# COUPLING MACHINE LEARNING WITH FIDUCIAL INFERENCE, GENETICS AND EPIGENETICS

Gang Li

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Statistics and Operations Research.

Chapel Hill
2021

# ABSTRACT

GANG LI: Coupling Machine Learning with Fiducial Inference, Genetics and Epigenetics
(Under the directions of Jan Hannig and Yun Li)

This dissertation consists of three research topics.

In the first part, we present deep fiducial inference and approximate fiducial computation (AFC) algorithm. Since the mid-2000s, there has been a resurrection of interest in modern modifications of fiducial inference. To date, the main computational tool to extract a generalized fiducial distribution is Markov chain Monte Carlo (MCMC). We propose an alternative way of computing a generalized fiducial distribution that could be used in complex situations. In particular, to overcome the difficulty when the unnormalized fiducial density (needed for MCMC) is intractable, we design a fiducial autoencoder (FAE). The fitted FAE is used to generate generalized fiducial samples of the unknown parameters. To increase accuracy, we then apply an approximate fiducial computation (AFC) algorithm, by rejecting samples that do not replicate the observed data well enough when plugged into a decoder. Our numerical experiments show the effectiveness of our FAE-based inverse solution and the excellent coverage performance of the AFC corrected FAE solution.

In the second part, we present SMNN, a supervised mutual nearest neighbor method, for batch effect correction in single-cell RNA-sequencing (scRNA-seq) data. Batch effect correction has been recognized to be indispensable when integrating single-cell RNA sequencing (scRNA-seq) data from multiple batches. State-of-the-art methods ignore single-cell cluster label information, but such information can improve the effectiveness of batch effect correction, particularly under realistic scenarios where biological differences are not orthogonal to batch effects. To address this issue, we propose SMNN for batch effect correction of scRNA-seq data via supervised mutual nearest neighbor detection. Our extensive evaluations in simulated and real datasets show that SMNN provides improved merging within the corresponding cell types across batches, leading to reduced differentiation across batches over alternative methods including MNN, Seurat v3 and LIGER. Furthermore, SMNN retains more cell-type-specific features, partially manifested by differentially expressed genes identified between cell types after SMNN correction being biologically more relevant, with precision improving by up to 841.0%.

In the third part, we present an ensemble imputation framework for DNA methylation across different platforms. DNA methylation at CpG dinucleotides is a biological process by which methyl groups are added to the DNA molecule. It is one of the most extensively studied epigenetic marks. With technological advancements, geneticists can profile DNA methylation with multiple reliable approaches. However, different profiling platforms can differ substantially in the density and measurements for the CpGs they assess, consequently hindering joint analysis across platforms. For this project, we focus on the two most commonly used commercial methylation platforms from the Illumina company, specifically aiming to impute from the HumanMethylation450 (HM450) BeadChip to 850K CpG sites on the HumanMethylationEPIC (HM850) BeadChip. We present CUE, CpG imputation Ensemble, which ensemble multiple classical statistical and modern machine learning methods. Our results highlight CUE as a valuable tool for imputing from HM450 to HM850.

*I dedicate this dissertation work to my parents,*

*Zhenping Li and Shaowei Shi,*

*who have loved and supported me throughout my life.*

## ACKNOWLEDGEMENTS

What a journey! This dissertation would not be successful without the help and the encouragement from many people. Here I only listed some of them, but I will always keep all of those who helped me in my mind.

For this dissertation I am forever grateful of the guidance of my advisors, Dr. Jan Hannig from statistics and operations research department and Dr. Yun Li form genetics department. I always tell others that I am lucky to have one advisor mainly working in theoretical statistics and one advisor who mainly works in genetics application end. Jan is the one who first introduced me to the Bayes Fiducial Frequentist (BFF) community, where I learned a lot about the fundamental inference problems. I enjoy this journey so far and hope I could keep contributing to this field and make our dream come true. Yun is the one who carefully guide me to the genetics field, where I was able to apply all ”weapons”, I had learned, to see the real impacts on some scientific frontiers, and explored to build my own weapons. With the exclusive opportunity to get my hands on both fields, I am able to choose the career I would like to pursue.

