E}(X_i)$, which hopefully will make the algebra easier later on. First,

$$\mathrm{E}(X_i) = \frac{1}{\lambda} \ .$$

So,

$$\mathrm{Var}(X_i) = \frac{1}{\lambda^2} = [\mathrm{E}(X_i)]^2 = (\mu_i^{(x)})^2$$

We also derived earlier that $\mu_i^{(x)} = \exp(\eta_i^{(x)})$. Therefore

$$\frac{\partial \mu_i^{(x)}}{\partial \eta_i^{(x)}} = \exp(\eta_i^{(x)})$$

Let's put all these together into the general form. We'll delay substituting the values of the

covariates (z_i) for now.

$$\begin{aligned}
\mathrm{S}_j &= \sum_{i=1}^{n} \left[\frac{x_i - \mu_i^{(x)}}{\mathrm{Var}(X_i)} \; z_{ij} \left(\frac{\partial \mu_i^{(x)}}{\partial \eta_i^{(x)}} \right) \right] \\
&= \sum_{i=1}^{n} \left[\frac{x_i - \exp(\eta_i^{(x)})}{[\exp(\eta_i^{(x)})]^2} \; z_{ij} \; \exp(\eta_i^{(x)}) \right] \\
&= \sum_{i=1}^{n} \left[\frac{x_i - \exp(\eta_i^{(x)})}{\exp(\eta_i^{(x)})} \; z_{ij} \right] \\
&= \sum_{i=1}^{n} \left[\frac{x_i - \mu_i^{(x)}}{\mu_i^{(x)}} \; z_{ij} \right]
\end{aligned}$$

Now we have an intercept term and one covariate (z_1), so we'll have two Score Equations in this model. So $z_{i,j=1} = 1$ and $z_{i,j=2} = z_{i1}$. That is,

$$\mathrm{S}_1 = \sum_{i=1}^{n} \left[\frac{x_i - \mu_i^{(x)}}{\mu_i^{(x)}} \; 1 \right],$$

and

$$\mathrm{S}_2 = \sum_{i=1}^{n} \left[\frac{x_i - \mu_i^{(x)}}{\mu_i^{(x)}} \; z_{i1} \right].$$

If we factor out the intercept and the covariate and put them into a vector, the *vector* of Score Equations, for the ith observation becomes

$$\mathbf{S}_i = \begin{bmatrix} 1 \\ z_{i1} \end{bmatrix} \left\{ \frac{x_i - \mu_i^{(x)}}{\mu_i^{(x)}} \right\}.$$

We can emphasis $\mu_i^{(x)}$ as a function of $\eta_i^{(x)}$ as follows

$$\mathbf{S}_i = \begin{bmatrix} 1 \\ z_{i1} \end{bmatrix} \left\{ \frac{x_i - \mu_i^{(x)}(z_{i1})}{\mu_i^{(x)}(z_{i1})} \right\}.$$

3.B.1 GLM Score Equations when $x \sim$ LogNormal

An alternative to modelling x with Gaussian distribution and log link is to model the natural log of x with the Gaussian distribution. The estimation of the parameters then become performing GLM with Gaussian family and identity link. The GLM Score Equation for this

modelling is

$$\mathbb{S}_2(\boldsymbol{\alpha}) = \begin{bmatrix} 1 \\ \mathbf{z}_i \end{bmatrix} \left(\frac{1}{\sigma^2} \{ \ln(x_i) - \mu_i(\boldsymbol{z}_i\,;\,\boldsymbol{\alpha}) \} \right), \tag{3.17}$$

where $\mu_i = \alpha_0 + \alpha_1 z_i$ for the intercept and one $\boldsymbol{z}$ term.

Appendix 3.C Conditional Expectation of GLM Score Equations

3.C.1 Estimation Standard Errors for $\boldsymbol{\beta}$

As shown in Section3.3.4, when we estimate the parameters, we use the conditional expectation of the score equations. The conditional expectation of the vector of the score equations (Equation 3.2) to estimate $\boldsymbol{\beta}$ is

$$\begin{aligned} \mathrm{E}_{x|\text{observed}}(\boldsymbol{s}_{1i}) &= \mathrm{E}_{x_i|\text{obs.}} \left\{ \begin{bmatrix} 1 \\ x_i \\ z_{i1} \end{bmatrix} (y_i - \mu_i(x_i, z_{i1})) \right\} \\ &= \begin{bmatrix} \mathrm{E}_{x_i|\text{obs.}} \{ y_i - \mu_i(x_i, z_{i1}) \} \\ \mathrm{E}_{x_i|\text{obs.}} \{ [y_i - \mu_i(x_i, z_{i1})] x_i \} \\ \mathrm{E}_{x_i|\text{obs.}} \{ [y_i - \mu_i(x_i, z_{i1})] z_{i1} \} \end{bmatrix} = \begin{bmatrix} \mathrm{E}_{x_i|\text{obs.}} \{ \boldsymbol{s}_{1i(1)} \} \\ \mathrm{E}_{x_i|\text{obs.}} \{ \boldsymbol{s}_{1i(2)} \} \\ \mathrm{E}_{x_i|\text{obs.}} \{ \boldsymbol{s}_{1i(3)} \} \end{bmatrix} . \end{aligned} \tag{3.18}$$

Note that we have abbreviated the word observed with *obs.*. What is observed in this case are the outcome y_i, coarsened covariate x_i^* and un-coarsened covariates $\mathbf{z}_i$. Expanding the conditional expectations as follows

$$\mathrm{E}_{x|\text{observed}} \{ \boldsymbol{s}_{1i(1)} \} = y_i - \mathrm{E}_{x|\text{observed}} \{ \mu_i(x_i, z_{1i}) \} , \tag{3.19}$$

$$\mathrm{E}_{x|\text{observed}} \{ \boldsymbol{s}_{1i(2)} \} = \mathrm{E}_{x|\text{observed}} \{ [y_i - \mu_i(x_i, z_{1i})] x_i \} , \tag{3.20}$$

$$\begin{aligned} \mathrm{E}_{x|\text{observed}} \{ \boldsymbol{s}_{1i(3)} \} &= \mathrm{E}_{x|\text{observed}} \{ [y_i - \mu_i(x_i, z_{1i})] z_{1i} \} \\ &= \mathrm{E}_{x|\text{observed}} [y_i - \mu_i(x_i, z_{1i})]\, z_{1i} \\ &= \{ y_i - \mathrm{E}_{x|\text{observed}} [\mu_i(x_i, z_{1i})] \}\, z_{1i} . \end{aligned} \tag{3.21}$$

3.C.2 Incorporating Importance Sampling

If we expand the conditional expectations from Equation 3.19 to Equation 3.21, we see that we need $\mathrm{E}_{x|\text{observed}}(x_i)$ and $\mathrm{E}_{x|\text{observed}}[\mu_i(x_i, z_{1i})]$. For example, expanding Equation 3.19

$$\begin{aligned}
\mathrm{E}_{x|\text{observed}}\{\boldsymbol{s}_{1i(1)}\} &= y_i - \mathrm{E}_{x|\text{observed}}[\mu_i(x_i, z_{1i})] \\
&= y_i - \int_{\chi} \mu_i(x_i, z_{1i})\, f(x_i|\text{observed})\, \mathrm{d}x \\
&= y_i - \int_{\chi} \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 z_{i1})}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 z_{i1})} f(x_i|\text{observed})\, \mathrm{d}x \ .
\end{aligned}$$

where χ is the support of $f(x_i|\text{observed})$. As shown in Section3.3.5, obtaining closed form solutions to the conditional expectations is impossible. As a result, we use Importance Sampling to approximate $\mathrm{E}_{x|\text{observed}}[\mu_i(x_i, z_{1i})]$ such that

$$\mathrm{E}_{x|\text{observed}}[\mu_i(x_i, z_{1i})] \approx \sum_{k=1}^{m} \mu_i(x_k, z_{i1})\, w_k \ ,$$

where m is the number of *importance samples* and w_k is the kth *normalised weight.*

3.C.3 Conditional Expectations with Importance Sampling

Incorporating importance sampling, the conditional expectations of the Score Equations in Equation 3.19 through to Equation 3.21 become

$$\begin{aligned}
\mathrm{E}_{x|\text{observed}}\{\boldsymbol{s}_{1i(1)}\} &= y_i - \mathrm{E}_{x|\text{observed}}[\mu_i(x_i, z_{1i})] \\
&\approx y_i - \sum_{k=1}^{m} [\, \mu_i(x_{ik}, z_{1i})\, w_k] \qquad (3.22)
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}_{x|\text{observed}}\{\boldsymbol{s}_{1i(2)}\} &= \mathrm{E}_{x|\text{observed}}\{[y_i - \mu_i(x_i, z_{i1})]\, x_i\} \\
&\approx \sum_{k=1}^{m} \left[\left\{ [y_i - \mu_i(x_{ik}, z_{1i})]\, x_{ik} \right\} w_k \right] \qquad (3.23)
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}_{x|\text{observed}}\{\boldsymbol{s}_{1i(3)}\} &= \{y_i - \mathrm{E}_{x|\text{observed}}[\mu_i(x_i, z_{1i})]\, \} z_{1i} \\
&\approx \left[y_i - \sum_{k=1}^{m} \mu_i(x_{ik}, z_{1i})\, w_k \right] z_{1i} \qquad (3.24)
\end{aligned}$$

We use these approximations to obtain the conditional expectations of the vector of Score Equations $\boldsymbol{s}_1$.

Appendix 3.D Aggregation of Outer Products of Score Equations

Suppose we have the following vectors:

$$\boldsymbol{m}_j = \begin{pmatrix} a_j \\ b_j \\ c_j \end{pmatrix}$$

Their corresponding outer product (e.g., $\boldsymbol{m}_j\, \boldsymbol{m}_j^\top$) would be:

$$\boldsymbol{m}_j\, \boldsymbol{m}_j^\top = \begin{pmatrix} a_j^2 & a_j b_j & a_j c_j \\ a_j b_j & b_j^2 & b_j c_j \\ a_j c_j & b_j c_j & c_j^2 \end{pmatrix}$$

The summation of the outer products can be achieved by multiplying the outer product of a single summation with the number of summations.

$$\begin{aligned}
\sum_{j=1}^{2} (\boldsymbol{m}_j\, \boldsymbol{m}_j^\top) &= \begin{pmatrix} a_j^2 & a_j b_j & a_j c_j \\ a_j b_j & b_j^2 & b_j c_j \\ a_j c_j & b_j c_j & c_j^2 \end{pmatrix} + \begin{pmatrix} a_j^2 & a_j b_j & a_j c_j \\ a_j b_j & b_j^2 & b_j c_j \\ a_j c_j & b_j c_j & c_j^2 \end{pmatrix} \\
&= \begin{pmatrix} 2a_j^2 & 2a_j b_j & 2a_j c_j \\ 2a_j b_j & 2b_j^2 & 2b_j c_j \\ 2a_j c_j & 2b_j c_j & 2c_j^2 \end{pmatrix} \\
&= 2 \begin{pmatrix} a_j^2 & a_j b_j & a_j c_j \\ a_j b_j & b_j^2 & b_j c_j \\ a_j c_j & b_j c_j & c_j^2 \end{pmatrix} \\
&= 2(\boldsymbol{m}_j\, \boldsymbol{m}_j^\top)
\end{aligned}$$

This type of calculation is applicable when we calculate the outer product of the vector of Score Equations, using *unique combination of observed values*. We multiply each outer product by the number of unique combinations we observe.

Appendix 3.E Monte Carlo Integral

Using the general notation from Robert and Casella (2010, p.65), we'll go through a brief overview of Monte Carlo Integration. Suppose the conditional expectation we wish to evaluate is

$$\mathrm{E}_f[h(X)] = \int_{\chi} h(x)f(x)\,\mathrm{d}x\,.$$

Where χ is the values the random variable X can take, which is usually the support of $f(\cdot)$. We approximate this conditional expectation by first generating m samples, $(X_1, X_2, \ldots, X_m)$ from $f(\cdot)$. We then substitute these values into $h(\cdot)$, then take an average of the sum. That is, the approximation is

$$\bar{h}_m = \frac{1}{m}\sum_{j=1}^{m} h(X_j)\,.$$

In our case, $f(\cdot) = f(x|\text{observed})$ and $h(\cdot)$ is the GLM Score Equations, i.e., $h(\cdot) = \mathbb{S}_i(\cdot)$ However, generating samples from $f(x|\text{observed})$ is not simple. One method to generate these sample is via methods such as *rejection sampling* (Robert and Casella, 2010, pp.52-57), (Rubinstein and Kroese, 2017, pp.59-71), and *importance sampling* (Robert and Casella, 2010, p.69). We used importance sampling technique because rejecting sampling was inefficient in sampling from low probability area of $f(x|\text{observed})$.

3.E.1 Incorporating Importance Sampling

With importance sampling, instead of generating samples from $f(\cdot)$, from Equation 3.E, we change the expectation itself as shown

$$\begin{aligned}\mathrm{E}_f[h(X)] &= \int_{\chi} h(x)\frac{f(x)}{g(x)}g(x)\mathrm{d}x \\ &= \mathrm{E}_g\left[h(x)\frac{f(x)}{g(x)}\right] \quad .\end{aligned}$$

In this re-formulation, $g(\cdot)$ is known as the *importance function* and it's a distribution we can easily sample from (e.g., Uniform). χ is still the support of the random variable X; it is *smaller* than the support of $g(\cdot)$ (Robert and Casella, 2010, p.69). In importance sampling, we sample from $g(\cdot)$ to obtain where $(X_1, X_2, \ldots, X_m)$, as opposed to $f(\cdot)$. Theses samples are called *importance samples* in this context. The approximation then is

$$\frac{1}{m}\sum_{k=1}^{m} h(X_k)\,\frac{f(X_k)}{g(X_k)} \to \mathrm{E}_f[h(X)]\,.$$

The fraction $f(\cdot)\big/ g(\cdot)$ is also known as weights. As it is shown in Equation 3.E.1, these weights are not normalised; they may not add up to 1. As a result, we'll normalise the weights as follows

$$w_j = \Big[\frac{f(X_k)}{g(X_k)}\Big]\Big/\Big[\sum_{k=1}^{m}\frac{f(X_k)}{g(X_k)}\Big].$$

Relating importance sampling back to the integral in Equation 3.7 that we want to approximate, the $h(\cdot)$ function is our GLM Score Equations $\mathbb{S}_i(\cdot)$ and $f(\cdot) \propto f(x|\text{observed})$. The approximation using importance sampling becomes

$$\frac{1}{m}\sum_{k=1}^{m}\mathbb{S}_j(X_k)\,\frac{f(x = X_j|\text{observed})}{g(X_k)}\ .$$

If we normalise the weights, $\big[f(x = X_k|\text{observed})\big]\,/\,\big[g(X_k)\big]$, the approximation is then

$$\sum_{k=1}^{m}\mathbb{S}_j(X_k)\,w_k\ .$$

Equation 3.E.1 can be recognised in the form of weighted GLM. This means for our M-Step, we can just run a GLM as opposed to specifying other equations to maximise.

Appendix 3.F Implementation Notes - Estimation Algorithm

The previous sections described the various parts of the algorithms. Now we'll put it all together. The algorithm to estimate the parameters using our work is shown in Algorithm 1.

Algorithm 1 Algorithm to estimate the parameters

Require: $N = N_s + N_r$
  $\mathbb{D}_s \leftarrow N_s$ samples from $\mathbb{D}$
  $\mathbb{D}_r \leftarrow N_r$ observations not in $\mathbb{D}_s$ {has N_s observations}
  $\boldsymbol{s}_1(\cdot) \leftarrow$ GLM *Score Equations* for $f(y|x, \boldsymbol{z}; \boldsymbol{\beta})$
  $\boldsymbol{s}_2(\cdot) \leftarrow$ GLM *Score Equations* for $f(x|\boldsymbol{z}; \boldsymbol{\alpha})$
  $\hat{\boldsymbol{\beta}}_s \leftarrow \boldsymbol{s}_1(\mathbb{D}_s)$
  $\hat{\boldsymbol{\alpha}}_s \leftarrow \boldsymbol{s}_2(\mathbb{D}_s)$
  $\mathbb{D}_r{}^* \leftarrow a[c(x, \mathbb{D}_r)]$ {coarsen and aggregate $\mathbb{D}_r$}
Require: Specification of $f(x|\vec{z})$ also is $f(x|\text{observed})$
Require: $m \leftarrow$ number of Monte Carlo samples
Require: Initialise the augmented data to be used in EM algorithm, $\mathbb{D}_{\text{EM}}$
Require: $t \leftarrow$ tolerance level for convergence
  $\hat{\boldsymbol{\beta}}^0 \leftarrow \hat{\boldsymbol{\beta}}_s$ {Initialise parameter for EM}
  $\hat{\boldsymbol{\alpha}}^0 \leftarrow \hat{\boldsymbol{\alpha}}_s$ {Initialise parameter for EM}
  $J \leftarrow$ all unique combinations of $(y, z^*, \boldsymbol{z})$
  while $(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\alpha}})$ not converged **do**
    for all j in J **do**
      $\boldsymbol{g}_j \leftarrow$ samples from Uniform distribution within ranges of x_j^* {Obtain m Importance Samples}
      $\boldsymbol{w}_j \leftarrow \frac{f(x|\text{observed})}{\boldsymbol{g}_j}$
      $\boldsymbol{W}_j \leftarrow$ normalise $\boldsymbol{w}_j$
      Append $\boldsymbol{g}_j$ and $\boldsymbol{W}_j$ to $\mathbb{D}_{\text{EM}}$
    end for
    Transform $\mathbb{D}_s$ into structure of $\mathbb{D}_{\text{EM}}$
    Append transformed $\mathbb{D}_s$ to $\mathbb{D}_{\text{prev}}$
    $\hat{\boldsymbol{\beta}}^{\text{new}} \leftarrow \boldsymbol{s}_1(\mathbb{D}_{\text{EM}})$
    $\hat{\boldsymbol{\alpha}}^{\text{new}} \leftarrow \boldsymbol{s}_2(\mathbb{D}_{\text{EM}})$
    $\hat{\boldsymbol{\beta}}^{\text{diff}} \leftarrow \text{abs}[\hat{\boldsymbol{\beta}}^{new} - \hat{\boldsymbol{\beta}}^{\text{prev}}]$
    $\hat{\boldsymbol{\alpha}}^{\text{diff}} \leftarrow \text{abs}[\hat{\boldsymbol{\alpha}}^{new} - \hat{\boldsymbol{\alpha}}^{\text{prev}}]$
    if $\hat{\boldsymbol{\beta}}^{\text{diff}} \leq t$ **and** $\hat{\boldsymbol{\alpha}}^{\text{diff}} \leq t$ **then**
      $(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\alpha}})$ converged
    end if
  end while

4 Meta-analysis on studies with heterogeneous covariates using propensity scores [9]

4.1 Introduction

Meta-analysis is the practice of combining results from many individual studies to form an overall result (McKenzie, Beller, and Forbes, 2016). It is widely used in many fields of research (Sutton and Higgins, 2008), ranging from public health (Ryan, 2008) to field trials in agricultural settings (Damesa et al., 2017). In this sense, meta-analysis can be thought of as a generalisation of the divide and recombine concept discussed in Section 2. In meta-analysis, the results from individual studies and their corresponding uncertainties are calculated for each study, then combined together to form an estimate of an 'overall effect' and its corresponding uncertainty. Theses results from individual studies are often estimates of an underlying effect. In one type of meta-analysis, we obtain some form of statistics and their uncertainties from individual studies, and our task is to combine them. In another type of analysis, called *individual participant data*(IPD) meta-analysis, we have all the data from all the studies. This means we not only have to calculate the results for individual studies, we also have to combine them. We can relate meta-analysis to distributed data if we treat distributed data as individual studies in the context of meta-analysis. By treating distributed data this way, we can apply meta-analysis techniques to distributed data. As we'll describe later, the Divide and Recombine paradigm used to perform statistical analysis on distributed data, is equivalent to one type of meta-analysis (fixed effect meta-analysis).

Since meta-analysis combines data from different studies, a characteristic to taken into account is the variation across the individual studies known as study to study *heterogeneity*. The sources of heterogeneity include different study population, outcome variable, and covariate designs (Liu, Liu, and Xie, 2015; Sutton and Higgins, 2008). Heterogeneity is of great interest because the the differences between the studies can impact the interpretation of the synthesised result from meta-analysis. McKenzie, Beller, and Forbes (2016) note that in the presence of substantial heterogeneity, synthesised meta-analysis result such as the average effect may become at best uninformative.

One particular cause of between study to study heterogeneity is differing covariate structure. Although the studies in a meta-analysis may look at the effect of a variable of interest on the same outcome variable, each study may have different design goals in mind. As a result, these that studies have different covariates (Quartagno and Carpenter, 2016). If we are

[9] This research used data contributed and maintained by team members of HBGDki project, which is funded by Bill and Melinda Gates foundation.

synthesising regression coefficients in a IPD setting, the differing covariate structure means we have different regression models across the studies, as shown in Table 4.1. One method of dealing with this problem would be to manually perform variable selection across all the studies so all the studies have the set of relevant variables. Another technique we can adopt for IPD meta-analysis is to impute missing covariates so that all the studies have the same covariates (Quartagno and Carpenter, 2016). In practice, such an approach can be very cumbersome and impractical.

In this work, we propose using propensity scores as a means to adjust for different covariates across the studies of a meta-analysis in an IPD setting. In particular, using the propensity scores to perform covariate adjustment, to control for covariate effects in regression models (Elze et al., 2017). Covariate adjustment using propensity score summaries all the characteristics of individual subjects into a single covariate (Elze et al., 2017), which we then use in our regression model involving the outcome variable. By using propensity scoring, we can obtain equivalent regression models across all the studies in a meta-analysis.

An additional issue we need to address is missing data. The missingness in this context is about observations not having recorded values for variables, as opposed to studies missing the covariates directly as in the case above. Since our studies have many covariates, we may not have enough observations in some of the studies to estimate the propensity scores using a complete case analysis. As a result, before we calculate our propensity scores, we also perform imputation using MICE (van Buuren and Groothuis-Oudshoorn, 2011). Addressing missing data issue using MICE is further explored in a later section.

The structure of this chapter is as follows. Section 4.2.1 provides an overview of the techniques used in this chapter such as meta-analysis and propensity scores. This is followed by Section 4.3 where we show the application of our methods to child growth data from the Healthy Birth Growth & Development knowledge integration (HBGDki) project sponsored by the Bill and Melinda Gates Foundation. We then discuss the result of the analysis in Section 4.4, follow by a discussion of future research in Section 4.6.

4.2 Methods used in this chapter

4.2.1 Meta-analysis

Sutton and Higgins (2008) describe meta-analysis as a 'two-stage' approach, where analyses are performed on individual studies, then their results are combined as a weighted average. Suppose we have $k = 1, 2, \ldots, \kappa$ independent studies. In the first stage, we perform an analysis on each of the separate studies such that we have $\{\theta_1, \theta_2, \ldots, \theta_\kappa\}$ results. In the second

stage, we combine our results to obtain an overall estimate $\widehat{\theta}$, which is

$$\widehat{\theta} = \frac{\sum w_k \theta_k}{\sum w_k} ,$$

where w_k is the weight for kth study. The weights are calculated under two different assumptions according to two types of meta-analysis models, *fixed effect* and *random effect* meta-analysis (McKenzie, Beller, and Forbes, 2016; Normand, 1999; Sutton and Higgins, 2008).

In the fixed effect model, the result from each study is assumed to be distributed around a common result. That is, there is a common result that is shared by all the studies (Normand, 1999). Denoting θ as the common result, estimate from the kth study can be characterised as

$$y_k = \theta + \epsilon_k ,$$

where ϵ_k is the estimation error of the estimator obtained from the kth study, which is the standard error of the estimate, $s_k = \text{SE}(y_k)$ (McKenzie, Beller, and Forbes, 2016). We can further assume that estimates from each study y_k are realisations from a Gaussian distributed population of estimates with the common result θ as the mean (Normand, 1999), such that

$$y_k \sim \mathcal{N}(\theta, \epsilon_k) .$$

The weights assigned to each study w_k is the inverse of the *variance* of the estimate within each study, i.e., $w_k = 1/[\text{SE}(y_k)]^2$ (McKenzie, Beller, and Forbes, 2016). In the fixed-effect model, only the variation of the result from each study, from the common result is modelled. That is, the study to study variation is not modelled. As well, statistical inference using the fixed effect model then restricted to the set of κ studies included in the analysis (Viechtbauer, 2010). Note that the way divide and recombine technique calculates the common result from distributed data is the same as the fixed effect model.

To account for study to study heterogeneity, we use the random effect meta-analysis model, which models the study specific estimates and the overall result as distributions. The study specific estimate is modelled as a sample from a study specific distribution with mean u. That is, for the kth study, the study specific estimate is

$$y_k = u_k + \epsilon_k ,$$

under a Gaussian distribution with mean u_k,

$$y_k \sim \mathcal{N}(u_k, \epsilon_k^2) .$$

The interpretation here is that for kth study, there is a 'true' effect u_k, which is estimated y_k, with some amount of error ϵ_k. In random effects meta-analysis model, the 'true' effects within individual studies are also considered as samples from a Gaussian distribution with the common result θ as the mean, and with variance τ^2

$$u_k \sim \mathcal{N}(\theta, \tau^2)\,.$$

The inference from random effects model is different from fixed effect model because the studies are considered to be samples from a hypothetical population of studies that can be conducted (Viechtbauer, 2010), as opposed to being limited to the number of studies as in the fixed effect model. Since there are distributions at two levels, we now have two sources of variations, the study to study variation (τ^2), and the within study variation $[\mathrm{SE}(y_k)]^2$. The weights assigned to each study during synthesis becomes (McKenzie, Beller, and Forbes, 2016)

$$w_k = \frac{1}{(\tau^2 + [\mathrm{SE}(y_k)]^2)}\,.$$

See Normand (1999, Figure 3 and Figure 4) and McKenzie, Beller, and Forbes (2016, Figure 1) for illustrations that show these two different models. The incorporation of study to study heterogeneity allows the the random effect model to take into account different study design characteristics. This makes the random effects meta-analysis model to be suitable for use with distributed data, since the units of distributed data can have different characteristics. As a result, in this chapter, we use the random effects meta-analysis.

Note both the fixed effect and random effect meta-analysis assume normality in their models. Jackson and White (2018) discuss the implication and limitation of this normality assumption. In cases where this assumption does not hold, alternative methods such as Bayesian models can be used. Lee and Thompson (2008) demonstrate the use of Bayesian hierarchy model to perform random effect meta-analysis on a dataset where the results from some of the studies are outliers. Instead of modelling the random effects as Gaussian distributed, Lee and Thompson (2008) model the random effects with different distributional assumptions such as t-distribution, skewed-normal and skewed-t.

4.2.2 Propensity scoring

Randomised controlled trials (RCT) in medical setting often are designed to determine the effect of a particular 'treatment'. In many of these trials, the 'treatment' is binary in nature. The comparison of interest is between subjects in 'treated' and 'un-treated' group, or between 'treated' and 'control' group. The randomisation of the assignment 'treatment' to subjects

allows an unbiased estimate of the average treatment effect to be calculated. In contrast to randomised controlled trials, there is no guarantee of randomised treatment assignment to subjects in observational studies. As a result, there can be systematic difference of the covariates between the 'treated' and 'un-treated' (or 'control') group (d'Agostino, 1998). Due to this systematic difference, when we calculate the average treatment effect, we'll have bias in our estimates (Austin, 2011; Gelman and Hill, 2006; d'Agostino, 1998). The technique of using propensity score, developed by Rosenbaum and Rubin (1983) is a way to address the challenge of performing statistical inference using observational studies.

In the context of binary 'treatment', the propensity score is the *conditional probability* of a subject being 'treated' (or 'exposed') given the subject's covariates (d'Agostino, 1998). Specifically, those covariates that are *pre-treatment* Gelman and Hill (2006). Rosenbaum and Rubin (1983) showed that if the covariates of the subjects are sufficient to control for confounding, then adjusting for the propensity score is also sufficient (Vansteelandt and Daniel, 2014). Propensity scoring methods allow us to mimic covariate balance of randomised controlled trials through the propensity score. Subjects with similar propensity score should *on average* have the same distribution of the covariates, regardless of their assignments to the 'treatment' or 'non-treatment' ('control') group (Elze et al., 2017). Traditionally, propensity score methods use a binary treatment variable. However, recent works such as Imai and Van Dyk (2004) and Yang et al. (2016) generalise the propensity score methods to allow for multiple levels, and continuous propensity scores.

The propensity score is a scalar value for each individual subject. To estimate the propensity score, we regress the 'treatment' variable on the pre-treatment covariates of the subjects. For the jth subject, the *propensity score model* (Austin, 2011) is then

$$z_j = \delta_0 + \delta_1 x_{1j} + \delta_2 x_{2j} + \ldots + \delta_p x_{pj} + \varepsilon_j \,, \tag{4.1}$$

where z_j is the 'treatment', $\varepsilon_j \sim \mathrm{N}(0, \sigma_\rho^2)$ and $x_1, x_2, \ldots, x_p$ are the pre-treatment covariates. For a binary 'treatment' (i.e., $z_j \in \{0, 1\}$) (4.1) can be fitted using logistic regression. With regards to pre-treatment covariates to include in (4.1), Elze et al. (2017) recommends including all the pre-treatment covariates that are related to the outcome and/or 'treatment' variable, but exclude those that are exclusively related to 'treatment'. The *predicted values* from (4.1) then become the estimated propensity scores ρ_j, i.e., $\rho_j = \widehat{z}_j$.

Some popular ways of using propensity scores are *propensity score matching*, *inverse probability of treatment weighting* and covariate adjustment using propensity score where the estimated propensity scores as a covariate in the outcome model (Austin, 2011; Williamson and Forbes, 2014; Williamson et al., 2012).

Propensity score matching follows the definition of the propensity score closely, whereby subjects with similar propensity scores have similar distribution of covariates. Using this definition, subjects with similar propensity scores in the 'treatment' and non-treatment (or 'control') group (assuming a binary 'treatment') are *matched*. Another way of thinking about matching is that propensity score matching discards observations so that the remaining subjects have similar covariate distribution across the 'treatment' and non-treatment groups (Gelman and Hill, 2006). Statistical analysis using these matched subjects mimic analysis that would have been performed using subjects from a randomised controlled trial.

In the inverse probability of treatment weighting method, subjects in the original dataset are weighted by the inverse of their propensity scores to create a pseudo-population, in which the subjects have similar distribution of covariates (Austin, 2011; Pearl, Glymour, and Jewell, 2016). The effect of 'treatment' is then estimated directly from this pseudo-population.

Finally, propensity scores can be used as a form of covariate adjustment, with the outcome variable regress on the 'treatment' and the propensity score. The *response model* for jth subject is

$$y_j = \alpha_0 + \alpha_1 z_j + \alpha_2 \rho_j + \varepsilon_j \tag{4.2}$$

where y is the outcome variable, z is the 'treatment' variable, and ρ is the estimated propensity score, and $\varepsilon \sim \mathrm{N}(0, \sigma^2)$ is the error term. Using propensity score this way allows the investigator of a dataset to control for many covariates as possible, which allows for the possibility that any of them are confounding covariates (Gelman and Hill, 2006). This is in contrast to to the traditional method of covariate adjustment, where a multivariate regression the outcome variable is regressed on all relevant covariates are fitted (Elze et al., 2017; Williamson and Forbes, 2014), where the relevant covariates are selected with methods such as forward selection, or comparing some measure of the fitted model such as the AIC.

One advantage of using propensity score in covariate adjustment compared to traditional multivariate regression is that since the propensity score is scalar, it allows us to use a simpler model for confounding adjustment that might not be possible with high-dimensional confounders (Vansteelandt and Daniel, 2014). This advantage becomes clear when covariate adjustment needs to be performed for multiple studies. For instance, suppose we have κ studies with different covariate structures as shown in Table 4.1 (the model structure is in R formula format to emphasis the covariates in each model). Due to different covariate structure within each study, the *response model* (*outcome model*) from individual study are different. As such, the studies can be considered as having different models. One way to create a common model across all the studies is through the use of propensity scores. By performing covariate adjustment using the propensity score model shown in (4.1), we can create the

same response model across all the studies as shown in Table 4.2.

There are other methods for combing regression slopes without using propensity scoring. Liu, Liu, and Xie (2015) combine *confidence density functions* instead of summary statistics from each study in a meta-analysis. Confidence densities are sample dependent distributions that provide all aspects of statistical inference for a parameter of interest (Liu, Liu, and Xie, 2015, p.328). Since they contain more information than just a summary measure, confidence density functions can incorporate the variations between studies into meta-analysis. Another approach taken by Becker and Wu (2007) involves using generalised least squares methods to combine the regression slopes across studies, provided the variance-covariance matrix of the regression slopes of the individual studies are available. These methods however, require extra calculation such as density of confidence intervals, which can add additional complexity to the meta-analysis process.

		Covariates						Model structure
Study	Y	Z	X1	X2	X3	...	X_p	
S_1	✓	✓	✓	✓	✓	...		$Y = Z + X1 + X2 + X3$
S_2	✓	✓	✓		✓	...	✓	$Y = Z + X1 + X3 + X_p$
S_3	✓	✓		✓		...		$Y = Z + X2$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
S_K	✓	✓	✓	✓				$Y = Z + X1 + X2$

Table 4.1: *An illustration of varying covariate structures across studies. Y is the outcome variable, Z is the covariate of interest, and $X_1, X_2, \ldots, X_p$ are other covariates. In order to emphasis the covariates in each model, the model structure is shown as R formula format. A checkmark ✓ indicates the presence of the covariate in the study (i.e., not 100% missing). The model structure shows the dependency of the outcome variable on the other covariates. Different studies can have different covariate structure due to different study designs.*

4.2.3 Imputation via MICE

Before the estimation of propensity scores the issue of missing data needs be addressed. Missing data in this context refer to missing values of covariates, which is also known as *item non-response* (Lee and Simpson, 2013). We can always perform a *complete case analysis* by ignoring subjects with item non-responses. However, removing too many subjects with missing data can introduce bias to our estimates of regression coefficients since the model is fitted using those subjects with all the covariate measurements (Lee and Simpson, 2013). Another possible strategy of dealing with missing data is to use likelihood based methods that allows for missing data such as EM algorithm (van Buuren and Groothuis-Oudshoorn, 2011). However, such approach is sensitive to model mis-specification. A more robust approach we take

			Other covariates					Model structure
Study	Y	Z	X1	X2	…	X_p	ρ	
S_1	Y_1	Z_1	✓	✓	…		✓	$Y_1 = Z_1 + \rho_1$
S_2	Y_2	Z_2	✓		…	✓	✓	$Y_2 = Z_2 + \rho_2$
S_3	Y_3	Z_3		✓	…		✓	$Y_3 = Z_3 + \rho_3$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
S_K	Y_K	Z_K	✓	✓			✓	$Y_K = Z_K + \rho_K$

Table 4.2: *An illustration of using propensity scoring as covariate adjustment across many studies. Y is the outcome variable, Z is the 'treatment' variable (or variable of interest), and $X_1, X_2, \ldots, X_p$ are other covariates. ρ is the estimated propensity score for each study. In order to emphasis the covariates in each model, the model structure is shown as R formula format. The propensity model allows controlling of many covariates in case one of them is a confounder variable. Using propensity score this way means the model of interest for each study is the same.*

is multiple imputation approach by Rubin (1987).

In general, there are two approaches to imputing multivariate data, joint modelling (JM) and fully conditional specification (FCS) (Van Buuren et al., 2006). In joint modelling, missing values are drawn from a parametric multivariate density that relates the data and all parameters (Van Buuren et al., 2006). The joint modelling approach can be viewed as imputing all the missing values simultaneously (Lee and Simpson, 2013). Schafer (1997) outlines the different multivariate distributions that can be used in joint modelling technique. Multivariate imputation using joint modelling however, is not flexible in the sense it may be difficult to perform imputation with non-standard multivariate densities.

In the fully conditional specification (FCS) method, instead of specifying a density incorporating all the covariates, density models for the individual covariates with missing values, conditioned on the observed covariates (including the outcome variable) are generated. One of the most commonly used implementation of fully conditional specification method is the multi-variate imputation by chained equations (MICE) (van Buuren and Groothuis-Oudshoorn, 2011). Due to its flexibility and software availability, we use MICE to impute missing values.

In multiple imputation techniques, plausible values for the missing data are sampled from predictive distributions (Mitra and Reiter, 2016). In MICE, the plausible values for missing values are modelled through iterative regression models and Gibbs sampling of the conditional densities of these regression models. The result of multiple imputation is a set of complete data that can then be used in analysis. These presence of multiple complete datasets require additional consideration in in our modelling process.

In the simplest case of fitting a regression model using dataset missing values, as out-

lined in van Buuren and Groothuis-Oudshoorn (2011, Section 2.2), we first impute the missing values using technique such as MICE resulting in a number of imputed datasets (complete datasets). We then fit our regression model using each of the imputed datasets. We will have multiple regression coefficients for each of the parameters in the model from multiple datasets. We then *pool* these regression coefficients using Rubin's Rule (McKenzie, Beller, and Forbes, 2016). To clarify, suppose we have data from a single study s_1 with missing data. Assume MICE will generates five imputed datasets $\{s_1^{(1)}, s_1^{(2)}, s_1^{(3)}, s_1^{(4)}, s_1^{(5)}\}$, where the superscript indicates the imputation number. Suppose we fit the regression model in (4.2) to each of the imputed data set, and our coefficient of interest is α_1. Since there are five imputed datasets, we will have five estimates $\{\hat{\alpha}_1^{(1)}, \hat{\alpha}_1^{(2)}, \hat{\alpha}_1^{(3)}, \hat{\alpha}_1^{(4)}, \hat{\alpha}_1^{(5)}\}$. These estimates, and their standard errors are then pooled using Rubin's Rule to obtain an overall estimate and its corresponding uncertainty.

Our work differs from the analysis flow described in previous paragraph in two important ways. Firstly, we have multiple datasets from different studies in HBGDki, which means after multiple imputation, we will have multiple imputed datasets *per study*. Secondly, our goal is to perform a meta-analysis of the regression coefficient for growth velocity, across all the studies. Since we have multiple imputed datasets per study we have an additional complication to consider with regards to the order of analysis step.

There are two different paths to performing meta-analysis using imputed datasets from multiple studies, as shown in Burgess et al. (2013, Figure 1). One option is to apply the conventional method as described above *per study*, i.e., for each study fit regression models on the imputed datasets, then pool the estimates. We then meta-analyse the pooled regression coefficients across all the studies. Quartagno and Carpenter (2016) call this approach 'RR then MA' (Rubin's Rule, then meta-analysis). Using the notation from previous paragraph suppose we have two studies $\{s_1, s_2\}$ with missing values. Assuming MICE generates five imputed sets, we have $2{\times}5 = 10$ total number of imputed datasets $\{s_1^{(1)}, s_1^{(2)}, s_1^{(3)}, s_1^{(4)}, s_1^{(5)}, s_2^{(1)}, s_2^{(2)}, s_2^{(3)}, s_2^{(4)}, s_2^{(5)}\}$. We then fit our regression model on these imputed datasets, with α as our coefficient of interest, we have $\{\hat{\alpha}_1^{(1)}, \hat{\alpha}_1^{(2)}, \hat{\alpha}_1^{(3)}, \hat{\alpha}_1^{(4)}, \hat{\alpha}_1^{(5)}, \hat{\alpha}_2^{(1)}, \hat{\alpha}_2^{(2)}, \hat{\alpha}_2^{(3)}, \hat{\alpha}_2^{(4)}, \hat{\alpha}_2^{(5)}\}$. In 'RR then MA' approach, the next step is to pool these estimates *within each study*. Denoting $\hat{\alpha}_k^*$ as the pooled estimate of α for the kth study, with our example, pooling our estimates will result in $\{\hat{\alpha}_1^*, \hat{\alpha}_2^*\}$. We then perform meta-analysis over these pooled estimates.

An alternative approach to 'RR then MA' is 'MA then RR', meta-analysis followed by Rubin's Rule(Quartagno and Carpenter, 2016). In this alternative approach, we fit regression models on the imputed datasets as before, but instead of pooling the estimates, we first meta-analysis on the coefficients across imputation order, then pool the combined estimates. To illustrate this alternative approach, using the same notation as above with two datasets $\{s_1, s_2\}$

with missing data. Assuming MICE generates five imputed datasets per study so that we have $\{s_1^{(1)}, s_1^{(2)}, s_1^{(3)}, s_1^{(4)}, s_1^{(5)}, s_2^{(1)}, s_2^{(2)}, s_2^{(3)}, s_2^{(4)}, s_2^{(5)}\}$ imputed data sets. The major difference from previous method is that instead of performing analysis on each study, we perform it on same imputed dataset across studies. As in the previous method, assuming α as our coefficient of interest, when we apply regression to these imputed datasets, we have $\{\hat{\alpha}_1^{(1)}, \hat{\alpha}_1^{(2)}, \hat{\alpha}_1^{(3)}, \hat{\alpha}_1^{(4)}, \hat{\alpha}_1^{(5)}, \hat{\alpha}_2^{(1)}, \hat{\alpha}_2^{(2)}, \hat{\alpha}_2^{(3)}, \hat{\alpha}_2^{(4)}, \hat{\alpha}_2^{(5)}\}$. Instead of pooling these estimates within each study, we meta-analyse them according to their imputation order. That is, we first group these estimates according to their imputation number so that we have $\{\hat{\alpha}_1^{(1)}, \hat{\alpha}_2^{(1)}\}$, $\{\hat{\alpha}_1^{(2)}, \hat{\alpha}_2^{(2)}\}$, $\{\hat{\alpha}_1^{(3)}, \hat{\alpha}_2^{(3)}\}$, $\{\hat{\alpha}_1^{(4)}, \hat{\alpha}_2^{(4)}\}$, $\{\hat{\alpha}_1^{(5)}, \hat{\alpha}_2^{(5)}\}$. Performing meta-analysis on each group of estimate, we have the following combined estimates $\{\hat{\alpha}^{(1)}, \hat{\alpha}^{(2)}, \hat{\alpha}^{(3)}, \hat{\alpha}^{(4)}, \hat{\alpha}^{(5)}\}$. We then pool these combined estimates using Rubin's Rule.

Burgess et al. (2013) performed a simulation study to examine the characteristics of combining imputation and individual participant data meta-analysis. They recommend the 'RR then MA' approach. The main reason is that the multiple imputation process generates additional heterogeneity, which can add to the heterogeneity of the meta-analysis. Therefore, to separate the study to study heterogeneity from heterogeneity due to multiple imputation, analysis using the imputed datasets should first be combined with Rubin's Rule.

4.3 Application to child growth data from HBGDki

The Healthy Birth, Growth and Development knowledge integration (HBGDki) initiative, sponsored by the Bill and Melinda Gates Foundation, aims to develop better interventions for children with faltering growth (Jumbe, Murray, and Kern, 2016). As part of achieving this goal, HBGDki has collated child growth data from over 100 studies (Anderson et al., 2018)with varying characteristics. These studies can be longitudinal or cross-sectional, have different question of interest, and the subject cohorts are from different countries. Anderson et al. (2018) describe 21 of these studies with longitudinal measures as having 800,000 observations recorded for over 100,000 subjects. Using the data from HBGDki, our question of interest is the overall effect of *average physical growth velocity* of the subjects with their first year, on their cognitive growths. To explore this question, we apply the methods described in Section 4.2.1 to a set of longitudinal studies from HBGDki. By combining propensity scoring with random effect meta-analysis, we address the challenge of performing data analysis on data with varying characteristics.