I would also like to thank my committe members Dr. Yufeng Liu, Dr. Andrew Nobel and Dr. Michael Love. I enjoyed the courses you taught me a lot and those courses provides me not only invaluable tools for this dissertation but also the mind and the logic to deal with any future complex problems.

Finally, I am forever thankful to my parents, Zhenping Li and Shaowei Shi, for the life they have provided for me. It’s my pleasure to be their child and make them proud.

# TABLE OF CONTENTS

## LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| AFC | Approximate Fiducial Computation |
| CC | Confidence Curve |
| CI | Confidence Interval |
| CUE | CpG impUtation Ensemble |
| ELGAN | Extremely Low Gestational Age Newborns study |
| FAE | Fiducial Autoencoder |
| GFD | Generalized Fiducial Distribution |
| GFI | Generalized Fiducial Inference |
| HM27 | HumanMethylation27 BeadChips |
| HM450 | HumanMethylation450 BeadChips |
| HM850 | HumanMethylationEPIC BeadChips |
| KNN | K-Nearest Neighbors |
| MLE | Maximum Likelihood Estimation |
| MNN | Mutual Nearest Neighbor |
| PFR | Penalized Functional Regression |
| PTSD | Posttraumatic Stress Disorder genetics repository |
| RF | Random Forest |
| SMNN | Supervised Mutual Nearest Neighbor |

CHAPTER 1

# Introduction

With the advancement of technology and the volume of big data, data science and statistics play even more important roles in the 21st century's genetics and epigenetics than before. In particular, machine learning methods are of increasing importance in both foundations of statistical inference and modern genetics applications. This dissertation contains two parts. In the first part of this dissertation (Chapter 2), we focus on the foundation of the inference problem. We introduce a new inference framework, deep fiducial inference, accompanied by a computational algorithm, approximate fiducial computation. This new framework provides an alternate approach to generalized fiducial inference, compared to the traditional MCMC-like approach, and is designed for big data circumstances.

Since the mid-2000s, there has been a resurrection of interest in modern modifications of fiducial inference. To date, the main computational tool to extract a generalized fiducial distribution is MCMC and MCMC derived methods. We aim to propose an alternative way of computing a generalized fiducial distribution via inverse solutions. The main idea of the inverse solution is to use the inverse function or the approximate inverse function as a sampler to directly generate fiducial samples, and then get the kernel density estimation of fiducial distribution without knowing the analytical density form, which might be difficult or impossible to calculate. With the fiducial distribution built on the inverse solution, typical inference can be carried on. The main challenges for this project are how to approximate the inverse function and how to generate fiducial samples lives on the data manifold. Deep fiducial inference framework employs deep neural networks to approximate the inverse function and approximate fiducial computation is an algorithm that uses a reject-sampling scheme to generate valid fiducial samples. The competitive performance of AFC corrected FAE solutions both in terms of efficiency and accuracy suggests that this is a promising area for future research.

In the second part of this dissertation (Chapters 3 and 4), we introduce one application in genetics in Chapter 3 and one application in epigenetics in Chapter 4. Although the methods we built and improved for those two applications are rooted in two particular examples, they can also be generalized to other situations. However, for the brevity and consistency in each chapter, we focus on the motivated application itself in this dissertation.

An ever-increasing amount of single-cell RNA-sequencing (scRNA-seq) data has been generated as scRNA-seq technologies mature and sequencing costs continue dropping. However, large-scale scRNA-seq data, for example, those profiling tens of thousands to millions of cells (such as the Human Cell Atlas Project (Rozenblatt-Rosen et al., 2017), almost inevitably involve multiple batches across time points, laboratories, or experimental protocols. The presence of batch effect renders joint analysis across batches challenging (Chen and Zhou, 2017; Stegle et al., 2015). Batch effect, or systematic differences in gene expression profiles across batches, not only can obscure the true underlying biology, but also may lead to spurious findings. Thus, batch effect correction, which aims to mitigate the discrepancies across batches, is crucial and deemed indispensable for the analysis of scRNA-seq data across batches (Stuart and Satija, 2019a). Because of its importance, several batch effects correction methods have been recently proposed and implemented. However, when applied to scRNA-seq data, the corrected results derived from these methods widely adopted for bulk RNA-seq data might be even inferior to raw data without no correction, in some extreme cases (Haghverdi et al., 2018).