We are interested in the physical growth of children during their first year because linear growth during these initial periods impacts child development later on life (Kowalski et al., 2018), such as cognition. One measure of physical growth is the height (length) of the chil-

dren. Instead of directly using physical measurement in our model, we use their statistical standardised equivalents. For height, this means using height for age z-score (HAZ) from the World Health Organisation (WHO) standardised growth charts (Ebrahim, 2010; Onis et al., 2006). The WHO standardised growth charts allow physical growth of a child to be quantified at a particular age, and allow comparison with the population distribution of children at the same age and sex (Anderson et al., 2018).

There are many types of models that can be used to model child growth, such as polynomial, penalised splines and also techniques such as Super Imposition by Translation And Rotation (SITAR) by Cole, Donaldson, and Ben-Shlomo (2010) (Anderson et al., 2018). In this work, we model the physical child growth using the *broken stick* model, which is a piecewise linear spline model (Fitzmaurice, Laird, and Ware, 2011). We use the brokenstick model due to its ease of construction and interpretability. In addition, when Anderson et al. (2018) compared the predictive abilities of various growth modelling techniques on longitudinal child growth data, the brokenstick model has a high accuracy rate.

To model height for age z-score (HAZ), we construct our brokenstick using first order B-Spline basis functions placed at five evenly spaced *knots* across the first year of a child's age. We place two of these knots at the birth and at one year of age. In this setup, these two knots at the beginning and the end of the age of interest are called *external knots*. The three remaining knots, are placed at three, six and nine months of a child's age. In our setup, these three knots are called *internal knots*. We are aware that there are other methods of knot placements, such as using quantiles of the data (Ruppert, Wand, and Carroll, 2003b, Section 5.5.3). We chose this particular knot placement strategy due to its interpretability as well as ease of calculating the *growth velocity*.

Since the growth pattern can vary from children to children, we are particularly interested in *growth velocity*. Growth velocity measurement are particularly useful for modelling 'stunted' growth and potential subsequent recovery growth. In this work, the growth velocity measure we use is the average of the first year growth velocity, which is a single numerical value for each children in a study.

Once we have constructed growth trajectories for all the subjects in each study, we then calculate the *average growth velocity* across the first year of age for those subjects with cognitive outcome measurements. Since we placed the knots uniformly every three months, the predicted average growth velocity over one year simply becomes the change in predicted growth over one year. For a child, assuming $\widehat{h}_i$ to be the value of HAZ at ith measurement time, t_i as the month where measurement was taken, with $i \in \{1, 3, 6, 9, 12\}$ months, the av-

erage first year growth velocity can be calculated as

$$\frac{1}{4} \times \left[\frac{(\widehat{h}_3 - \widehat{h}_1)}{(t_3 - t_1)} + \frac{(\widehat{h}_6 - \widehat{h}_3)}{(t_6 - t_3)} + \frac{(\widehat{h}_9 - \widehat{h}_6)}{(t_9 - t_6)} + \frac{(\widehat{h}_{12} - \widehat{h}_9)}{(t_{12} - t_9)} \right] \quad . \tag{4.3}$$

Since our knots are placed every three months, they are equi-distance. That is

$$(t_3 - t_1) = (t_6 - t_3) = (t_9 - t_6) = (t_{12} - t_9) = 3 \, .$$

Substituting the constant month differences into (4.3), the average growth velocity over one year then simplifies to

$$\frac{1}{4} \times \frac{1}{3} \times \left[(\widehat{h}_3 - \widehat{h}_1) + (\widehat{h}_6 - \widehat{h}_3) + (\widehat{h}_9 - \widehat{h}_6) + (\widehat{h}_{12} - \widehat{h}_9) \right] = \frac{\widehat{h}_{12} - \widehat{h}_1}{12} \, ,$$

which is the change of HAZ over one year.

For the cognitive growth of the children, we use results from cognitive tests as the measurement. As with physical growth modelling, there are many types of tests that can be administered to measure cognitive growth. In this work, we prefer the *global* cognitive tests, which are well known battery of tests that can be tailored to different countries and cultures. Some example of these global cognitive tests are Bayley Scales of Infant Development (BSID), Wechsler Intelligence Scale for Children (WISC), and Wechsler Abbreviated Scale of Intelligence (WASI). In our analysis, we use the cognitive test results from either around two or seven years age. To examine the effect of average first year growth velocity on the cognitive outcome, we have the following regression model for the jth study

$$y_j = \beta_0 + \beta_1 z_j + \ldots + \beta_p x_p + \epsilon_j \, , \tag{4.4}$$

where y_j is the cognitive outcome, z_j is the average growth velocity in the first year, $x_2, x_3, \ldots, x_p$ are the subject specific covariates such as sex, and ϵ_j is the error term. The methods outlined in Section 4.2.1 are used to address various challenges when attempting the perform meta-analysis on the estimated coefficients for the average growth velocity in the first year in (4.4) i.e., $\hat{\beta}_1$. As described in Section 4.2.1, the goal is to have the same model across all the studies through the use of propensity score, while taking into account the multiple imputed datasets.

The covariates $x_2, x_3, \ldots, x_p$ in (4.4) are referred to as subject-specific covariates since they are constant throughout the time period in which data was collected. They include covariate specific to the subject such as sex, birth weight, as well as social-economic covariates such as parents education level and income. In many studies within HBGDki, these social-economic

covariates can be missing, in the sense of item non-response, i.e., values of the covariate not recorded for some subjects. We can ignore the subjects with missing values and use a complete case analysis, however since there are many social-economic covariates, we will be discarding a lot of subjects. As such, we will have an inefficient model when using complete case analysis. In addition, the model will have bias since the data contain only those subjects who have all the covariates.

In order to use as many subjects as possible in our analysis, we impute the missing values using MICE (van Buuren and Groothuis-Oudshoorn, 2011). Not all the missing data should be imputed however. If there are too many missing values, imputation introduce instability to the estimation process. In this work, we do not impute variables that have more than 50% of their values are missing. In addition, since the plausible values to impute that MICE generates depend on the observed values in the variable, we also do not impute those categorical variables which have multiple levels, but we only observe one level.

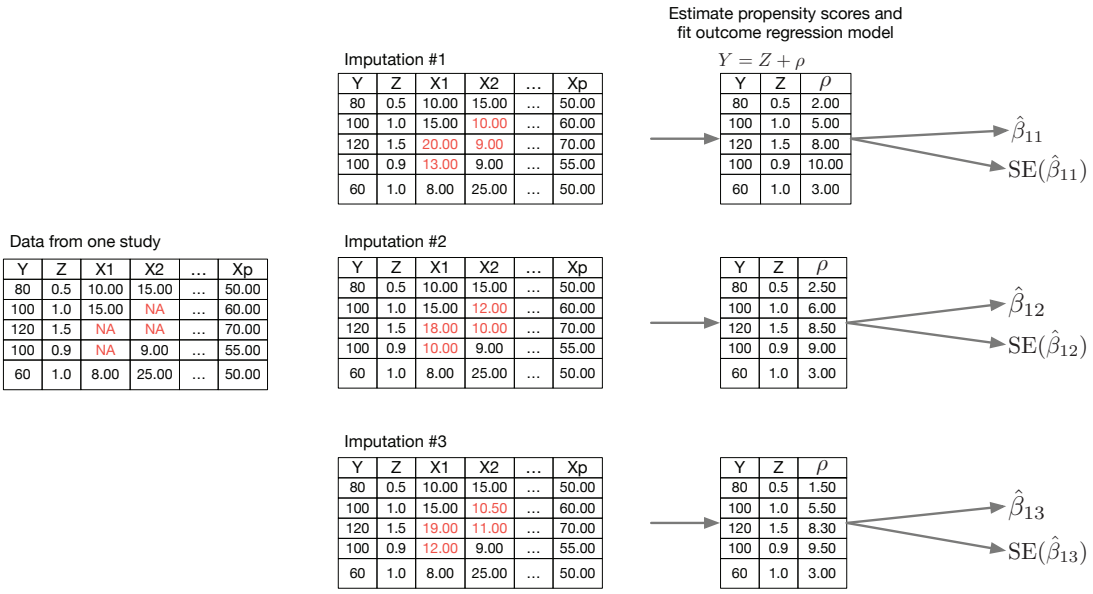


Figure 4.1: *In 'RR then MA' approach, the sequence of operation is to first impute using multiple imputation, follow by estimation of the propensity scores, and then fit the regression model of interest. These estimated regression coefficients $\{\hat{\beta}_{11}, \hat{\beta}_{12}, \hat{\beta}_{13}\}$ are then pooled. Each study now has a pooled estimate, which we use in our random-effect meta-analysis.*

In Section 4.2.1, we describe the two methods of analysis workflow to take into account multiple imputed datasets generated by MICE, namely 'RR then MA' which involves applying Rubin's Rule first, then perform meta-analysis, or its alternative 'MA then RR' which is the reverse. In this work we use the 'RR then MA' approach, as illustrated in Figure 4.1.

Following the 'RR then MA' approach, we first impute the missing values within each study. The imputation process will generate multiple imputed datasets. We then estimate the propensity scores using the propensity score model shown in (4.5), where our 'treatment' variable is the average of first year growth velocity of the subjects. Assuming we have κ studies indexed by $k \in \{1, 2, \ldots, \kappa\}$, and for each study there are M imputed datasets indexed by $m \in \{1, 2, \ldots, M\}$, with N number of subjects indexed by $j \in \{1, 2, \ldots, N\}$. Denoting z as the average first year growth velocity, and $x_1, x_2, \ldots, x_p$ as subject-specific covariates, the propensity score model for jth subject of mth imputed dataset within kth study is

$$z_{(jmk)} = \delta_{0(mk)} + \delta_{1(mk)}x_{1(jmk)} + \delta_{2(mk)}x_{2(jmk)} + \ldots + \delta_{p(mk)}x_{p(jmk)} + \epsilon_{(jmk)} \,, \tag{4.5}$$

where $\epsilon_{(jmk)} \sim \mathcal{N}(0, \sigma_z^2)$. The estimated propensity score, per imputed dataset, per study, denoted by $\rho_{(jmk)}$ is the *predicted* values from (4.5), i.e., $\rho_{(jmk)} = \widehat{z}_{(jmk)}$. We then fit our *outcome regression model* by regressing the cognitive measurement (y), on the estimated propensity score (ρ) and the average first year growth velocity (z). For the jth subject of mth imputed dataset within kth study, the outcome model is

$$y_{(jmk)} = \beta_{0(mk)} + \beta_{1(mk)}z_{(jmk)} + \beta_{2(mk)}\rho_{(jmk)} + \epsilon_{(jmk)} \,, \tag{4.6}$$

where $\epsilon_{(jmk)} \sim \mathcal{N}(0, \sigma_c^2)$. For M imputed datasets in kth study, we have $\{\hat{\beta}_{k1}, \hat{\beta}_{k1}, \ldots, \hat{\beta}_{kM}\}$ as the estimated regression coefficients for average first year growth velocity. We then pool these for each study using Rubin's rule (Rubin, 1987). The pooled estimate for kth study is

$$\beta_{k,\text{MI}} = \frac{1}{m}\sum_{m=1}^{M}\hat{\beta}_{(km)} \,.$$

To pool the standard errors, Rubin rule takes into account the variation *within* the imputed datasets as well as the variation across the imputed datasets. The pooled standard error for kth study is

$$\text{SE}(\beta_{k,\text{MI}}) = \sqrt{V_k + (1 + \frac{1}{M})B_k} \quad , \tag{4.7}$$

where V_k pools the variations within each of the imputed the dataset, calculated as

$$V_k = \frac{1}{M}\sum_{j=1}^{M}[\text{SE}(\hat{\beta}_{(km)})]^2 \,,$$

and B is the variation between the imputed data calculated as

$$B_k = \frac{1}{M-1} \sum_{j=1}^{M} \left(\hat{\beta}_{(km)} - \beta_{k,\mathrm{MI}} \right)^2 .$$

After the completion of the above process for each study, we apply random effect meta-analysis on the pooled regression estimates $\{\beta_{1,\mathrm{MI}}, \beta_{2,\mathrm{MI}}, \ldots, \beta_{k,\mathrm{MI}}\}$ and their corresponding standard errors across κ studies.

4.4 Results

Study	Cognitive measurement	Number of subjects	Age of cognitive measurement	Estimated coefficient ($\beta_{k,\mathrm{MI}}$)	Standard error (SE($\beta_{k,\mathrm{MI}}$))
cvb	General IQ	147	8 years ± 6 months	-0.0837	0.356
cph	Academic Attainment	2166	11 years	0.285	0.159
cph	General IQ	2252	8.5 to 9 years	0.792	0.323
cppbtn	WISC	8676	7 years	0.483	0.192
cpprmd	Stanford-Binet	1762	4 years	0.239	0.314
cpprmd	WISC	2719	7 years	0.491	0.246
cppbtm	Stanford-Binet	2551	4 years	0.802	0.321
cppbtm	WISC	2403	7 years	0.806	0.291
cppbfl	Stanford-Binet	1471	4 years	0.8	0.585
cppbfl	WISC	2077	7 years	0.916	0.433
cppmph	Stanford-Binet	1041	4 years	-0.378	0.51
cppmph	WISC	2815	7 years	-0.314	0.324
cppmnp	Stanford-Binet	1645	4 years	0.559	0.46
cppmnp	WISC	2300	7 years	0.239	0.359
cppnwo	Stanford-Binet	1388	4 years	3.3	1.26
cppnwo	WISC	2063	7 years	1.14	0.95
cppnwk	Stanford-Binet	3030	4 years	-0.505	0.395
cppnwk	WISC	2792	7 years	-0.0604	-1.61
cppphl	Stanford-Binet	4570	4 years	0.534	0.27
cppphl	WISC	7109	7 years	0.377	0.195
cpppld	Stanford-Binet	1713	4 years	0.948	0.357

Study	Cognitive measurement	Number of subjects	Age of cognitive measurement	Estimated coefficient ($\beta_{k,\text{MI}}$)	Standard error (SE($\beta_{k,\text{MI}}$))
cpppld	WISC	2407	7 years	0.769	0.277
cpppvd	Stanford-Binet	1665	4 years	1.32	0.44
cpppvd	WISC	2983	7 years	0.314	0.285
gto	Bayley Cognitive	151	2 years	1.473	1.781
jta	Bayley Cognitive	677	2 years	-0.422	0.461
nhb	Bayley Cognitive	398	1-2 years	0.477	0.628
pvd	Bayley Cognitive	421	2 years	0.51	0.322
pvd	WPPSI	225	4 years	2.14	0.66
pvd	Raven	415	2 years	1.35	0.491
pbt	WASI	147	6 years	-1.064	2.087
scc	General IQ	397	5.5 to 6 years	-0.050	1.020

Table 4.3: *Estimated regression coefficients and standard errors when propensity scores are used within each study. WISC stands for Weschler Intelligence Scales test. WPPSI stands for Wechsler Preschool and Primary Scale of Intelligence test. WASI stands for Wechsler Abbreviated Scale of Intelligence test.*

The result of performing random effect meta-analysis on regression coefficients is summarised in Table 4.3 and the forest plot is shown in Figure 4.2. Meta-analysis was performed using `metafor` (v2.0.0) (Viechtbauer, 2010) software package in R (v3.4.3) (R Core Team, 2017). We have anonymised the names of the studies during the meta-analysis process. The forest plot in Figure 4.2 shows that the overall effect of average growth velocity on cognitive outcome is slightly positive (mean effect size of 0.46 with (95%CI 0.31, 0.61). Figure 4.2 also indicate that there are considerable variations between the estimates in each study; most of the studies with positive effects have narrow confidence intervals, while studies with negative effects have wider confidence intervals. The measure of between study variance, τ^2, estimated using restricted maximum likelihood REML, in $\tau^2 = 0.0439$ (95%CI 0.0053, 0.4524). The I^2 statistics, which indicates the percentage of of total variation across studies that are due to heterogeneity rather than chance (McKenzie, Beller, and Forbes, 2016) is estimated to be 27% (95%CI 4.298%, 79.21%).

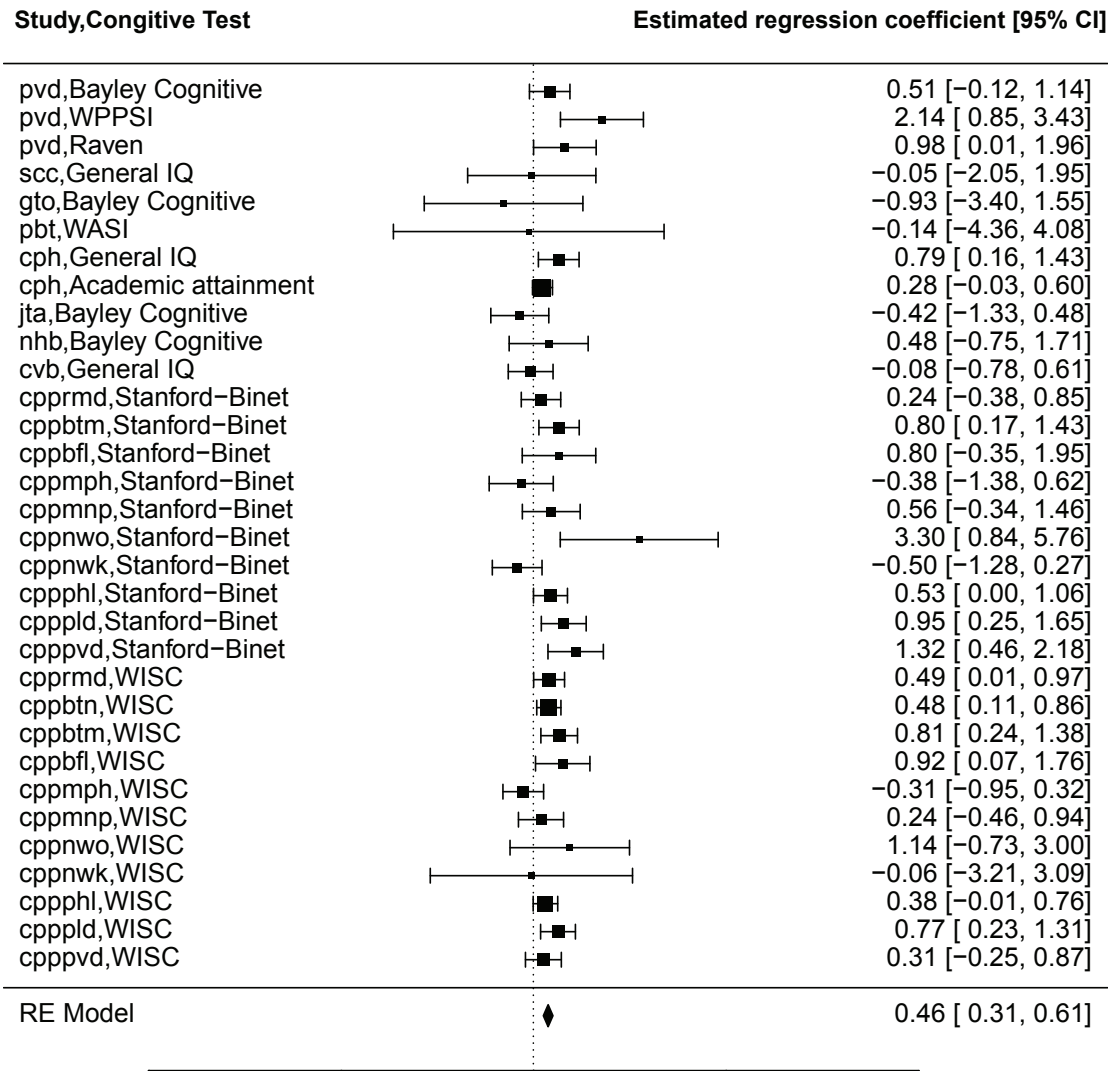


Figure 4.2: *Forestplot from the random effect meta-analysis of the estimated regression coefficients with covariate adjustment using propensity scores. The vertical dotted line indicates no effect. There is an overall statistically significant effect of average first year growth velocity on the cognitive test scores 0.46 (95% CI 0.31, 0.61).*

4.5 Random effect one stage analysis model

When we have all the individual level data across all studies, an alternative method of meta-analysis is to fit a mixed model regression using all the data. This approach of performing analysis using all the individual level data is known as *one-stage* individual participant data (IPD) meta-analysis.

Within the context of IPD meta-analysis, the approach to obtain the results above is known as *two-stage* IPD meta-analysis because there are two distinct steps involved in the analysis process. We first perform our analysis on each of the studies, then combine the results using a random effect meta-analysis model. Recall our outcome regression model for each study, shown in (4.6) consists of cognitive score regressed on the average first year growth velocity, and the estimated propensity score.

To aid the explanation of the two IPD models, we first rewrite the outcome regression model shown in (4.6) by removing the indexation for imputation. Recall in the outcome regression model, y is the cognitive ability measured by scores in a cognition test, z is the average first year growth velocity and ρ is the estimated propensity score. The first part of our two-stage IPD meta-analysis consist of fitting the following regression model for jth subject in kth study

$$y_{jk} = \beta_0 + \beta_1 z_{jk} + \beta_2 \rho_{jk} + \epsilon_{jk}\,, \tag{4.8}$$

where $\epsilon_{jk} \sim \mathcal{N}(0, \sigma_g^2)$, and coefficient of interest is β_1. For κ studies, we'll have κ number of estimated regression coefficients $\{\hat{\beta}_{11}, \hat{\beta}_{12}, \ldots, \hat{\beta}_{1\kappa}\}$. The second step of the two-stage IPD meta-analysis consists of combining $\{\hat{\beta}_{11}, \hat{\beta}_{12}, \ldots, \hat{\beta}_{1\kappa}\}$ using a weighted average.

In contrast, the one-stage IPD meta-analysis fit a mixed model using all the data from all the studies. Covariates that vary from study are modelled as random effects. For the model in (4.8), one possible one-stage IPD random effect model for jth subject in kth study is

$$\begin{aligned} y_{jk} &\sim \mathcal{N}(\mu_{jk}, \sigma_g^2)\,, \\ \mu_{jk} &= \beta_{0k} + \beta_{1k} z_{jk} + \beta_{2k} \rho_{jk}\,, \\ \beta_{1k} &\sim \mathcal{N}(\beta_1, \tau^2)\,. \end{aligned} \tag{4.9}$$

In model (4.8), to model the study to study variation of the coefficient of interest β_1, we specify a random effect by modelling the kth coefficient β_{1k} to be a random sample from a sample from a distribution with the true value β_1 as it's mean. Model (4.8) also takes into account the study to study variation (τ^2) and within study variation (σ^2). Another way to represent (4.9) is

$$y_{jk} = \beta_0 + (\beta_1 + a_{1k}) z_{jk} + \beta_{2k} \rho_{jk} + \varepsilon_{jk}\,, \tag{4.10}$$

with $\varepsilon_{jk} \sim \mathcal{N}(0, \sigma^2)$. and $a_1 \sim \mathcal{N}(0, \tau^2)$.

We have modelled β_0, the intercept as a common effect, although it's possible to model it on a per study basis as a fixed effect, or model it as another random effect (Debray et al., 2015). The coefficient for average first year velocity is modelled as a random effect (random

regression slope). The effect of propensity scoring in (4.10) is modelled as a fixed effect, even though we know it vary study by study.

Similar to the definition of random effect by Searle, Casella, and McCulloch (1992), we treat average first year growth velocity covariate as random effect because it is part of a larger population of growth velocities from different studies. However, the propensity scores covariate is not suitable to be viewed in this way. In our regression model, the role of propensity scores is that of covariate adjustment. The propensity scores are not treated as random effect because the random effect structure is unclear. That is, it is not clear whether it can be argued that the propensity scores are random samples from a larger population of propensity scores across different studies.

The formula format to pass into `lmer` function of `lme4` package in R for the mixed model representation shown in (4.10) is

$$cognition \sim velocities + pscores \times study + (0 + velocities|study) \tag{4.11}$$

In (4.11), we first specify the random effects are based on studies. We then specify the average first year growth velocity to be a random slope; we do not have any random intercept. For the propensity score term, to model a fixed effect that varies across studies, we specify an interaction term between the propensity score and studies. Note that again, the interaction term specification for `lmer` function is to allow the propensity score covariate to vary study by study.

We apply this random effects mixed modelling to studies with `cpp` prefix, which makes up a majority of the studies used for the results in Figure 4.2, but nevertheless have enough studies). The IPD mixed model estimated the effect of average growth velocity to be 0.42191 (SE 0.07634), this is in the interval from the random effect model 0.4146 (SE 0.0824).

A question of interest is given that we have individual participants data, and IPD meta-analysis is considered as gold standard, in what scenarios do we perform two-stage analysis instead. The first scenario is when all the individual participants data are not available, since the one-stage analysis requires individual participant data across all studies. The second scenario is when the computation cost to model the one-stage becomes too large (Damesa et al., 2017). In terms of addressing very large datasets and distributed data, the two-stage analysis is more suitable when used in conjunction with approaches to address analysis on large datasets such as as divide and recombine (described in Section 2). In area of agriculture trials, Damesa et al. (2017) and Piepho et al. (2012) show how statistical models fitted on a per trial basis can be combined to achieve the same model as fitting a large statistical model using individual trial data. In terms of missing data, imputation in the context of IPD can be

much more complicated. As discussed by Quartagno and Carpenter (2016), there are two forms of missing data in the IPD context. Firstly, data can be missing within individual studies, and secondly, an entire variable can be missing because it was not collected for that study. Quartagno and Carpenter (2016) argue that the second case is quite common since IPD meta-analyses are often based on opportunistic gathering of datasets. If we are to perform an IPD meta-analysis, the missing data in the HBGDki datasets fall into the second category since the studies have different designs and purposes. The imputation method proposed by Quartagno and Carpenter (2016) in the context of IPD is to use the joint modelling approach as opposed to the fully conditional specification method such as MICE.

4.6 Discussion

We have shown that using propensity scoring allow us to perform meta-analysis on a large number of studies with varying covariate structure in a systematic and automatic way. By using propensity scores as a covariate in our regression model, we are able to perform meta-analysis using a a common model across our studies, as opposed to performing meta-analysis on measures that come from different models.

We can abstract the concept of distributed data from from Chapter 2 such that they are analogous to the studies in a meta-analysis. Thinking of distributed data this way provides us with another method of performing statistical analysis on distributed computing system. In this approach, we first divide a large data set by studies, we then perform the statistical procedures outlined in this chapter such as multiple imputation and propensity score estimation within each study. After that we perform the regression, then finally combine the per study estimates of regression coefficients using meta-analysis. Applying our work to the work by McMahan et al. (2017), where mobile phones of people are considered as distributed data, instead of combining the score function using data from individual mobile phones, we can perform a random effect meta-analysis over the regression coefficients estimates from the individual mobile phones. As well, since different mobile phones have different capabilities, the data from the mobile phones can potentially have different covariate structures. We can address this kind of issue through the use of propensity scoring. As a result, even widely used method such as meta-analysis has a place in statistical analysis of big data using distributed computing systems. We now outline several other interesting matters with regards to the work in this chapter.

Firstly, there are many methods to model child growth. On the question of which variables best represent growth and cognition, in this work we use HAZ as the physical measure of growth. Although this is choice is not entirely arbitrary, depending on the goal of the analy-

sis, subject matter experts may prefer other physical growth measures such as weight on age z-score (HAZ) or head circumference size. Similarly, most of the studies we use in the meta-analysis have *global* cognitive test scores such as those from Bayley and Wechsler family of tests. With the involvement of subject matter experts, we may be able to determine other cognitive tests are also comparable, which can lead to more studies being included in our analysis.

In the same way as the variables, there are also many growth trajectory models. In this work, we use *brokenstick* approach to model the first year physical growth. Depending on the question there are many other growth modelling techniques ranging from quadratic model, splines to functional principal component analysis (Anderson et al., 2018). Another growth modelling technique is Superimposition by Translation and Rotation (SITAR) (Cole, Donaldson, and Ben-Shlomo, 2010), which is more sophisticated in that growth trajectories for individual subjects are constructed from applying transformations to the mean growth trajectory of the study. Although we use the brokenstick model, with the involvement of a subject matter expert, our physical growth model may change.

With regards to the use of propensity scores, within traditional setting of a binary 'treatment' variable, techniques such as propensity score matching can be achieved easily. A popular diagnostic of how similar the covariate distribution of subjects in 'treatment' and 'non-treatment' groups is to compare the plot of the distributions of the covariates of the subjects within these groups. Gelman and Hill (2006, Fig 10.7) and Elze et al. (2017, Figure 1) are examples of this kind of diagnostic plots. In this work however, our estimated propensity scores are continuous. As a result, the traditional approach of diagnostic such as histogram may not be possible. Research do exist for continuous propensity score methods. Imai and Van Dyk (2004) generalised propensity score method to non-binary 'treatment' cases with the generalised propensity score (GPS). For example, Yang et al. (2016) show that matching using propensity scores can be performed for multi-level 'treatment'. Formal methods of diagnostic for continuous propensity scores are topic of active research (Yang et al., 2014).

In this work, we have a linear propensity model, and assume that it is correct. Even though there is a possibility of model misspecification, work such as Drake (1993) showed that more bias to the estimates are introduced when the outcome or response model is misspecified, compared with a misspecified propensity model.

As shown in Section 4.4 in the HBGDki project, we have access to subject level level from all the studies, as opposed to just having the regression coefficients and their standard errors from study investigators. As a result, it is possible to fit a random effect model using all the individual patient data, as opposed to fitting a random effect meta-analysis model. With individual subject data meta-analysis, there is a more sophisticated method of performing data

imputation. In this work we impute the missing values first, fit our regression models, combine the regression coefficients, then fit the meta-analysis model. In our imputation process, we ignore any possible study to study variation. That is, the plausible data imputed are drawn from observed data within the study. A more complex approach suggested by Quartagno and Carpenter (2016) is to use a multilevel joint modelling approach, with a random study specific covariance matrix. Quartagno and Carpenter (2016) claim that their method allows IPD imputation to retain the between study heterogeneity. An important difference we need to consider when imputing data this way is the order of analysis. As mentioned before, in this work, we perform imputation within studies, then perform meta-analysis ('RR then MA'). If we use the imputation method proposed by Quartagno and Carpenter (2016), we have to treat all the data as one dataset. As a result, when we apply imputation, we perform meta-analysis first, then apply Rubin's rule. This is illustrated in Figure 4.3 where we treat data from all the studies as a single dataset. With this setup, after imputation using a method proposed by Quartagno and Carpenter (2016), for example, we fit our random-effect meta-analysis model for each imputed dataset. We then combine the estimates from the meta-analysis models using Rubin's Rule. This is equivalent to performing regression on each of the imputed data then pooling them.

Although we apply 'RR then MA' approach, the effect of different sequence of operations in different context is an interesting area for future research. For example, in the context of using multiple imputation with only propensity score matching, Mitra and Reiter (2016) explored how to estimate the treatment effect with two methods. In the first method, treatment effect are first estimated using propensity score matching using each of imputed datasets (which has complete data), then these estimates are averaged. In the second method, the average propensity score across all the imputed dataset is first used to perform matching, follow by estimating the overall treatment effect. Similar to the 'RR then MA' and 'MA then RR', each of these approach attempt to address how to perform analysis using the multiple imputed datasets. Unlike the 'RR then MA' approach we use in the context of analysis, Mitra and Reiter (2016) recommend using their second approach (i.e., using the average propensity score across the imputed datasets) to perform matching and subsequent estimate of the treatment effect.

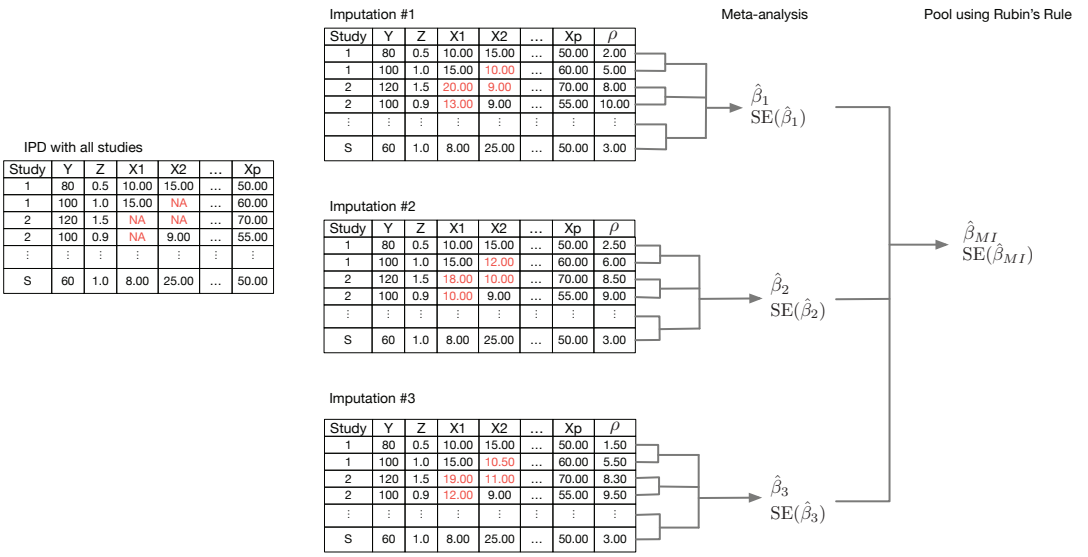


Figure 4.3: *Putting data from all the studies into a single dataset allows, then performing imputation using method such as those by Quartagno and Carpenter (2016) allows us to treat the data as if though we have one dataset. Just as we can fit regression models in each imputed dataset then pool them, we fit our random-effect meta-analysis model for each imputed dataset, then pool those estimates using Rubin's Rule. This is in contrast to our work with HBGDki dataset where we treat the data from each study as a separate data set.*

Appendix 4.A Linear mixed model

The starting point of our work is modelling the physical growth of the children within each study. We model the physical growth using growth trajectories that are constructed using longitudinal growth measurements. We assume that even though the growth trajectories vary from child to child, there is an overall trajectory for all the children within a study. With this assumption, we construct the growth trajectories within a mixed model framework. Anderson et al. (2018) outline alternatives to constructing growth trajectories such as SITAR (Cole, Donaldson, and Ben-Shlomo, 2010). We chose the mixed model approach due to its simplicity.

The actual measurement of physical growth we are interested in is the height (or length) of the children. Physical growth measurement such as height is influenced by age and gender. Instead of explicitly incorporating age and gender into your mixed model, we model on a standardised height measurement Z-scores developed by the World Health Organisation (WHO), which for height is called height-for-age (HAZ) score (Onis et al., 2006).

Assuming y represents HAZ and t as the measurement time, in longitudinal mixed model context we model the growth of the ith subject at jth measurement point as

$$y_{ij} = \beta_0 + \beta_1 t_{ij} + b_{0i} + b_{1i} t_{ij} + \varepsilon_{ij} \quad , \quad j = 1, \ldots, n_i \ , \tag{4.12}$$

with n_i the number of observations for the ith subject. In this formulation, β_0, β_1 are *fixed effects*, and b_{0i}, b_{1i} are *random effects* for the ith subject. The random effects b_{0i} and b_{1i} represent the deviation of the ith subject from the global intercept β_0 and global slope β_1 respectively.

The longitudinal mixed model formulation in (4.12) models the growth trajectories as a straight line. These straight line trajectories fail to capture the variation in the growth pattern of children. Some child may grow then stun, while other may follow the opposite pattern. In order to capture the ups and downs of a child's growth, we extend the longitudinal mixed model of (4.12) to capture non-linear trajectories.

Since child growth is not entirely linear over the first year of age, we refine (4.12) by using linear spline model, specifically the brokenstick approach whereby we construct piecewise linear segments between different time points and join them together. In this form, the time period to create segments are called *knots*. This is the most simplest form of spline model, a later chapter outlines details of the splines. In this work, we place knots at 3 months, 6 months and 9 months of age and construct line segments from birth to 3 months, between the knots, and 9 moths to 12 months of age. These line segments are *truncated line* functions which takes the form $(x)_+$, where $(x)_+ = x$ when x is positive and zero otherwise. If we treat k as a

single knot, the truncated line function at this knot becomes $(t_{ij} - k)_+$. This means

$$(t_{ij} - k)_+ = \begin{cases} t_{ij} - k & t_{ij} > k \\ 0 & t_{ij} \leq k \end{cases} .$$

When we incorporate truncated lines into (4.12), we have

$$\begin{aligned} y_{ij} = \beta_0 + \beta_1 t_{ij} + & \\ & \beta_2(t_{ij} - k_1)_+ + \beta_3(t_{ij} - k_2)_+ + \beta_4(t_{ij} - k_3)_+ + \\ & b_{0i} + b_{1i} t_{ij} + \\ & b_2(t_{ij} - k_1)_+ + b_3(t_{ij} - k_2)_+ + b_4(t_{ij} - k_3)_+ + \\ & \varepsilon_{ij} \quad . \end{aligned}$$

To improve numerical stability, we construct our truncated lines using first order B-Spline (De Boor et al., 1978) with inner knots at 3 months, 6 months and 9 months of age and two outer knots at birth and 12 months of age.

The interpretation of the truncated line segments between the knots is *growth velocity* every three months. From the truncated line segments from this model, we can then calculate the growth velocity. What we are interested is the *average growth velocity* over one year.

5 Estimating smoothing parameter of derivatives of penalised spline

5.1 Introduction

In Chapter 4, we use random effect meta-analysis in conjunction with propensity scoring to investigate the relationship between first year growth velocity of children and their cognition growth. In that particular analysis, we are interested not only with the regression function, but its derivative. The regression function relates the physical growth of a child, measured in height to age z-score, to a set of covariates such as gender, parent's income and so on. The first derivative of the regression function then models the *growth velocity* of a child (i.e., the rate of growth) over a time period, such as one year. Likewise, the second derivative of the regression function models the *growth acceleration* of a child. For non-linear relationships in regression models, we can use techniques such as penalized splines. When we model using penalized splines, we need to estimate the covariates as well as other parameters for the splines. Estimation using techniques such as penalized splines require the selection of a smoothing parameter, usually denoted as λ. This parameter determines how smooth the regression function will be when used with the penalized spline approach. For the estimation of regression function, λ is usually selected using methods such as generalized cross validation (GCV) or restricted maximum likelihood (REML). However, there are no well agreed-upon methods to estimate the optimal smoothing parameter for the derivatives of the regression function, one approach being to naïvely use the same λ to model the derivative.

In a big data setting, it may not be possible to use all the data to estimate the smoothing parameter. If we store the big data in a distributed computing system such as Hadoop and then attempt to estimate the smoothing parameter using distributed data. We can then take the divide and recombine approach by attempting to combine non-covariate parameters such as the smoothing parameters. However, we need to investigate whether the smoothing parameters estimated using individual units of distributed data can adequately model derivatives of regression functions.

In this chapter, we investigate the need for new smoothing parameter selection rules tailor made for the first and derivative of a regression function. Using simulations, we obtain the extra amount of observations needed in a data set when the derivatives of the regression functions are naïvely modelled using the smoothing parameter for the regression function. This calculation is based on the differences of risk measurement between the estimated derivative function modelled using different smoothing parameters.

5.2 Penalized splines

We begin by briefly explaining classical *parametric* regression, then transition into *non-parametric* regression using penalized splines. Suppose there is an outcome variable denoted as y. We can model the relationship between it and a set of linear predictors of length p, $\beta_0, \beta_1, \ldots, \beta_p$ such that for an individual ith observation,

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_p x_{pi} + \varepsilon_i \ , \tag{5.1}$$

where ε_i is known as the *error term* with the property $\mathrm{E}(\varepsilon_i) = 0$. The model in (5.1) can also be represented with the mean of the outcome variable as

$$\mathrm{E}(y_i \mid x_i) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_p x_{pi} \ .$$

In the univariate case where there is one predictor ($p = 1$), the simple linear regression model is

$$y_i = \beta_0 + \beta_1 x_{1i} + \varepsilon_i \ . \tag{5.2}$$

For example, the relationship between the growth of a child, such as the child's height and predictors such as the child's gender and parents' income, can be modelled in classical parametric regression as

$$\texttt{height}_i = \beta_0 + \beta_1 \texttt{gender}_i + \beta_2 \texttt{parincome}_i + \varepsilon_i \ , \tag{5.3}$$

where `gender` is the gender of the child and `parincome` is the income of the parents of the child. We can also represent the regression equation (5.1) in vector notation such that $\boldsymbol{y}$ is a $n \times 1$ *column vector* containing n outcome values (such as the child's height), $\boldsymbol{X}$ is a $n \times p$ matrix, also known as a *design matrix* that contains observed values for p co-variates. Finally, $\boldsymbol{\varepsilon}$ is a $n \times 1$ vector that contains the error terms. That is

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix},$$

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{p1} \\ 1 & x_{12} & \cdots & x_{p2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & \cdots & x_{pn} \end{bmatrix}, \text{ and} \qquad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix} .$$

The regression equation in vector form is then

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \ .$$

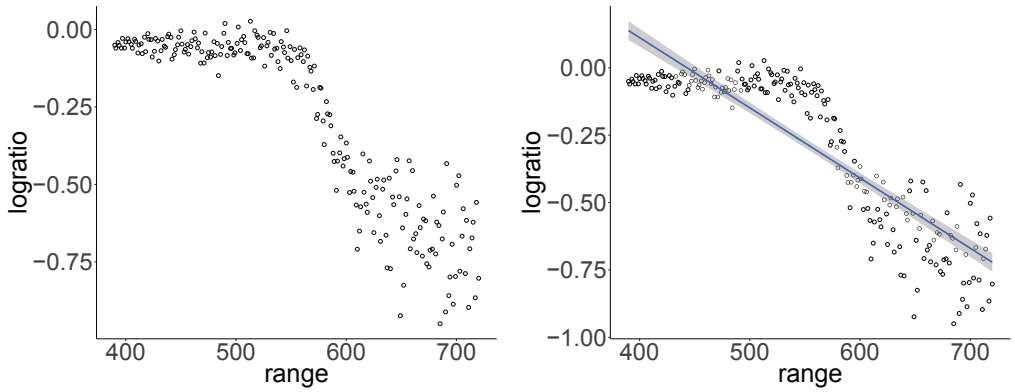
In regression, whether it be parametric or semi-parametric, what we are interested is estimating the coefficients $\boldsymbol{\beta}$. The estimates of the coefficients, $\widehat{\boldsymbol{\beta}}$ are obtained using technique such as the least-squares method

$$\text{minimize} \sum_{i=1}^{n} \left[y_i - (\beta_0 + \beta_1 x_{1i}) \right]^2 .$$

The model in (5.1) and (5.2) are considered as linear because it is linear in the parameters ($\boldsymbol{\beta}$). There are cases where the linearity between the the mean response and the predictors can not be assumed. An example of a data set where non-linear model is more appropriate is the LIDAR data from Sigrist, Winefordner, and Kolthoff (1994). In these cases, non-linear models are required. A scatter plot of the outcome variable `logratio` against the single predictor `range` it is shown in Figure 5.1a. Linear regression for this data-set is not appropriate because the assumption of linearity of mean response does not hold. In addition, the variance is not constant around the linear mean response. One way to deal with non-linearity is by transforming the outcome. However, this process can be delicate and time consuming, and is not always viable. As an alternative, we can use polynomial models as well. However, fitted curves based on polynomial model can oscillate wildly due to their constraints of being able to interpolate the data and also to have all the derivatives with respect to the variables (e.g., the range variable in Figure 5.1a) be continuous(Wood, 2017, Section 4.2.1). These oscillations are not the result of the features of the data, but due to the polynomial models, and they can become problematic when modelling the derivatives (Ruppert, Wand, and Carroll, 2003b, Section 2.7). A better way is to use non-parametric regression techniques which have the ability to handle the non-linearity automatically as part of of the model (Ruppert, Wand, and Carroll, 2003a, Chapter 3).