To address the heterogeneity and high dimensionality of complex data, we present SMNN, a supervised mutual nearest neighbor method, for batch effect correction in single-cell RNA-sequencing (scRNA-seq) data in Chapter 3. An ever-increasing deluge of scRNA-seq data has been generated, often involving different time points, laboratories, or sequencing protocols. Batch effect correction has been recognized to be indispensable when integrating scRNA-seq data from multiple batches. SMNN either takes cluster/cell-type label information as input or infers cell types using scRNA-seq clustering in the absence of such information. It then detects mutual nearest neighbors within matched cell types and corrects batch effect accordingly. Compared to other state-of-arts batch effects correction methods, SMNN provides improved merging within the corresponding cell types across batches and retains more cell-type-specific features after correction.

We next consider an application in epigenetics. DNA methylation of cytosine residues at CpG dinucleotides is one of the most extensively studied epigenetic marks. Rich recent literature provides evidence regarding its important role not only in normal development but also in risk and progression to many diseases (Bird, 2002; Gonzalo, 2010; Joubert et al., 2016; Klutstein et al., 2016; Iurlaro et al., 2017; Horvath and Raj, 2018; Turecki and Meaney, 2016; Bakusic et al., 2017). With the emergence of powerful technologies such as DNA methylation microarray (Bibikova et al., 2011b) and bisulfite sequencing, geneticists can profile DNA methylation at increasingly higher resolutions. Taking methylation microarrays for an example, we have witnessed new platforms replacing old ones every few years (Moran et al., 2016; Bibikova et al., 2009; Dedeurwaerder et al., 2014). However, different platforms (for example, the widely used Illumina HumanMethylation27, HumanMethylation450, MethylationEPIC BeadChips) target different CpG sites and have different marker densities, consequently hindering joint analysis across platforms.

In Chapter 4, we aim to impute an HM450 dataset up to an HM850 dataset, for increased coverage of this epigenomic landscape. Specifically, we present CUE, CpG imputation Ensemble, which ensemble multiple classical statistical and modern machine learning methods, to impute from the Illumina HumanMethylation450 (HM450) BeadChip to 850K CpG sites on the Illumina HumanMethylationEPIC (HM850) BeadChip. We analyzed data from two population cohorts measured both by HM450 and HM850: the Extremely Low Gestational Age Newborns (ELGAN) study (n=127, placenta) and the Posttraumatic Stress Disorder (PTSD) study (n=144, whole blood). Our results highlight CUE as a valuable tool for imputing from HM450 to HM850.

# CHAPTER 2
# Deep Fiducial Inference and Approximate Fiducial Computation

## 2.1 Introduction

Generalized fiducial inference (GFI) (Hannig et al., 2016), a modern re-incarnation of R. A. Fisher's fiducial inference (Fisher, 1930), provides inferentially meaningful probability statements about subsets of parameter space without the need for subjective prior information. GFI specifies a generalized fiducial distribution (GFD) by defining a data-dependent measure on the parameter space through an inverse of a data-generating algorithm (see Sections 2.3). The data-generating algorithm plays the role of a model and is sometimes called data-generating equation or data-generating function. With GFD as a distribution estimator for the fixed parameter(s), we can further define approximate confidence (fiducial) sets which are often shown in simulation to have very desired properties (Hannig et al., 2016).