5.3 Non-Parametric regression

We explore non-parametric regression by explaining how to handle non-linearity in the univariate case using scatterplot smoothing. This simple model allows us to introduce notation and concepts relate to non-parametric regression. When we use non-parametric regression, the modelling of the relationship between the outcome and predictors is no longer in the form of (5.2). Instead, the model now consists of a *function* of the predictor term. For the univariate



(a) *LIDAR data.* (b) *LIDAR data with mean response*

Figure 5.1: *The LIDAR data (Holst et al., 1996) is an example of a data that is not suitable for linear models. The assumption of a linear relationship between the mean of the outcome and the predictors do not hold. In addition, the variance around the mean of the outcome is not constant.*

case, we denote the single predictor as x. Ignoring the intercept term for now, the non-linear model becomes

$$y_i = f(x_i) + \varepsilon_i \ , \tag{5.4}$$

where $1 \leq i \leq n$, for n observations. The *underlying trend* of (5.4) would be

$$\mathrm{E}(y_i \mid x_i) = f(x_i) \ .$$

In a penalized spline model, the function of the predictor term takes the form

$$f(x) = \beta_0 + \beta_1 x + \sum_{k=1}^{K} u_k \, z_k(x) \ , \tag{5.5}$$

where z_k are *spline basis functions* and K is the number of basis functions. In our work, the spline basis functions z_k, are cubic *O'Sullivan splines* (Wand and Ormerod, 2008). We explain the reason for using O'Sullivan spline basis functions in a later section. Since O'Sullivan splines are a generalisation of smoothing splines, the parameters β_0, β_1 and the coefficients for the spline basis functions u_k in (5.5) can be estimated from the solution to the general optimization problem for smoothing spline

$$\text{minimize}\left[\sum_{i=1}^{n} \{y_i - f(x_i)\}^2 + \lambda \int_{-\infty}^{\infty} f^{(r)}(x)^2 \, \mathrm{d}x \right] , \tag{5.6}$$

where $r = 2$, i.e., $f''(x)^2$. The integral $\int_{-\infty}^{\infty} f^{(r)}(x)^2 \, \mathrm{d}x$ is known as the *roughness penalty*, and λ is known as the *smoothing parameter*. With $r = 2$, the solution to (5.6) is the commonly used cubic smoothing splines. In general, the solution to (5.6) is $2r + 1$ degree smoothing splines. For further details on smoothing spline penalty, refer to Ruppert, Wand, and Carroll (2003a, Chapter 3) and Hastie, Tibshirani, and Friedman (2009, Chapter 5). We prefer O'Sullivan spline basis functions because they allow us to use a smaller number of basis functions compared to smoothing splines. With smoothing splines, the number of basis functions roughly equal the sample size. As a result, as the sample size increases, the computational cost of fitting will increase as well (Wand and Ormerod, 2008). We instead use O'Sullivan spline basis functions, which are low-rank approximation of smoothing splines with a reasonable choice of K in (5.5) (Harezlak, Ruppert, and Wand, 2018, Section 2.2) and (Wand and Ormerod, 2008). Representing (5.5) in matrix form, we first define two new matrices as shown in Harezlak, Ruppert, and Wand (2018, Section 2.6)

$$\boldsymbol{C} = \begin{bmatrix} 1 & x_1 & z_1(x_1) & \dots & z_K(x_1) \\ 1 & x_1 & z_1(x_1) & \dots & z_K(x_2) \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n & z_1(x_n) & \dots & z_K(x_n) \end{bmatrix} \quad \text{and} \quad \boldsymbol{D} = \begin{bmatrix} 0_{2\times 2} & 0_{2\times K} \\ 0_{K\times 2} & I_K \end{bmatrix}.$$

The fitted values using these are then

$$\begin{bmatrix} \widehat{f}(x_1;\lambda) \\ \widehat{f}(x_2;\lambda) \\ \vdots \\ \widehat{f}(x_n;\lambda) \end{bmatrix} = \boldsymbol{C}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \boldsymbol{y} \, .$$

Although in (5.6) both the number of basis functions K and the smoothing parameter λ affect the penalised spline, the smoothing parameter affects the resulting penalised spline much more. If the smoothing parameter is too low the data will be undersmoothed and overfitting will occur. As λ approaches zero, the fitted function is close to interpolation of the data. On the other hand, if the smoothing parameter is too high, the data will be oversmoothed (Wood, 2017). As the value of λ becomes higher, the fitted function will become the least-squares line fit, since the spline basis functions have too little influence (Harezlak, Ruppert, and Wand, 2018).

One way to estimate the smoothing parameter is through visual inspection of the fitted function. However, it is more preferable to estimate the smoothing parameter from the data (Harezlak, Ruppert, and Wand, 2018). A widely used data-driven method is *generalized cross validation* (GCV), proposed by Craven and Wahba (1978). Details on GCV can be found in Sec-

tion 2.6 Harezlak, Ruppert, and Wand (2018) and Section 4.2.3 of Wood (2017). The other method of estimating the smoothing parameter is to reformulate (5.5) as a mixed-effect model, by specify a distribution for u_k and use *restricted maximum likelihood* (REML) to estimate the smoothing parameter. See Section 2.7 of Harezlak, Ruppert, and Wand (2018) and Section 4.2.4 of Wood (2017) for details on using REML to estimate the smoothing parameter.

So far we only have presented the modelling of the regression function. We now present the a way to model the derivatives of the regression function. We begin by representing (5.5) using matrices, then decompose it into two parts such that

$$\hat{f}(x) = \boldsymbol{X}_x \widehat{\boldsymbol{\beta}} + \boldsymbol{Z}_x \widehat{\boldsymbol{u}} \,, \tag{5.7}$$

where

$$\boldsymbol{X}_x = \begin{bmatrix} 1 & x \end{bmatrix} \,, \quad \widehat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{bmatrix} \,,$$

$$\boldsymbol{Z}_x = \begin{bmatrix} z_1(x) & z_2(x) & \dots & z_K(x) \end{bmatrix}, \quad \text{and} \quad \widehat{\boldsymbol{u}} = \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_k \end{bmatrix} \,.$$

We include the derivative level in (5.7) using the notation (r), such that

$$\hat{f}^{(r)}(x) = \boldsymbol{X}_x^{(r)} \widehat{\boldsymbol{\beta}} + \boldsymbol{Z}_x^{(r)} \widehat{\boldsymbol{u}} \,. \tag{5.8}$$

Using this notation, $r = 0$ models the the regression function, $r = 1$ models the first derivative of the regression function, and $r = 2$ models the second derivative of the regression function. The design matrix in (5.8) at different levels of the derivatives are

$$\boldsymbol{X}_x^{(r)} = \begin{cases} \begin{bmatrix} 1 & x \end{bmatrix}, & r = 0 \\ \begin{bmatrix} 0 & 1 \end{bmatrix}, & r = 1 \\ \begin{bmatrix} 0 & 0 \end{bmatrix}, & r = 2 \end{cases} \,.$$

Likewise

$$\boldsymbol{Z}_x^{(r)} = \begin{cases} \begin{bmatrix} z_1(x) & z_2(x) & \dots & z_K(x) \end{bmatrix}, & r = 0 \\ \begin{bmatrix} z_1'(x) & z_2'(x) & \dots & z_K'(x) \end{bmatrix}, & r = 1 \\ \begin{bmatrix} z_1''(x) & z_2''(x) & \dots & z_K''(x) \end{bmatrix}, & r = 2 \end{cases} \,,$$

and $\boldsymbol{C}$ from Section (5.3) becomes

$$\boldsymbol{C}^{(r)} = \begin{bmatrix} \boldsymbol{X}_x^{(r)} & \boldsymbol{Z}_x^{(r)} \end{bmatrix} .$$

For the derivative of the fitted function, there are no generally agreed upon method such as GCV or REML to estimate the smoothing parameter. Charnigo, Hall, and Srinivasan (2011) attempt to estimate the derivative function directly, but the method is computationally expensive since it require a set of proxy data set to be generated. Simpkin and Newell (2013) propose applying additional penalty to estimate the derivatives, which then require selecting additional smoothing parameters. Simpkin et al. (2018) also outline how to model the derivative, with the smoothing parameter derived using mixed model representation with error terms from the data function. In this chapter, we estimate the smoothing parameter for the derivative by deriving the smoothing parameter estimation method in Wand (1999) to the derivatives. Since the estimation involves the evaluating the performance criteria of the fitted function, we now outlines these.

5.4 Performance criteria

A measurement of the differences between the fitted function and the true function is necessary to automatically select the smoothing parameters. Ruppert, Wand, and Carroll (2003a) states that being able to measure this difference (also called the *error*) is a cornerstone of statistical estimation theory. We denote the estimate of the regression function, which uses a *fixed* smoothing parameter (λ) as $\widehat{f}(x_i\,;\,\lambda)$. The first error measurement is the *mean squared error* (MSE) (Ruppert, Wand, and Carroll, 2003a), which is defined per observation as

$$\text{MSE}(\widehat{f}) = \text{E}\big[\,\{\widehat{f}(x_i) - f(x_i)\}^2\,\big] .$$

If we are interested in comparing the entire fitted curve, one possible measure is the *mean summed squared error* (MSSE) (Ruppert, Wand, and Carroll, 2003a) defined as

$$\text{MSSE}(\widehat{f}) = \text{E}\Big[\,\sum_{i=1}^{n}\{\widehat{f}(x_i) - f(x_i)\}^2\,\Big] .$$

Another measurement across the entire function is the *mean average squared error* (MASE) (Wand, 1999) defined as

$$\text{MASE}(\widehat{f}) = \frac{1}{n}\,\text{MSSE}(\widehat{f}) .$$

An advantage of both MSSE and MASE is that they can be decomposed into bias and variance terms. MASE can be rewritten as

$$\text{MASE}(\widehat{f}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\{\text{E}[\widehat{f}(x_i)] - f(x_i)\}^2}_{\text{average squared bias}} + \underbrace{\frac{1}{n}\sum_{i=1}^{n}\text{var}[\widehat{f}(x_i)]}_{\text{average variance}} .$$

In this chapter, we will use MASE as our error measurement. The decomposition of the error term into the bias and variance components demonstrates the concept of bias-variance tradeoff, where larger values of smoothing parameter λ lead to an increase in bias but a decrease in variance. Going the other way, smaller values of λ will result in decrease in bias but an increase in variance (Ruppert, Wand, and Carroll, 2003a). For known f, the *theoretical* optimal smoothing parameter λ_{MASE} is

$$\lambda_{\text{MASE}} = \text{argmin MASE}(\widehat{f}) .$$

We can speed up the numerical search for λ_{MASE} by approximating a suitable starting point. To achieve this, we use Asymptotic MASE (AMASE) from Wand (1999, Equation 5), which is a good approximation to λ_{MASE}, and it can be computed quickly. However, Wand (1999, Equation 5) is only applicable to the case of regression function. Since we are interested in the derivatives of the functions, we incorporate the derivative notations from Section (5.3), and obtain a general form of Wand (1999, Equation 5) for the derivatives such that $\lambda^{(r)}_{\text{MASE}}$ is

$$\lambda^{(r)}_{\text{AMASE}} = \frac{\sigma^2 \,\text{tr}[\{\boldsymbol{G}^{(r)}\}^{-1}\boldsymbol{D}]}{\| \boldsymbol{C}^{(r)}\{\boldsymbol{G}^{(r)}\}^{-1}\boldsymbol{D}(\boldsymbol{C}^{(r)^\top}\boldsymbol{C}^{(r)})^{-1}\boldsymbol{C}^{(r)^\top} f^{(r)} \|^2 + \sigma^2 \,\text{tr}\left[\left\{\{\boldsymbol{G}^{(r)}\}^{-1}\boldsymbol{D}\right\}^2\right]} ,$$

where

$$\boldsymbol{G}^{(r)} = \boldsymbol{C}^{(r)^\top}\boldsymbol{C}^{(r)} .$$

5.5 Efficiency Quantification

As we discussed in Section 5.3, estimation of $f^{(1)}$ and $f^{(2)}$ via penalized splines still requires choice of smoothing parameters for each estimate. A commonly used naïve approach is to use the same smoothing parameter chosen for estimation of f, typically using a data-driven method such GCV or restricted maximum likelihood. However the result of these methods, $\widehat{\lambda}^{(0)}_{GCV}$ is a consistent estimator of the optimal smoothing parameter for $\lambda^{(0)}_{MASE}$ (Härdle, Hall, and Marron, 1988) but there is no such consistency regarding $\lambda^{(1)}_{MASE}$ and $\lambda^{(2)}_{MASE}$. This raises the question: how much we lose by using a sub-optimal $\lambda^{(0)}_{MASE}$ estimate compared with an

estimate that is consistent for $\lambda^{(r)}_{MASE}$ when $r = 1, 2$? That is, the information loss resulting from using a sub-optimal smoothing parameter when $r = 0$, to model the derivatives, $r = 1, 2$. As a prelude to future work on consistent estimation of $\lambda^{(r)}_{MASE}$, $r = 1, 2$ we will, in this section, study the loss of efficiency in for synthetic data where all optimal smoothing parameters and MASE computations can be performed exactly.

To quantify this loss, we start with the same number of observations n_{orig}, we attempt to find how many extra observations are needed when we use $\lambda^{(0)}_{\text{MASE}}$ to estimate $f^{(1)}$ and $f^{(2)}$, that will result in the same MASE value as when we use $\lambda^{(1)}_{\text{MASE}}$ and $\lambda^{(2)}_{\text{MASE}}$. The simulation begins by uniformly generating values of x_i, the input data to the simulation function, where $1 \leq i \leq n_{\text{orig}}$. We then generate four different variations of the of the noise and 'spatial' functions from Wand (2000, Table 1) and their derivatives. As the name suggests, the varying noise functions have varying noise in their data. The 'spatial' functions are not spatial in the sense of geographical space, but rather varying shape of the function.

The simulation process is as follows

- Select $\lambda^{(0)}_{\text{MASE,norig}}$ to estimate $f^{(0)}_{\text{norig}}$
- Follow the naïve approach by using $\lambda^{(0)}_{\text{MASE,norig}}$ to estimate the derivative functions $f^{(1)}_{\text{norig}}$ and $f^{(2)}_{\text{norig}}$ and obtain $\text{MASE}^{(1)}_{\text{naïve,norig}}$ and $\text{MASE}^{(2)}_{\text{naïve,norig}}$.
- Select $\lambda^{(1)}_{\text{MASE,norig}}$ to estimate $f^{(1)}_{\text{norig}}$, obtaining $\text{MASE}^{(1)}_{\text{opt}}$.
- Likewise, select $\lambda^{(2)}_{\text{MASE,norig}}$ to estimate $f^{(2)}_{\text{norig}}$, obtaining $\text{MASE}^{(2)}_{\text{opt}}$.
- Calculate the differences of the MASE values for each derivative function. That is, calculate
  - $\text{MASE}^{(1)}_{\text{diff,norig}} = \text{MASE}^{(1)}_{\text{opt}} - \text{MASE}^{(1)}_{\text{naïve,norig}}$
  - $\text{MASE}^{(2)}_{\text{diff,norig}} = \text{MASE}^{(2)}_{\text{opt}} - \text{MASE}^{(2)}_{\text{naïve,norig}}$
- For each derivative of the function, iterate while $\text{MASE}^{(r)}_{\text{opt}} \geq \text{MASE}^{(r)}_{\text{naïve,ncurr}}$
  - Increase the number of observations such that x_i, $1 \leq i \leq n_{\text{curr}}$.
  - Generate $f^{(0)}_{\text{ncurr}}$ and select a new $\lambda^{(0)}_{\text{MASE,ncurr}}$.
  - Use $\lambda^{(0)}_{\text{MASE,ncurr}}$ to estimate the derivative function $f^{(1)}_{\text{ncurr}}$ and $f^{(2)}_{\text{ncurr}}$ and obtain $\text{MASE}^{(1)}_{\text{naïve,ncurr}}$ and $\text{MASE}^{(2)}_{\text{naïve,ncurr}}$.
  - Calculate the MASE difference $\text{MASE}^{(r)}_{\text{diff,ncurr}} = \text{MASE}^{(r)}_{\text{opt}} - \text{MASE}^{(r)}_{\text{naïve,ncurr}}$

- Obtain the sample size difference, which we refer to as the equivalent sample size $n_{\text{equiv}}^{(r)}$ for each derivative functions $n_{\text{equiv}}^{(r)} = n_{\text{curr}}^{(r)} - n_{\text{orig}}$

Figure 5.2 to Figure 5.5 show the result of efficiency quantification for the first derivative of varying noise level data from Wand (2000, Table 1). Within each plot, the shape of the function f and its derivative $f^{(r)}$ when $n_{orig} = 400$ are shown in the top right and bottom left panels respectively. The bottom right panel illustrates the efficiency quantification by plotting the proportion of extra observations against n_{orig}.

The results for the second derivative of the noise data are shown from Figure 5.6 to Figure 5.9. Likewise, the resulting efficiency quantification for varying spatial data are shown from Figure 5.10 to Figure 5.13 for the first derivative, and shown in Figure 5.14 to Figure 5.17 for the second derivative.

The results for modelling the first derivative of the noise data in Figure 5.2 to Figure 5.5, show that in general, the proportion of the effective sample size increases monotonically as the starting number of sample size (n_{orig}) increases. However, this is not the case when the data has high level of noise and n_{orig} is small, as shown in Figure 5.5. In these scenarios, at certain n_{orig} such as the minimum point of the graph in bottom right panel of Figure 5.5 , there may not be any advantages in using the optimal smoothing parameter, since both the optimal and naïve smoothing parameters are used to model a highly noisy data. Beyond this n_{orig}, using the optimal smoothing parameter provides an advantage as the estimated regression function suddenly captures features of the true regression function. See the discussion of Figure 6 and Figure 7 of Marron and Wand (1992) for details regarding this subtle characteristic. Similar argument can be made for the second derivative of the noise data in situations shown in Figure 5.9.

We also modelled the first and second derivative of simulation data we refer to as spatial variation data, where the shape of the function varies. In the first derivative case, when the shape of the function is relatively simple ($j = 1$ and $j = 2$), the effective sample size decreased as we increase n_{orig} (Figure 5.10 and Figure 5.11), which is not what we see with noisy data. A possible explanation is that the derivatives functions at these j levels have relatively less features for the model to capture at low values of n_{orig}. What this means is beyond certain values of n_{orig}, there is no advantage in using the optimal smoothing parameter. When $j = 3$ and $j = 4$, however the effective sample sizes increased with n_{orig} (Figure 5.12 and Figure 5.13). In those cases, using the optimal smoothing parameter does provide an advantage. The issue of not having enough features to model is even more pronounced for the second derivative cases (Figure 5.14 to Figure 5.16). We see that although the proportion of effective sample increased as we increased n_{orig}, these proportions are much larger. For example, in

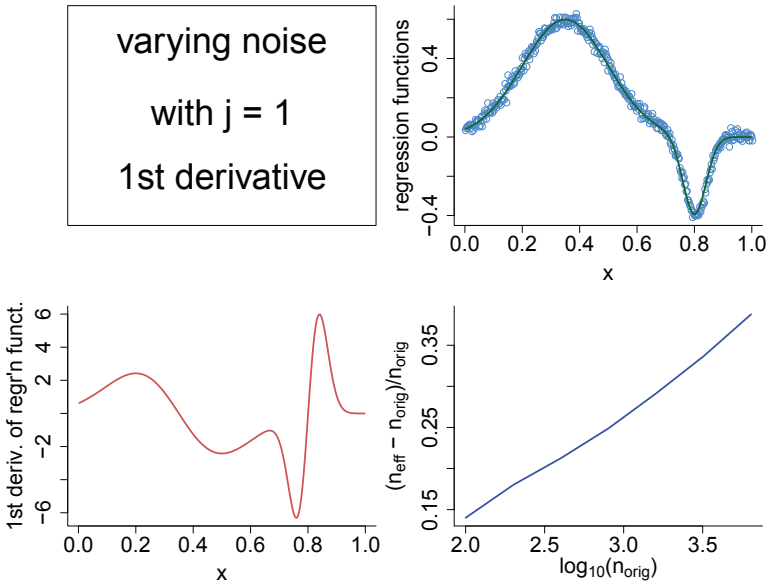


Figure 5.2: *Noise data from Wand (2000) with parameter* $j = 1$. *Top right panel shows the regression function* ($n = 400$). *Bottom left panel shows the first derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size, which ranges from 100 to 6400; each data point is obtained by doubling the starting sample size.*

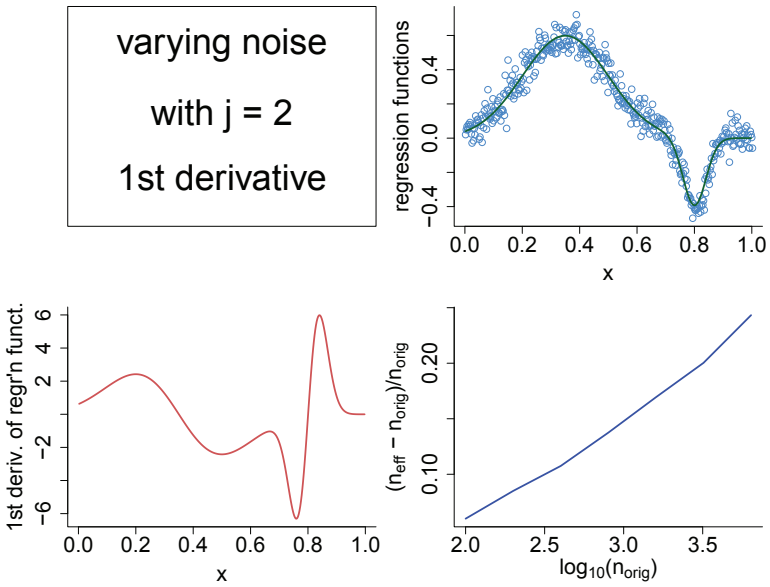


Figure 5.3: *Noise data from Wand (2000) with parameter* $j = 2$. *Top right panel shows the regression function* ($n = 400$). *Bottom left panel shows the first derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size.*

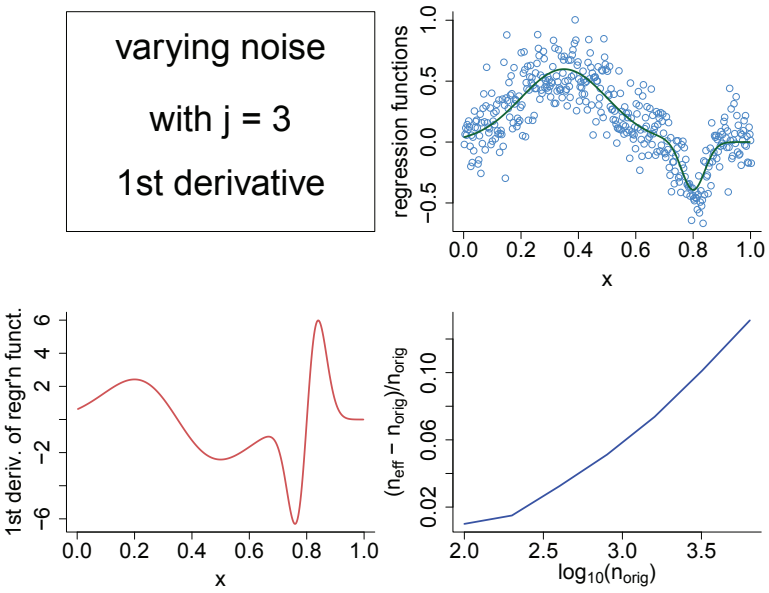


Figure 5.4: *Noise data from Wand (2000) with parameter $j = 3$. Top right panel shows the regression function($n = 400$). Bottom left panel shows the first derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size.*

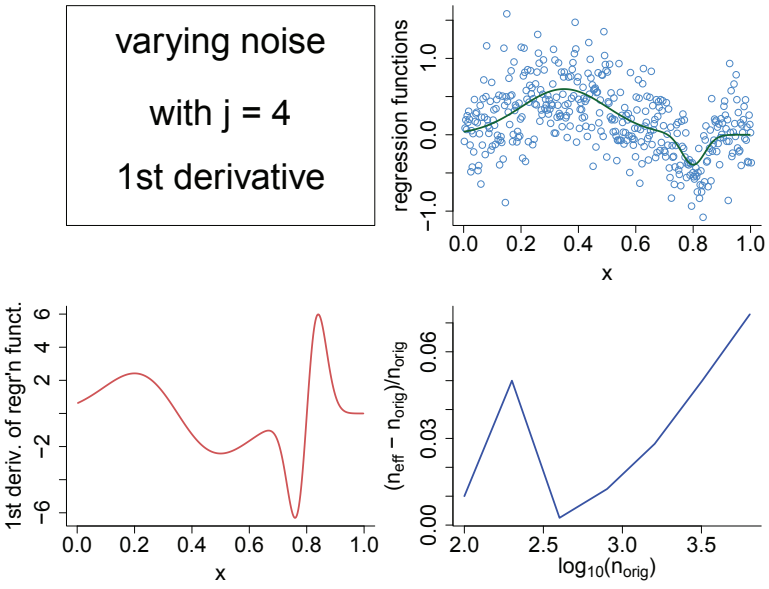


Figure 5.5: *Noise data from Wand (2000) with parameter $j = 4$. Top right panel shows the regression function($n = 400$). Bottom left panel shows the first derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. The non-monotonic nature of the effective sample size plot is due to the different shape of the regression function at low sample sizes.*

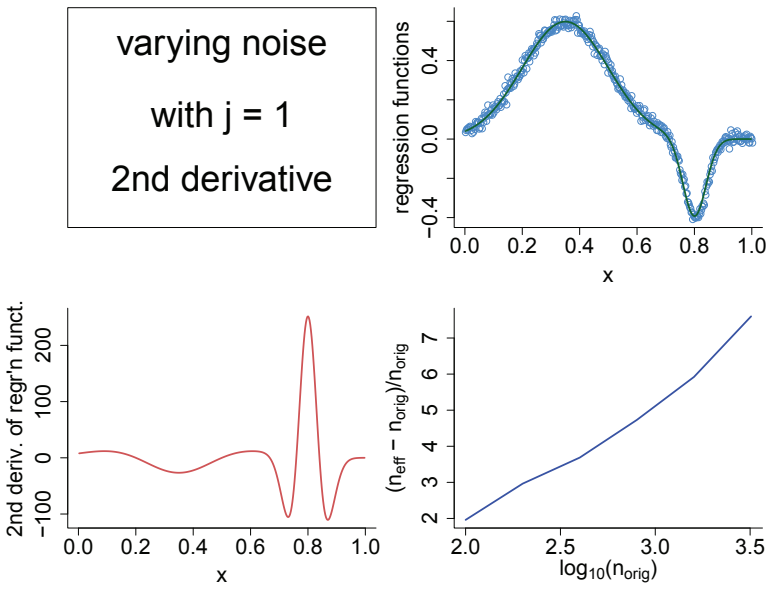


Figure 5.6: *Modelling the second derivative of noise data from Wand (2000) with parameter* $j = 1$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Note the large proportion of effective sample size.*

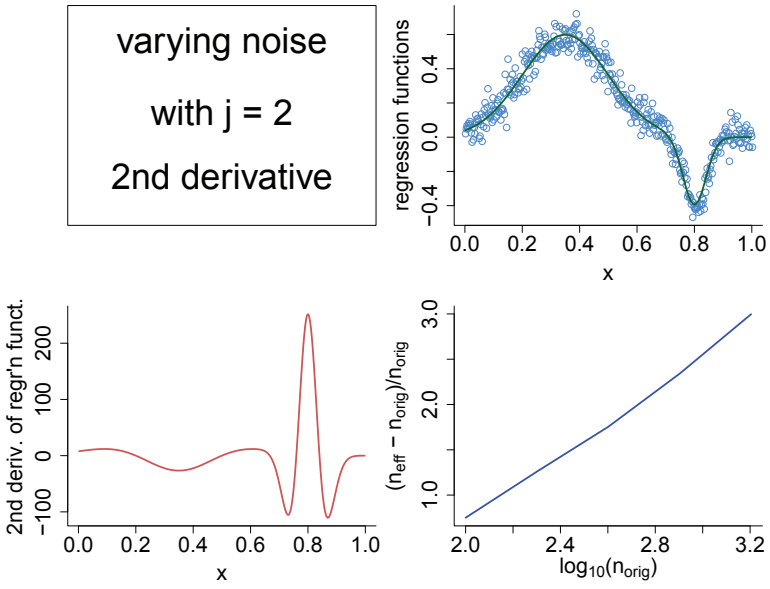


Figure 5.7: *Modelling the second derivative of noise data from Wand (2000) with parameter* $j = 2$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size.*

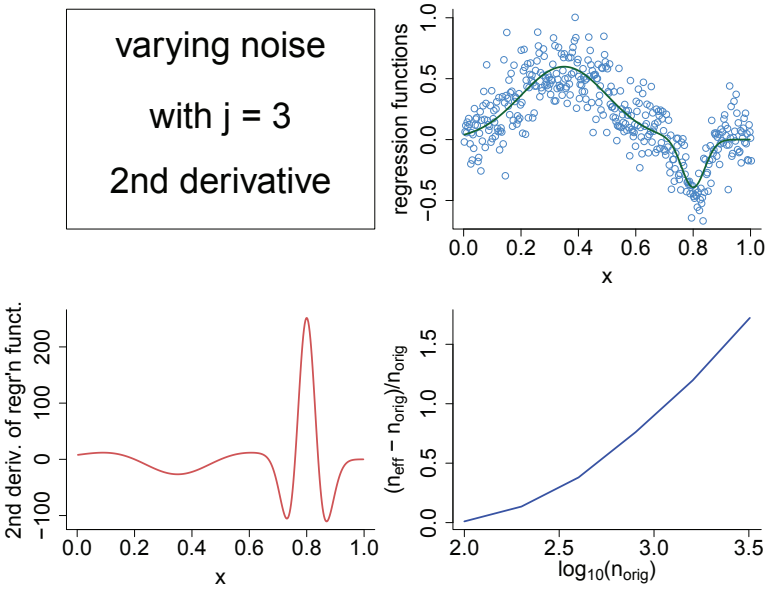


Figure 5.8: *Modelling the second derivative of noise data from Wand (2000) with parameter* $j = 3$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size.*

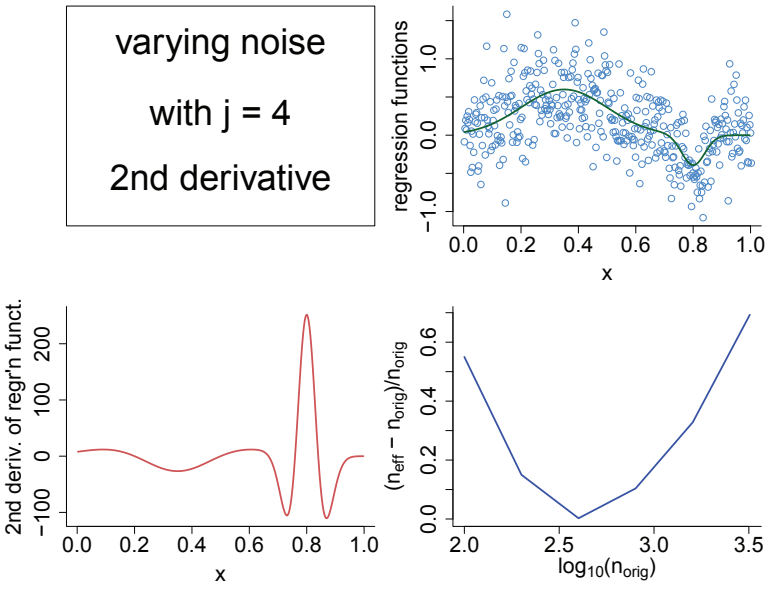


Figure 5.9: *Modelling the second derivative of noise data from Wand (2000) with parameter* $j = 4$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Note the concave shape of the effective sample graph. Above a certain* n_{orig} *value, the model begins to capture the features of the second derivative of the regression function.*

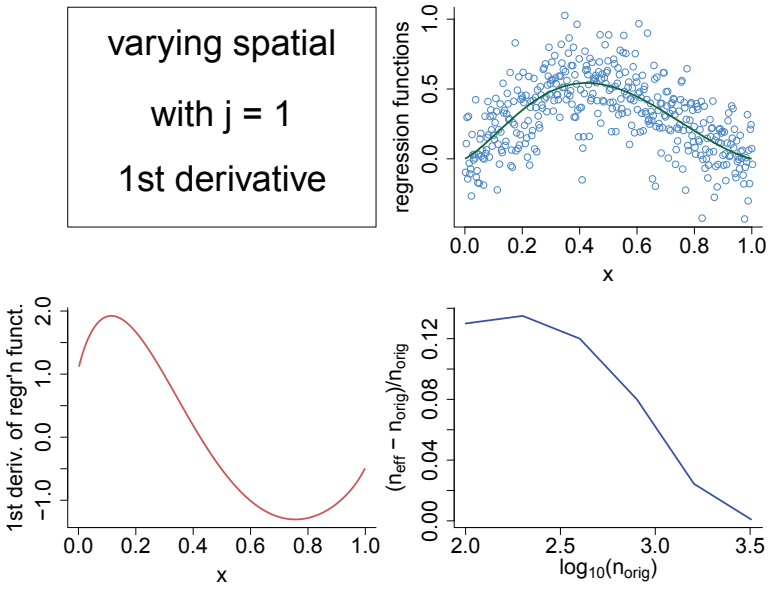


Figure 5.10: *The first derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 1$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Note the decreasing nature of the effective sample size.*

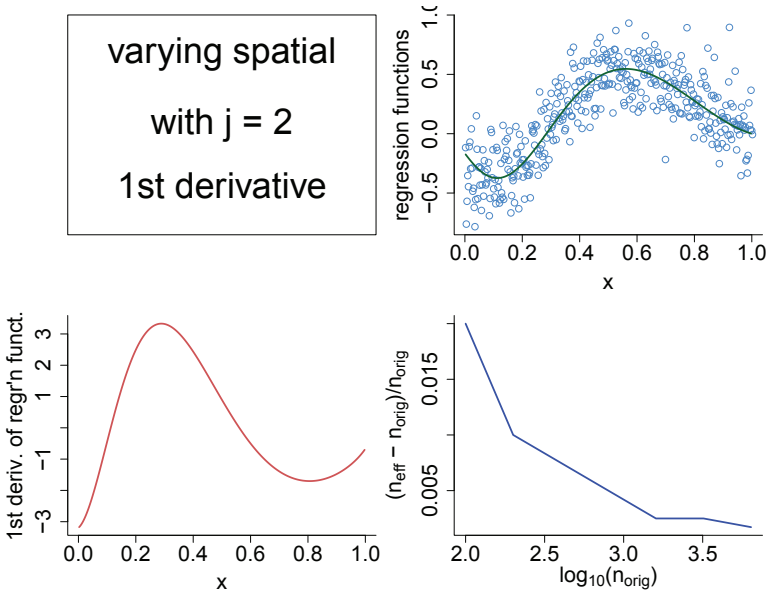


Figure 5.11: *The first derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 2$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Note the decreasing nature of the effective sample size.*

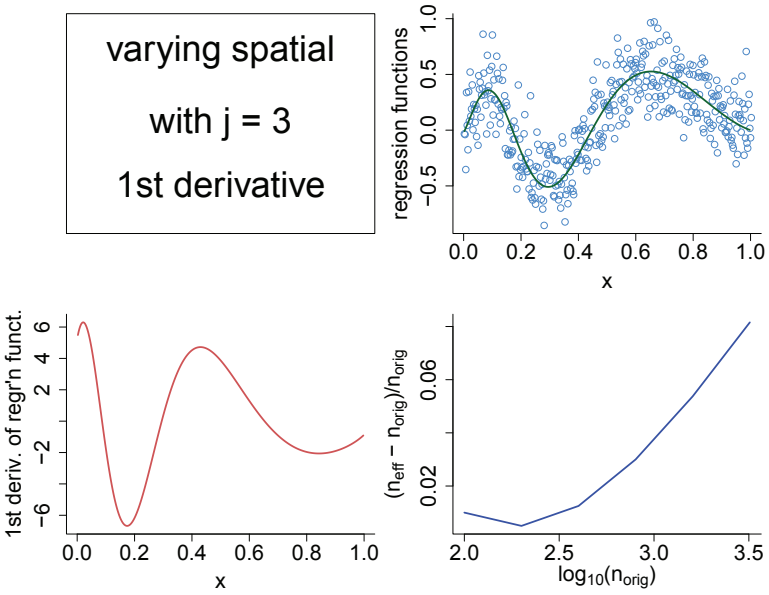


Figure 5.12: *The first derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 3$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Effective sample size increases with* n_{norig}.

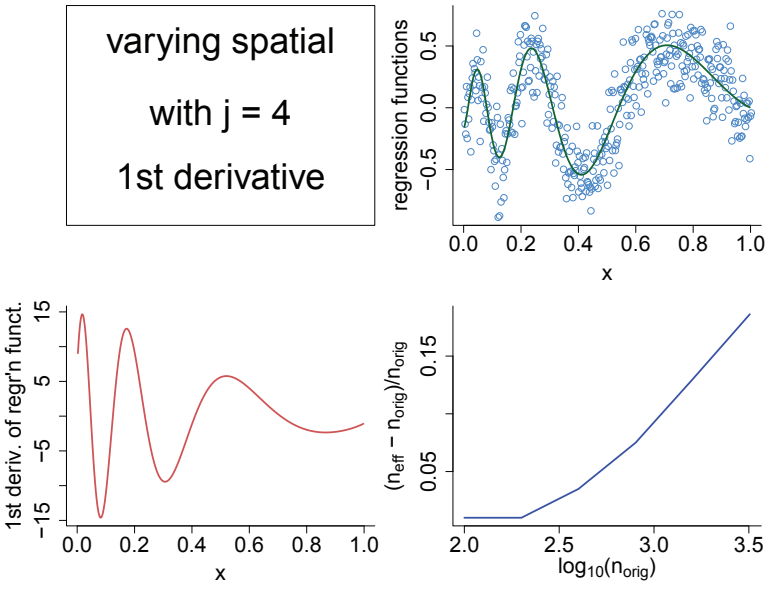


Figure 5.13: *The first derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 4$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Effective sample size increases with* n_{norig}.

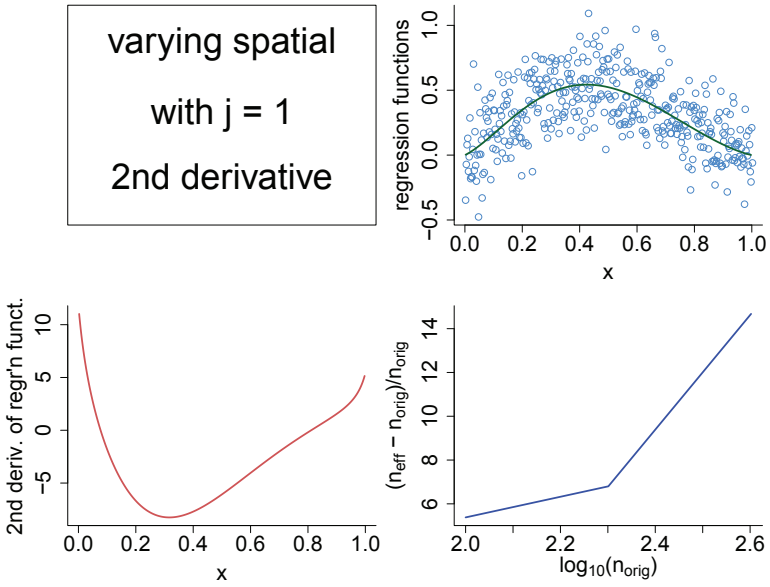


Figure 5.14: *Second derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 1$. *Top right panel shows the regression function*$(n = 400)$. *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Note the large amount of required effective samples.*

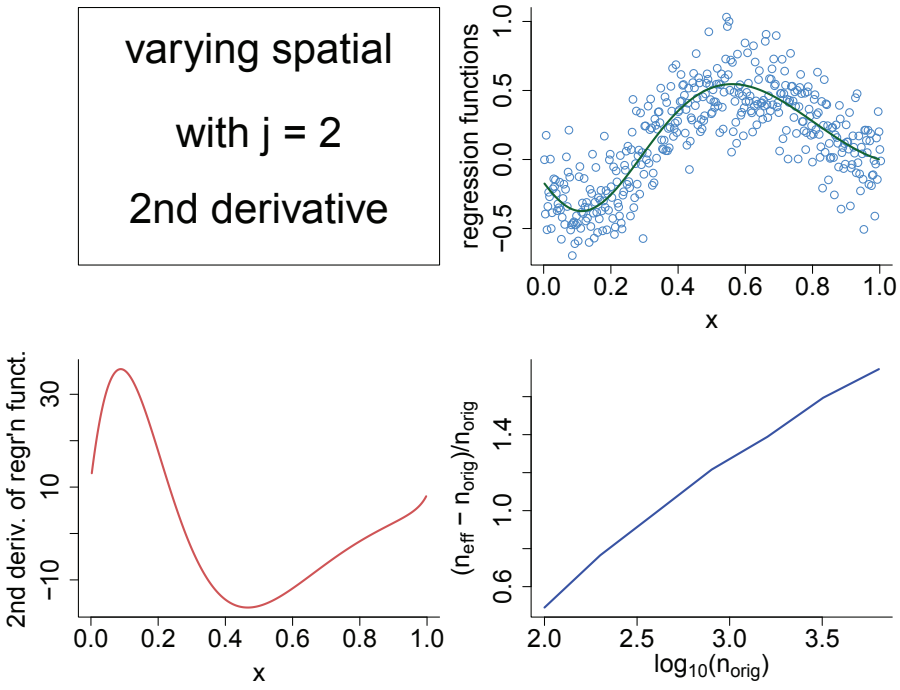


Figure 5.15: *Second derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 2$. *Top right panel shows the regression function*$(n = 400)$. *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. Note the large amount of required effective samples.*