Given the data-generating algorithm and the corresponding density of the GFD, one can form point estimates and asymptotic confidence sets similarly to a Bayesian posterior density. Standard MCMC-type sampling techniques have already been successfully implemented in many situations; see (Hannig et al., 2016) and the references therein. However, sometimes the generalized fiducial density can be hard to compute. Especially when the likelihood function of the data is intractable, MCMC may be difficult or impossible to implement. For this reason we propose using a deep neural network to approximate the nonlinear inverse to the data-generating algorithm, and to generate an approximation to the fiducial distribution without knowing the exact form the density.

Autoencoder (AE) (Hinton and Zemel, 1994; Schmidhuber, 2015) is a type of neural network architecture to learn data code, an efficient representation of the data, and to reduce data dimensions. Different variants of autoencoder (Vincent et al., 2010) have gained a lot of successful results in applications of the neural network, such as variational autoencoder (Doersch, 2016) and variational Bayes (Kingma and Welling, 2013). The denoising autoencoder (Vincent et al., 2008) is designed to remove the noise effects without losing the essential signal in the data. Inspired by the autoencoder's architecture, we design an FAE, which continues to use a neural network as the encoder to approximate the inverse of data generating function but employs the exact data-generating function as the decoder. Furthermore, we design and implement approximate fiducial computation algorithms, in addition to FAE, to generate generalized fiducial samples.

The rest of this chapter is organized as follows. In Section 2.2, we first introduce the generalized fiducial inference and its standard MCMC based solution. In Section 2.3.1, we design a fiducial autoencoder (FAE) to approximate the inverse function when it is not directly available, and we also propose the approximate fiducial computation (AFC) algorithm to further increase the accuracy of FAE. We demonstrate the performance of FAE and AFC in four numerical examples in Section 2.4. Section 2.5 concludes the chapter with some discussions.

## 2.2 Background on generalized fiducial inference

Before introducing our computational tool, FAE, we first briefly present the current state of ideas in GFI. The GFI framework is based on linking the observed data $\mathbf{x}$, the unknown parameter $\mu$, and some random component $\mathbf{z}$ via a data generating algorithm, also called data generating function. We shall discuss this in detail.

**Data generating algorithm:** The data generating algorithm is

$$\mathbf{x} = f(\mathbf{z}, \mu). \tag{2.1}$$

where $\mathbf{x}$ is the data, $\mu$ is the parameter, and $\mathbf{z}$ is a random component with random distribution $F_0$, e.g., i.i.d. standard Gaussian distribution, that is completely known and independent of the parameter $\mu$. It is assumed that the data could have been generated by fixing some parameter value $\mu$, generating a value $\mathbf{z}$ from distribution $F_0$, and plugging them into the equation (2.1). The data $\mathbf{x}$ is assumed observed, while the values of $\mu$ and $\mathbf{z}$ are unobserved. Notice that this procedure uniquely determines the sampling distribution of $\mathbf{x}$, and $\mu$ is fixed.

**Generalized fiducial distribution:** If both $\mathbf{x}$ and $\mathbf{z}$ were known, then inverting equation (2.1), i.e., solving for $\mu$, would give us the unknown parameter. Heuristically speaking, since $\mathbf{z}$ is unknown, we estimate it using its distribution $F_0$ and define GFD by propagating the distribution

$F_0$ through the inverse of the data generating algorithm. More precisely, GFD is defined rigorously as follows (Hannig et al., 2016):

For an $\epsilon > 0$, consider the following inverse problem

$$g(\mathbf{x}, \mathbf{z}) = \underset{\mu^\star}{\operatorname{argmin}} \|\mathbf{x} - f(\mathbf{z}, \mu^\star)\|. \tag{2.2}$$

In this chapter we will use $\|\cdot\|$ as either $\ell_2$ or Frobenius norm and call the inverse of the data generating algorithm $g(\mathbf{x}, \mathbf{z})$ the inverse function. Next define the random variable $\mu^\star_\epsilon = g(\mathbf{x}, \mathbf{Z}^\star_\epsilon)$, where $\mathbf{Z}^\star_\epsilon$ has distribution $F_0$ truncated to the set

$$\mathcal{C}_\epsilon = \{\mathbf{Z}^\star_\epsilon : \|\mathbf{x} - f(\mathbf{Z}^\star_\epsilon, \mu^\star_\epsilon)\| = \|\mathbf{x} - f(\mathbf{Z}^\star_\epsilon, g(\mathbf{x}, \mathbf{Z}^\star_\epsilon))\| \leq \epsilon\}. \tag{2.3}$$