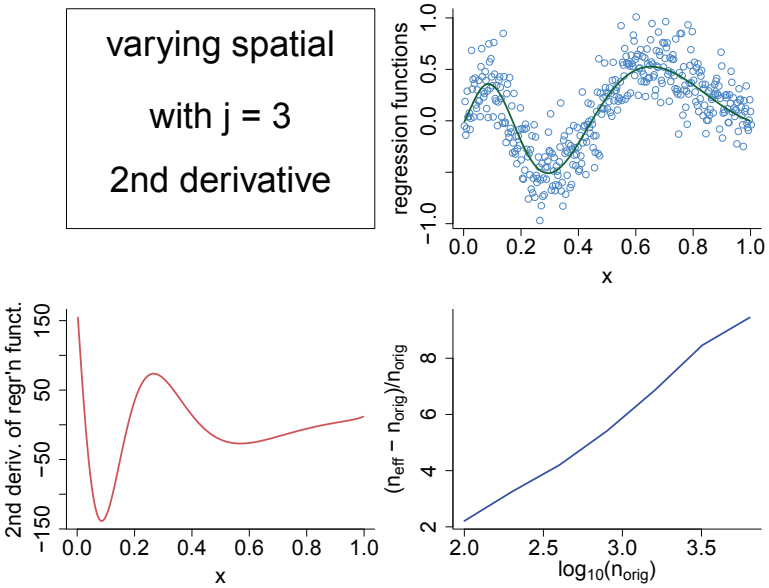


Figure 5.16: *Second derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 3$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size.*

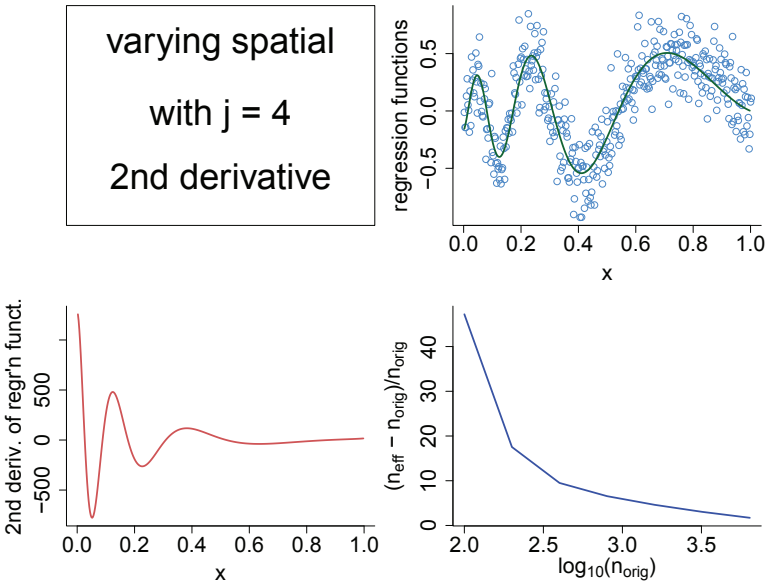


Figure 5.17: *Second derivative of varying spatial (shape) data from Wand (2000) with parameter* $j = 4$. *Top right panel shows the regression function*($n = 400$). *Bottom left panel shows the second derivative of the regression function. Bottom right shows the effective sample size against* $\log_{10}$ *of starting sample size. There are more features to model in the second derivative regression function.*

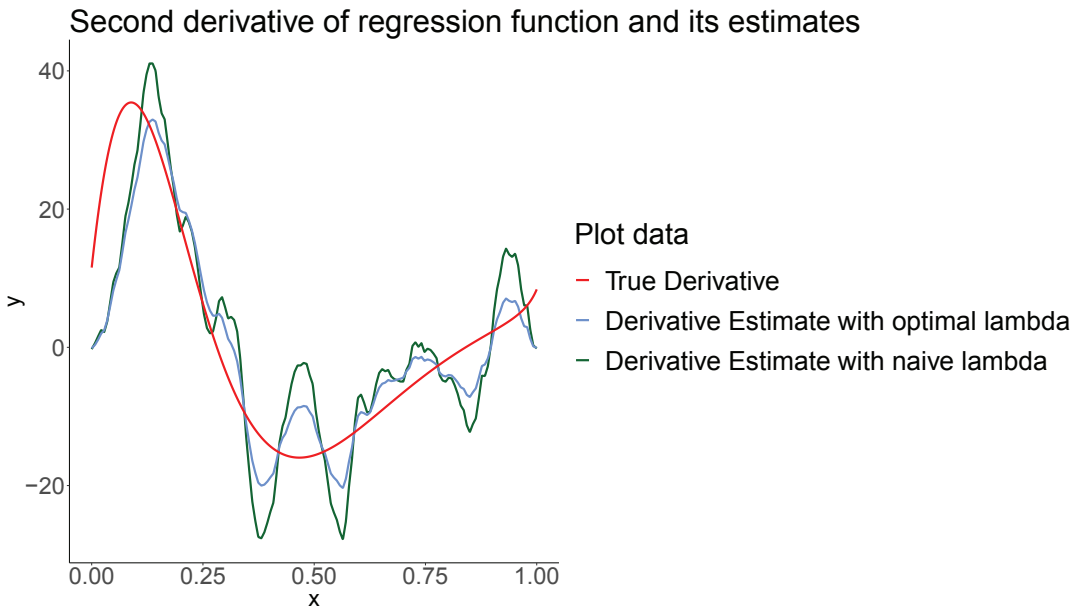


Figure 5.18: *Modelling 2nd derivative of varying shape data from Wand (2000), parameter $j = 2$ using optimal and naive smoothing parameter. The fit with optimal smoothing parameter result in an un-smooth fit.*

Figure 5.14, the proportion is reaching 14. To investigate this, we plotted the second derivative modelling spatial data when $j = 2$ in Figure 5.18. We can see that even using the optimal smoothing parameter resulted in a very wiggly fit to the second derivative. What this means is that to capture simple second derivative cases, we would need very large sample sizes using the naïve smoothing parameter.

5.6 Discussion

We can see from the results of simulations presented in Section 5.5 that in many cases it may not be feasible to model the derivatives of the regression functions $f^{(1)}$ and $f^{(2)}$ using the the smoothing parameter of the data $\lambda^{(0)}_{\text{MASE}}$. For example, to model the first derivative of the data varied by noise, we may need more than 10% of number of observations. In studies where obtaining observations is expensive or impossible, such rare outcomes, the extra observations may not be feasible to obtain. In the case of the noise variation, data, modelling of the second derivative requires even more observations. Similar argument can also apply to the spatial variation data. For their first derivatives, the worse case scenarios required extra 10% of number of observations. Modelling the second derivative of data with spatial variation, or shape variation, also requires large number of observations. The noise and shape variations data

used for the simulations can feature in child growth data. Data collections in child growth studies are imperfect, resulting in measurement errors, data can be missing, and not all data may be collected. In addition, children can have different growth pattern, some may suffer stunting and recover, while others may not.

It is possible to forgo using the model based approach to estimate the first derivative $f^{(1)}$ and t the second derivative $f^{(2)}$ by numerically calculating the gradient between data points of $f^{(0)}$. However, such approach can become computationally time consuming, especially if there are many subjects in a study. The ideal situation is to be able to select an optimal smoothing parameter for the derivatives. As this chapter demonstrates using the smoothing parameter of the regression function to model the derivative is not ideal. As a result, a future work is a model based procedure to estimate the optimal smoothing parameter for the derivative.

Appendix 5.A Representing $\hat{f}$

A summary of how the fitted function is a *linear function* of the data $\mathbf{y}$ can be found at Ruppert, Wand, and Carroll (2003a, Section 3.10, p76). The background information actually starts from liner regression Ruppert, Wand, and Carroll (2003a, Section 2.3, pp20-21). In particular, in linear regression the vector of estimated parameters is

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \ ,$$

where $\mathbf{X}$ is the *design matrix*. The *fitted* function,

$$\mathbf{y} = \mathbf{X}\widehat{\boldsymbol{\beta}} + \boldsymbol{\varepsilon}$$

then can be rewritten as

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{X}\,(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \\ &= \mathbf{H}\mathbf{y} \ , \end{aligned}$$

where

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \ .$$

The matrix $\mathbf{H}$ is known as the *hat matrix* because multiplying it by the data $\mathbf{y}$ results in fitted values $\hat{\mathbf{y}}$ (Ruppert, Wand, and Carroll, 2003a, Sec 2.3). Similarly for penalized splines Ruppert, Wand, and Carroll (2003a, Section 3.5) has the vector of estimated coefficients, for a particular smoothing parameter λ as

$$\hat{\boldsymbol{\beta}}_\lambda = (\mathbf{C}^\top \mathbf{C} + \lambda \mathbf{D})^{-1} \mathbf{C}^\top \mathbf{y} \ ,$$

where $\boldsymbol{C}$ and $\boldsymbol{D}$ are defined in Section 5.3. So in Ruppert, Wand, and Carroll (2003a, Section 3.10), the penalized spline model is generalized to

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{C}\,\hat{\boldsymbol{\beta}}_\lambda \\ &= \mathbf{C}\,(\mathbf{C}^\top \mathbf{C} + \lambda \mathbf{D})^{-1} \mathbf{C}^\top \mathbf{y} \\ &= \mathbf{S}_\lambda\, \mathbf{y} \ , \end{aligned}$$

where the *smoother matrix* is

$$\mathbf{S}_\lambda = \mathbf{C}(\mathbf{C}^\top \mathbf{C} + \lambda \mathbf{D})^{-1} \mathbf{C}^\top \ .$$

As a result, (5.3) can be written as

$$\begin{bmatrix} \widehat{f}(x_1;\lambda) \\ \widehat{f}(x_2;\lambda) \\ \vdots \\ \widehat{f}(x_n;\lambda) \end{bmatrix} = \boldsymbol{S}_\lambda \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} ,$$

where the dimension of $\boldsymbol{S}_\lambda$ is 2×2.

Appendix 5.B MASE expression for estimation of $f^{(r)}$

If we apply the concept of the *hat matrix* (Ruppert, Wand, and Carroll, 2003a, p21) from linear regression, which converts $\boldsymbol{y}$ to $\widehat{\boldsymbol{y}}$ to penalised splines, we have the smoother matrix such as (5.A). So the conversion becomes

$$\widehat{f_\lambda} = \boldsymbol{C}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \boldsymbol{y} \ .$$

Therefore, for the general derivative estimate becomes

$$\widehat{f_\lambda^{(r)}} = \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \boldsymbol{y} \ .$$

MASE can be expressed as the squared bias and variance components. We begin by calculating the bias term. The bias is

$$\mathrm{E}(\widehat{f_\lambda^{(r)}} - f_\lambda^{(r)}) = \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top f_\lambda - f_\lambda^{(r)}$$

The sum of squared bias contribution is then

$$\frac{1}{n} \| \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top f_\lambda - f_\lambda^{(r)} \|^2$$

For the calculation of the covariance term

$$\begin{aligned} \mathrm{Cov}(f_\lambda^{(r)}) &= \mathrm{Cov}[\ \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \boldsymbol{y}\ ] \\ &= \sigma^2 \{ \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \} \times \{ \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \}^\top \\ &= \sigma^2 \, \boldsymbol{C}^{(r)}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} \boldsymbol{C}^\top \, \boldsymbol{C}(\boldsymbol{C}^\top \boldsymbol{C} + \lambda \boldsymbol{D})^{-1} {\boldsymbol{C}^{(r)}}^\top \end{aligned}$$

$$\begin{aligned}\operatorname{tr}[\operatorname{Cov}(f_\lambda^{(r)})] &= \sigma^2 \operatorname{tr}[\boldsymbol{C}^{(r)}(\boldsymbol{C}^\top\boldsymbol{C} + \lambda\boldsymbol{D})^{-1}\boldsymbol{C}^\top\boldsymbol{C}(\boldsymbol{C}^\top\boldsymbol{C} + \lambda\boldsymbol{D})^{-1}{\boldsymbol{C}^{(r)}}^\top\boldsymbol{C}^{(r)}] \\ &= \sigma^2 \operatorname{tr}[\boldsymbol{M}\,\boldsymbol{M}^{(r)}] \,,\end{aligned}$$

where

$$\boldsymbol{M} = (\boldsymbol{C}^\top\boldsymbol{C} + \lambda\boldsymbol{D})^{-1}\boldsymbol{C}^\top\boldsymbol{C} \,,$$

and

$$\boldsymbol{M}^{(r)} = (\boldsymbol{C}^\top\boldsymbol{C} + \lambda\boldsymbol{D})^{-1}{\boldsymbol{C}^{(r)}}^\top\boldsymbol{C}^{(r)} \,.$$

The contribution from the sum of variances is

$$\sigma^2 \operatorname{tr}[\boldsymbol{M}^{(r)}\,\boldsymbol{M}]$$

6 Conclusion

This thesis outlines a number of ways to perform statistical analysis on distributed computing systems by applying many existing statistical methods in new context.

We begin with a scenario whereby the size of the data becomes a problem during the analysis. Specifically when the data is too large to fit into the memory of a single computer. We first outline a number of techniques to resolve this issue within the single computer context. When the single computer solution become inadequate, due to every increasing data size, we explore the use of distributed computing systems, which divide up large data into many smaller and manageable data (called shared, subsets or blocks). These shards are then stored across many computers. By dividing and storing data this way, distributed computing systems break the implicit assumption made by many statistical algorithms, that all the data are stored in one location, typically the memory of a single computer.

To analyse distributed data, we first introduce the idea of divide and recombine and how it can be implemented using programming abstraction such as MapReduce in modern distributed computing system such as Hadoop and Spark. Divide and recombine is a two step process where we first perform some analysis on each shard of distributed data. In the second step, we combine the results from these analyses. We outline the weighted average method as a mean of performing this recombination. We explain modern distributed computing systems such as Hadoop and Spark, and how MapReduce is a powerful abstraction that allows divide and recombine to applied on distributed computing systems. Our application of divide and recombine regression to a large dataset demonstrate the scalability of distributed computing systems. In addition, we show that even in a distributed computing setting, we gain computational performance if we can reduce the amount of data to process.

Chapter 3 shows another interpretation of divide and recombine, whereby the divide stage consists of coarsening and aggregation operations to reduce the size of the data. The recombine stage then consists of fitting regression model using this compressed dataset. An advantage of this approach is that by reducing a big dataset, we can fit our regression model on a single standard desktop computer. As well, being able perform statistical analysis on a single computer also means we can use many existing software packages and algorithms. To account for the bias resulting from the coarsening and aggregation process, we use the EM algorithm to recover the complete data model. If all the non-coarsened covariates are also categorical, the data can be reduced from individual observations to counts of the number of unique combination of covariate levels. A follow-on effect is that the regression model than only need to handle covariates at different levels. In a simulation study, we find that coarsening and using EM algorithm can approximate the complete data model, although our method

is sensitive to model misspecification.

In Chapter 4, we explore another existing statistical technique that is similar to performing divide and recombine, namely meta-analysis. Since traditional meta-analysis involves combining results from many studies into meta-analysis, we can view distributed data as data from different studies. To take into account the varying covariate structure that can exist across studies, we use propensity score as a method of covariate adjustment. We apply our work to child growth data from Bill and Melinda Gates Foundation whereby the studies for meta-analysis has varying covariate structures. We demonstrate that by considering shards of distributed data as studies in a meta-analysis, we can apply meta-analysis technique in distributed computing settings as a form of divide and recombine.

Finally in Chapter 5 we explore an issue related to Chapter 4, whereby we need to model the *derivatives* of child growth curves for a large number of children using smoothing splines method. In previous chapters, we assumed that the procedures to estimate quantities such as regression coefficients are well known and these procedures can also quantify the uncertainties of the estimates, e.g., standard errors. As a result, we can focus our attention on how to combine these estimates. However, in this chapter, we are uncertain whether one of the key parameters is optimal when modelling the derivatives of regression functions. One of the key parameters in smoothing spline method is the smoothing parameter. Although there are well established methods for selecting the smoothing parameter for non-derivative data, there are no such methods for the derivatives. We examine the effect of naïvely using the smoothing parameter for the non-derivative to model the derivatives, by exploring the extra number of samples that would be required. From our simulation using data functions with varying characteristics, we find that with certain characteristics of the underlying function and at particular derivatives, there are advantages to using an optimal smoothing parameter for the derivatives. As a result, when modelling the derivatives of a regression function with a large data set, we should be careful in naïve application of the smoothing parameter.

We show in this thesis how statistical analysis on distribute data of very large data is possible through different ways of implementing the divide and recombine paradigm. We explore the various ways in which existing statistical methods such as coarsening, the EM algorithm, and meta-analysis still are relevant in the context of distributed data in distributed systems. This thesis alludes to many areas of opportunities for future work. For Chapter 3, involving coarsening and EM algorithm, future work include application of the method with more covariates in the regression model, examining the sensitivity of the sub-sample. For Chapter 4, we treat the data from child growth studies as distributed data and showed how inference can be performed using technique such as meta-analysis. A possible future work is to use a more obvious form of distributed data, such as data from mobile phones, in a scenario where it

is not possible to perform the analysis when all the data are combined. The combination of multiple imputation, propensity score estimation also is a future research area. In particular investigating the effect of different sequence of operation (e.g., impute then estimate propensity score) in different contexts. For Chapter 5, an obvious future work is develop method to estimate the optimal smoothing parameter for the derivatives of regression functions. Follow on from that is whether parameters such as smoothing parameters can be combined in a divide and recombine setting.

References

Adler, D., Gläser, C., Nenadic, O., Oehlschlägel, J., and Zucchini, W., 2014. *ff: memory-efficient storage of large data on disk and fast access functions* **online**. Available from: `https://cran.r-project.org/package=ff`.

Albert, A. and Anderson, J.A., 1984. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1) (), pp.1–10.

Anderson, C., Hafen, R., Sofrygin, O., Ryan, L., and HBGDki Community, members of the, 2018. Comparing predictive abilities of longitudinal child growth models. *Statistics in Medicine*.

Atienza, N., García-Heras, J., Muñoz-Pichardo, J.M., and Villa, R., 2008. An application of mixture distributions in modelization of length of hospital stay. *Statistics in Medicine*, 27(9), pp.1403–1420.

Austin, P.C., 2011. An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate Behavioral Research*, 46(3). PMID: 21818162, pp.399–424.

Becker, B.J., Wu, M.-J., et al., 2007. The synthesis of regression slopes in meta-analysis. *Statistical science*, 22(3), pp.414–429.

Black, P.E. and Pieterse, V., 2004. *Key*. Available from: `http://xlinux.nist.gov/dads//HTML/key.html` [visited on Sept. 3, 2015].

Brumback, B.A., Cook, R.J., and Ryan, L.M., 2000. A meta-analysis of case-control and cohort studies with interval-censored exposure data: application to chorionic villus sampling. *Biostatistics*, 1(2), pp.203–217.

Burgess, S., White, I.R., Resche-Rigon, M., and Wood, A.M., 2013. Combining multiple imputation and meta-analysis with individual participant data. *Statistics in medicine*, 32(26), pp.4499–4514.

Campbell, T. and Broderick, T., 2017. Automated scalable bayesian inference via hilbert coresets. *arXiv preprint arXiv:1710.05053*.

Casella, G. and Berger, R.L., 2002. *Statistical inference*. Vol. 2. Duxbury Pacific Grove, CA.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R.E., 2006. Bigtable: A distributed storage system for structured data. *7th symposium on operating systems design and implementation (osdi '06), november 6-8, seattle, wa, usa*, pp.205–218.

Charnigo, R., Hall, B., and Srinivasan, C., 2011. A generalized C_p criterion for derivative estimation. *Technometrics*, 53(3), pp.238–253.

Chen, J., Monga, R., Bengio, S., and Jozefowicz, R., 2016. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*.

Cleveland, W.S. and Hafen, R., 2014. Divide and recombine (D&R): Data science for large complex data. *Statistical Analysis and Data Mining*, 7(6), pp.425–433.

Cloudera Inc., n.d. *Cloudera* **online**. Available from: `http://www.cloudera.com/` [visited on Sept. 7, 2015].

Cole, T.J., Donaldson, M.D., and Ben-Shlomo, Y., 2010. SITAR–a useful instrument for growth curve analysis. *International journal of epidemiology*, 39(6), pp.1558–1566.

Cormode, G., Garofalakis, M., Haas, P.J., and Jermaine, C., 2011. *Synopses for massive data: samples, histograms, wavelets, sketches*. now.

Cormode, G., 2011. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases*, 4(1-3), pp.1–294.

Cox, D. and Snell, E., 1989. *Analysis of binary data. 2nd edition*. 2nd ed., Monographs on statistics and applied probability. London: Chapman and Hall.

Coyle, P., 2015. Interview with a Data Scientist (Hadley Wickham). *Models are illuminating and wrong*.

Craven, P. and Wahba, G., 1978. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31(4) (), pp.377–403.

Damesa, T.M., Möhring, J., Worku, M., and Piepho, H.-P., 2017. One step at a time: stage-wise analysis of a series of experiments. *Agronomy Journal*, 109(3), pp.845–857.

Das, U., Maiti, T., and Pradhan, V., 2010. Bias correction in logistic regression with missing categorical covariates. *Journal of Statistical Planning and Inference*, 140(9), pp.2478–2485.

De Boor, C., De Boor, C., Mathématicien, E.-U., De Boor, C., and De Boor, C., 1978. *A practical guide to splines*. Vol. 27. Springer-Verlag New York.

Dean, B.Y.J. and Ghemawat, S., 2010. MapReduce: a flexible data processing tool. *Communications of the ACM*, 53, pp.72–77.

Dean, J and Ghemawat, S, 2008. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), pp.1–13.

Dean, J. and Ghemawat, S., 2004. MapReduce: Simplied Data Processing on Large Clusters. *Proceedings of 6th symposium on operating systems design and implementation*, pp.137–149.

Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M.A., Senior, A., Tucker, P., Yang, K., and Ng, A.Y., 2012. Large Scale Distributed Deep Networks. *NIPS 2012: Neural Information Processing Systems*, pp.1–11.

Debray, T.P.A., Moons, K.G.M., Valkenhoef, G. van, Efthimiou, O., Hummel, N., Groenwold, R.H.H., and Reitsma, J.B., 2015. Get Real in Individual Participant Data (IPD) Meta-Analysis: A Review of the Methodology. *Research Synthesis Methods*, 6(4), pp.293–309.

Dempster, A.P., Laird, N.M., and Rubin, D.B., 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pp.1–38.

Dobson, A.J. and Barnett, A., 2008. *An introduction to generalized linear models*. Padstow, Cornwall. Great Britain.: CRC press.

Donoho, D., 2017. 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4), pp.745–766.

Drake, C., 1993. Effects of misspecification of the propensity score on estimators of treatment effect. *Biometrics*, pp.1231–1236.

Dropbox, n.d. *How dropbox keeps your files secure* **online**. Available from: `https://www.dropbox.com/help/sign-in/how-security-works` [visited on Jan. 4, 2010].

Ebrahim, G.J., 2010. Who child growth standards. growth velocity based on weight, length and head circumference.methods and development. *Journal of Tropical Pediatrics*, 56(2), pp.136–136.

Elze, M.C., Gregson, J., Baber, U., Williamson, E., Sartori, S., Mehran, R., Nichols, M., Stone, G.W., and Pocock, S.J., 2017. Comparison of propensity score methods and covariate adjustment: evaluation in 4 cardiovascular studies. *Journal of the American College of Cardiology*, 69(3), pp.345–357.

Firth, D., 1993. Bias reduction of maximum likelihood estimates. *Biometrika*, 80(1), pp.27–38.

Fitzmaurice, G.M., Laird, N.M., and Ware, J.H., 2011. *Applied longitudinal analysis*.

García, S., Luengo, J., and Herrera, F., 2016. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, 98, pp.1–29.

Gelman, A. and Hill, J., 2006. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press.

Gentle, J.E., 2009. *Computational Statistics*, Statistics and Computing. New York, NY: Springer New York.

Ghemawat, S., Gobioff, H., and Leung, S.-T., 2003. *The Google file system*.

Gottfrid, D., 2007. *Self-Service, Prorated Supercomputing Fun!* Available from: `http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/`.

Guha, S., Hafen, R., Rounds, J., Xia, J., Li, J., Xi, B., and Cleveland, W.S., 2012. Large complex data: divide and recombine (D&R) with RHIPE. *Stat*, 1(1), pp.53–67.

Hafen, R., 2016. Divide and Recombine: Approach for Detailed Analysis and Visualization of Large Complex Data. In: Bühlmann, P., Drineas, P., Kane, M., and Lann, M. van der eds. *Handbook of big data*. Chap. 3.

Härdle, W., Hall, P., and Marron, J.S., 1988. How far are automatically chosen regression smoothing parameters from their optimum? *Journal of the American Statistical Association*, 83(401), pp.86–95.

Harezlak, J., Ruppert, D., and Wand, M.P., 2018. *Semiparametric Regression with R*. New York: Springer.

Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The elements of statistical learning*. 2nd ed. New York: Springer-Verlag New York.

He, B., Fang, W., Luo, Q., Govindaraju, N.K., and Wang, T., 2008. Mars: a MapReduce framework on graphics processors. *International conference on parallel architectures and compilation techniques* **online** pp.260–269. Available from: http://dl.acm.org/citation.cfm?id=1454152.

Heinze, G. and Schemper, M., 2002. A solution to the problem of separation in logistic regression. *Statistics in Medicine*, 21(16), pp.2409–2419.

Heitjan, D.F. and Rubin, D.B., 1991. Ignorability and coarse data. *Ann. Statist.* 19(4) (), pp.2244–2253.

Henry, L. and Wickham, H., 2018. *Purrr: functional programming tools* **online**. R package version 0.2.5. Available from: https://CRAN.R-project.org/package=purrr.

Holst, U., Hössjer, O., Björklund, C., Ragnarson, P., and Edner, H., 1996. Locally weighted least squares kernel regression and statistical evaluation of lidar measurements. *Environmetrics*, 7(4), pp.401–416.

Hortonworks Inc. *Horton Works* **online**. Available from: http://hortonworks.com [visited on Sept. 7, 2015].

Hsiao, C., 1983. Regression analysis with a categorized explanatory variable. *Studies in Econometrics, Time Series, and Multivariate Statistics*, pp.93–129.

Huggins, J., Adams, R.P., and Broderick, T., 2017. Pass-glm: polynomial approximate sufficient statistics for scalable bayesian glm inference. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. eds. *Advances in neural information processing systems 30* **online**Curran Associates, Inc., pp.3611–3621. Available from: http://papers.nips.cc/paper/6952-pass-glm-polynomial-approximate-sufficient-statistics-for-scalable-bayesian-glm-inference.pdf.

Huggins, J., Campbell, T., and Broderick, T., 2016. Coresets for scalable bayesian logistic regression. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., and Garnett, R. eds. *Advances in neural information processing systems 29* **online**Curran Associates, Inc., pp.4080–4088. Available from: http://papers.nips.cc/paper/6486-coresets-for-scalable-bayesian-logistic-regression.pdf.

Ibrahim, J.G., 1990. Incomplete data in generalized linear models. *Journal of the American Statistical Association*, 85(411), pp.765–769.

Ibrahim, J.G., Chen, M.-H., and Lipsitz, S.R., 1999. Monte carlo em for missing covariates in parametric regression models. *Biometrics*, 55(2), pp.591–596.

Imai, K. and Van Dyk, D.A., 2004. Causal inference with general treatment regimes: Generalizing the propensity score. *Journal of the American Statistical Association*, 99(467), pp.854–866.

Jackson, D. and White, I.R., 2018. When should meta-analysis avoid making hidden normality assumptions? *Biometrical Journal*, 60(6), pp.1040–1058.

Jeffreys, H., 1946. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 186(1007), pp.453–461.

Jumbe, N.L., Murray, J.C., and Kern, S., 2016. Data sharing and inductive learning toward healthy birth, growth, and development. *New England Journal of Medicine*, 374(25), pp.2415–2417.

Karau, H., Konwinski, A., Wendell, P., and Zaharia, M., 2015. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Inc, p.276.

Karmel, P. and Polasek, M., 1970. *Applied statistics for economists*.

Kiefer, J., Wolfowitz, J., et al., 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3), pp.462–466.

Kim, J.K., 2011. Parametric fractional imputation for missing data analysis. *Biometrika*, 98(1), pp.119–132.

Kim, J.K. and Hong, M., 2012. Imputation for statistical inference with coarse data. *Canadian Journal of Statistics-revue Canadienne De Statistique*, 40(3), pp.604–618.

Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M.I., 2014. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(4), pp.795–816.

Kowalski, A.J., Georgiadis, A., Behrman, J.R., Crookston, B.T., Fernald, L.C.H., and Stein, A.D., 2018. Linear growth through 12 years is weakly but consistently associated with language and math achievement scores at age 12 years in 4 low- or middle-income countries. *Journal of Nutrition*, 148(11), pp.1852–1859.

Lee, C.Y.Y. and Wand, M.P., 2016. Variational methods for fitting complex Bayesian mixed effects models to health data. *Statistics in Medicine*, 35(2), pp.165–188.

Lee, J.Y.L., Brown, J.J., and Ryan, L.M., 2017. Sufficiency revisited: rethinking statistical algorithms in the big data era. *The American Statistician*, 71(3), pp.202–208.

Lee, K.J. and Simpson, J.A., 2013. Introduction to multiple imputation for dealing with missing data. *Respirology*, 19(2), pp.162–167.

Lee, K.J. and Thompson, S.G., 2008. Flexible parametric models for random-effects distributions. *Statistics in Medicine*, 27(3), pp.418–434.

Lipsitz, S., Parzen, M., Natarajan, S., Ibrahim, J., and Fitzmaurice, G., 2004. Generalized linear models with a coarsened covariate. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 53(2), pp.279–292.

Liu, D., Liu, R.Y., and Xie, M., 2015. Multivariate meta-analysis of heterogeneous studies using only summary statistics: efficiency and robustness. *Journal of the American Statistical Association*, 110(509), pp.326–340.

Liu, Z., Jiang, B., and Heer, J., 2013. imMens : Real-time Visual Querying of Big Data. *Computer Graphics Forum*, 32(3pt4) (), pp.421–430.

Louis, T.A., 1982. Finding the observed information matrix when using the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp.226–233.

Lumley, T., 2013. *biglm: bounded memory linear and generalized linear models Title* **online**. Available from: `https://cran.r-project.org/package=biglm`.

Luraschi, J., Kuo, K., Ushey, K., Allaire, J., and The Apache Software Foundation, 2019. *sparklyr: R Interface to Apache Spark* **online**. R package version 0.9.4. Available from: `https://CRAN.R-project.org/package=sparklyr`.

MapR Technologies Inc. *MapR* **online**. Available from: `https://www.mapr.com` [visited on Sept. 7, 2015].

Marazzi, A., Paccaud, F., Ruffieux, C., and Beguin, C., 1998. Fitting the distributions of length of stay by parametric models. *Medical care*, 36(6), pp.915–927.

Marron, J.S. and Wand, M.P., 1992. Exact mean integrated squared error. *The Annals of Statistics*, 20(2) (), pp.712–736.

McKenzie, J.E., Beller, E.M., and Forbes, A.B., 2016. Introduction to systematic reviews and meta-analysis. *Respirology*, 21(4), pp.626–637.

McMahan, H.B., Moore, E., Ramage, D., Hampson, S., and Arcas, B.A. y, 2017. Communication-efficient learning of deep networks from decentralized data. *International conference on artificial intelligence and statistics*, pp.1273–1282.

Mehta, C.R. and Patel, N.R., 1995. Exact logistic regression: theory and examples. *Statistics in Medicine*, 14(19), pp.2143–2160.

Meilijson, I., 1989. A fast improvement to the em algorithm on its own terms. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(1), pp.127–138.

Mitra, R. and Reiter, J.P., 2016. A comparison of two methods of estimating propensity scores after multiple imputation. *Statistical Methods in Medical Research*, 25(1). PMID: 22687877, pp.188–204.

Nickolls, J., Buck, I., Garland, M., and Skadron, K., 2008. Scalable parallel programming with cuda. *Acm siggraph 2008 classes on*. Vol. 6, 2, pp.40–53.

Normand, S.-L.T., 1999. Meta-analysis: formulating, evaluating, combining, and reporting. *Statistics in Medicine*, 18(3), pp.321–359.

Onis, M. de, Onyango, A., Borghi, E., Siyam, A., Pinol, A., Garza, C., Martines, J., Martorell, R., Victora, C.G., Bhan, M.K., Araújo, C.L., Lartey, A., Owusu, W.B., Bhandari, N., Norum, K.R., Bjoerneboe, G.-E.A., Mohamed, A.J., Dewey, K.G., Belbase, K., Black, M., Chumlea, W., Cole, T., Frongillo, E., Grummer-Strawn, L., Shrimpton, R., Broeck, J.V. den, Pan, H., Rigby, R., Stasinopoulos, M., Buuren, S. van, Albernaz, E., Tomasi, E., Silveira, R. de Cássia Fossati da, Nader, G., Sagoe-Moses, I., Gomez, V., Sagoe-Moses, C., Taneja, S., Rongsen, T., Chetia, J., Sharma, P., Bahl, R., Baerug, A., Tufte, E., Rudvin, K., Nysaether, H., Alasfoor, D., Prakash, N.S., Mabry, R.M., Rajab, H.J.A., Helmi, S.A., Nommsen-Rivers, L.A., Cohen, R.J., and Heinig, M.J., 2006. Who child growth standards: length/height-for-age, weight-for-age, weight-for-length, weight-for-height and body mass index-for-age - methods and development. *(2006)*, p.312.

Orchard, T. and Woodbury, M.A., 1972. A missing information principle: theory and applications. *Proceedings of the sixth berkeley symposium on mathematical statistics and probability, volume 1: theory of statistics* **online**Berkeley, Calif.: University of California Press, pp.697–715. Available from: `https://projecteuclid.org/euclid.bsmsp/1200514117`.

Ormerod, J.T. and Wand, M.P., 2010. Explaining Variational Approximations. *The American Statistician*, 64(2), pp.140–153.

Pearl, J., Glymour, M., and Jewell, N.P., 2016. *Causal inference in statistics: a primer*. John Wiley & Sons.

Piepho, H.-P., Möhring, Schulz-Streeck, T., and Ogutu, J.O., 2012. A stage-wise approach for the analysis of multi-environment trials. *Biometrical Journal*, 54(6), pp.844–860.

Quartagno, M and Carpenter, J., 2016. Multiple imputation for ipd meta-analysis: allowing for heterogeneity and studies with missing covariates. *Statistics in medicine*, 35(17), pp.2938–2954.

R Core Team, 2017. *R: a language and environment for statistical computing* **online**. R Foundation for Statistical Computing. Vienna, Austria. Available from: `https://www.R-project.org/`.

Robbins, H. and Monro, S., 1951. A stochastic approximation method. Vol. 22, 3, pp.400–407.

Robert, C. and Casella, G., 2010. *Introducing monte carlo methods with r*. New York: Springer-Verlag.

Rosenbaum, P.R. and Rubin, D.B., 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1), pp.41–55.

Rotem-Gal-Oz, Arnon, n.d. *Fallacies of Distributed Computing Explained* **online**. Available from: `http://www.rgoarchitects.com/Files/fallacies.pdf` [visited on Mar. 28, 2019].

Rubin, D.B., 1987. *Multiple imputation for nonresponse in surveys*. John Wiley & Sons.

Rubinstein, R.Y. and Kroese, D.P., 2017. *Simulation and the monte carlo method*. 3rd ed. Hoboken, New Jersey: Wiley.

Ruppert, D., Wand, M.P., and Carroll, R.J., 2003a. *Semiparametric regression*. New York: Cambridge University Press.

Ruppert, D., Wand, M.P., and Carroll, R.J., 2003b. *Semiparametric regression. cambridge series in statistical and probabilistic mathematics 12*.

Ryan, L., 2008. Combining data from multiple sources, with applications to environmental risk assessment. *Statistics in Medicine*, 27(5) (), pp.698–710.

SAS Institute Inc, n.d. *SAS and Hadoop* **online**. Available from: `http://www.sas.com/en_au/insights/big-data/hadoop.html` [visited on Nov. 9, 2016].

Schafer, J.L., 1997. *Analysis of incomplete multivariate data*. Chapman and Hall/CRC.

Scheidegger, C., 2016. Interactive Visual Analysis of Big Data. In: Bühlmann, P., Drineas, P., Kane, M., and Lann, M. van der eds. *Handbook of big data*. Chap. 5.

Scott, S. and Blocker, A., 2013. Bayes and big data: The consensus monte carlo algorithm. *Bayes*, 250, pp.1–22.

Scott, S.L., Blocker, A.W., Bonassi, F.V., Chipman, H.A., George, E.I., and McCulloch, R.E., 2016. Bayes and big data: the consensus monte carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2), pp.78–88.

Searle, S., Casella, G, and McCulloch, C.E., 1992. *Variance components*, Wiley series in probability and mathematical statistics. New York: John Wiley & Sons.

Sigrist, M.W., Winefordner, J.D., Kolthoff, I., et al., 1994. *Air monitoring by spectroscopic techniques*. Ed. by M.W. Sigrist. Vol. 127. New York: John Wiley & Sons.

Simpkin, A.J., Durban, M., Lawlor, D.A., MacDonald-Wallis, C., May, M.T., Metcalfe, C., and Tilling, K., 2018. Derivative estimation for longitudinal data analysis: examining features of blood pressure measured repeatedly during pregnancy. *Statistics in medicine*, 37(19), pp.2836–2854.

Simpkin, A. and Newell, J., 2013. An additive penalty p-spline approach to derivative estimation. *Computational Statistics & Data Analysis*, 68, pp.30–43.

Smith, D., 2009. *Using R as a scripting language with Rscript* **online**. Available from: `http://blog.revolutionanalytics.com/2009/01/using-r-as-a-scripting-language-with-rscript.html` [visited on Aug. 15, 2016].

Somers, J., 2018. Binary stars. *New Yorker*, 94(40), pp.28 –35.

Sullivan, F. and Dongarra, J., 2000. Guest editors' introduction: the top 10 algorithms. *Computing in Science & Engineering*, 2 (), pp.22–23.

Sutton, A.J. and Higgins, J.P., 2008. Recent developments in meta-analysis. *Statistics in medicine*, 27(5), pp.625–650.

The Apache Software Foundation, n.d.(a). *Apache Hadoop project* **online**. Available from: `https://hadoop.apache.org/` [visited on Sept. 3, 2015].

The Apache Software Foundation, n.d.(b). *Apache Hive* **online**. Available from: `https://hive.apache.org`.

The Apache Software Foundation, n.d.(c). *Apache Pig* **online**. Available from: `https://pig.apache.org`.

The Apache Software Foundation, n.d.(d). *Apache Spark* **online**. Available from: `https://spark.apache.org` [visited on Oct. 26, 2015].

The Apache Software Foundation, n.d.(e). *HDFS Architecture* **online**. Available from: `http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html` [visited on Sept. 9, 2015].

The Apache Software Foundation, n.d.(f). *Hadoop Streaming* **online**. Available from: `http://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html` [visited on Aug. 25, 2016].

The City of New York. *New York City's Taxi and Limousine Commission (TLC) Trip Record Data* **online**. [Online; accessed 17-February-2017]. Available from: `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml` [visited on Feb. 17, 2017].

The Department of Statistics, P.U., n.d. *DeltaRho* **online**. Available from: `http://deltarho.org` [visited on Dec. 15, 2016].

The MathWorks, Inc., n.d. *MATLAB MapReduce and Hadoop* **online**. Available from: `http://au.mathworks.com/discovery/matlab-mapreduce-hadoop.html` [visited on Sept. 8, 2015].

The MathWorks, I., n.d. *Tall arrays* **online**. Available from: `https://www.mathworks.com/help/matlab/import_export/tall-arrays.html` [visited on Jan. 5, 2010].

Van Buuren, S., Brand, J.P., Groothuis-Oudshoorn, C.G., and Rubin, D.B., 2006. Fully conditional specification in multivariate imputation. *Journal of statistical computation and simulation*, 76(12), pp.1049–1064.

Vansteelandt, S. and Daniel, R.M., 2014. On regression adjustment for the propensity score. *Statistics in medicine*, 33(23), pp.4053–4072.

Venables, W. and Ripley, B., 2002. *Modern applied statistics with s*. 4th ed. Springer.

Viechtbauer, W., 2010. Conducting meta-analyses in R with the metafor package. *Journal of statistical software*, 36(3).

Wand, M.P, 1999. On the optimal amount of smoothing in penalised spline regression. *Biometrika*, 86(4) (), pp.936–940.

Wand, M.P., 2000. A comparison of regression spline smoothing procedures. *Computational Statistics*, 15(4), pp.443–462.

Wand, M.P. and Ormerod, J.T., 2008. On semiparametric regression with o'sullivan penalized splines. *Australian & New Zealand Journal of Statistics*, 50(2), pp.179–198.

Wasserman, L., 2004. *All of statistics: a concise course in statistical inference*. New York: Springer.

Wei, G.C. and Tanner, M.A., 1990. A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American statistical Association*, 85(411), pp.699–704.

White, T., 2012. *Hadoop. the definitive guide*. 3rd ed. Sebastopol, USA: O'Reilly Media, Inc.

White, T., 2015. *Hadoop: The Definitive Guide, 4th Edition*. 4th. O'Reilly Media, p.756.

Wickham, H. and Grolemund, G., 2016. *R for data science: import, tidy, transform, visualize, and model data*. "O'Reilly Media, Inc."

Williamson, E.J. and Forbes, A., 2014. Introduction to propensity scores. *Respirology*, 19(5), pp.625–635.

Williamson, E., Morley, R., Lucas, A., and Carpenter, J., 2012. Propensity scores: from naïve enthusiasm to intuitive understanding. *Statistical Methods in Medical Research*, 21(3). PMID: 21262780, pp.273–293.

Wood, S.N., 2017. *Generalized additive models. an introduction with r*. 2nd ed. Boca Raton, Florida: CRC Press.

Wood, S.N., Li, Z., Shaddick, G., and Augustin, N.H., 2017. Generalized additive models for gigadata: modeling the u.k. black smoke network daily data. *Journal of the American Statistical Association*, 112(519), pp.1199–1210.

Wright, S.T., Ryan, L.M., and Pham, T., 2018. A novel case-control subsampling approach for rapid model exploration of large clustered binary data. *Statistics in Medicine*, 37(6), pp.899–913.

Yang, S., Imbens, G.W., Cui, Z., Faries, D.E., and Kadziola, Z., 2016. Propensity score matching and subclassification in observational studies with multi-level treatments. *Biometrics*, 72(4), pp.1055–1065.

Yang, W., Joffe, M.M., Hennessy, S., and Feldman, H.I., 2014. Covariance adjustment on propensity parameters for continuous treatment in linear models. *Statistics in medicine*, 33(26), pp.4577–4589.

Yoo, R.M., Romano, A., and Kozyrakis, C., 2009. Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system. *Proceedings of the 2009 ieee international symposium on workload characterization, iiswc 2009*, pp.198–207.

Zorn, C., 2005. A Solution to separation in binary response models. *Political Analysis*, 13(2), pp.157–170.

d'Agostino, R.B., 1998. Propensity score methods for bias reduction in the comparison of a treatment to a non-randomized control group. *Statistics in medicine*, 17(19), pp.2265–2281.

nVidia Corporation, 2018. *Nvidia turing gpu architecture* **online**. Available from: `https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf`.

van Buuren, S. and Groothuis-Oudshoorn, K., 2011. mice: multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3), pp.1–67.