Then assuming that the random variable $\mu^\star_\epsilon$ converges in distribution as $\epsilon \to 0$, GFD is defined as the limiting distribution $\mu^\star = \lim_{\epsilon \to 0} \mu^\star_\epsilon$. In practice, a small threshold $\epsilon$ is selected for computation. Notice that the fiducial distribution of $\mu^\star$ depends on the observed data $\mathbf{x}$.

With this GFD as the distribution estimator for $\mu$, one can form point estimators and construct approximate confidence sets just like using a Bayesian posterior distribution. Notice that GFI does not need any prior information, and in fact, it provides a systematic approach to deriving objective Bayes-like posterior distributions (Hannig et al., 2016). The following theorem is a basis for most current numerical implementations of GFD:

**Theorem 1** ((Hannig et al., 2016))**.** *Under mild conditions, the limiting distribution above has density*

$$r_{\mathbf{x}}(\boldsymbol{\mu}) = \frac{L(\mathbf{x}|\boldsymbol{\mu}) J(\mathbf{x}, \boldsymbol{\mu})}{\int L(\mathbf{x}|\boldsymbol{\mu}') J(\mathbf{x}, \boldsymbol{\mu}') \, d\boldsymbol{\mu}'}$$

*where $L(\mathbf{x}|\boldsymbol{\mu})$ is the likelihood function and $J(\mathbf{x}, \boldsymbol{\mu}) = D\left(\nabla_{\boldsymbol{\mu}} f\,(\mathbf{z}, \boldsymbol{\mu})|_{\mathbf{z}=f^{-1}(\mathbf{x}, \boldsymbol{\mu})}\right)$, where $\nabla_{\boldsymbol{\mu}}$ is a gradient matrix with respect to $\mu$ and $D(A) = (\det A'A)^{\frac{1}{2}}$.*

Notice that the form of the GFD density in the theorem has a similar form to a Bayesian posterior density, where the Jacobian function $J(x, \mu)$ plays the role of a data-dependent prior. Similarly to Bayesian inference, one potential challenge using this formula to calculate the fiducial density is that the denominator might be intractable. Traditionally this has been addressed using MCMC algorithms. There are many well-known challenges associated with MCMC such as computational speeds and intractable high dimensional integrals. In particular, MCMC is difficult to implement when a closed form of the likelihood is unknown or difficult to derive.

In this chapter, we propose a sampling-based computational solution using (2.2) directly. There are two main challenges in implementing this approach. First, the solution to the optimization problem in (2.2) might not have an analytical form or might be difficult to calculate. Therefore we propose to approximate the solution $g(\mathbf{x}, \mathbf{Z}_\epsilon^\star)$ using a deep neural network (see details in section 2.3.1). Second, one needs to generate $\mathbf{Z}_\epsilon^\star$ from $\mathcal{C}_\epsilon$ for a small $\epsilon$, which could be very challenging when $\mathcal{C}_0$ is a lower dimensional-manifold. We propose an AFC algorithm (see details in section 2.3.2) to effectively generate fiducial samples.

**Confidence curve:** before proceeding further, we also introduce the confidence curve (CC), a useful graphical tool for plotting epistemic distributions (Birnbaum, 1961). Given $R_{\mathbf{x}}(\mu)$ the distribution function of GFD corresponding to a one-dimensional marginal of density in Theorem 1, CC is defined as $2|R_{\mathbf{x}}(\mu) - 0.5|$. CC shows two-sided confidence intervals at all significance levels stacked upon each other (e.g., see Figure 2.1 ). The median of GFD is the point where CC touches $x$-axis and can be used as a point estimator.

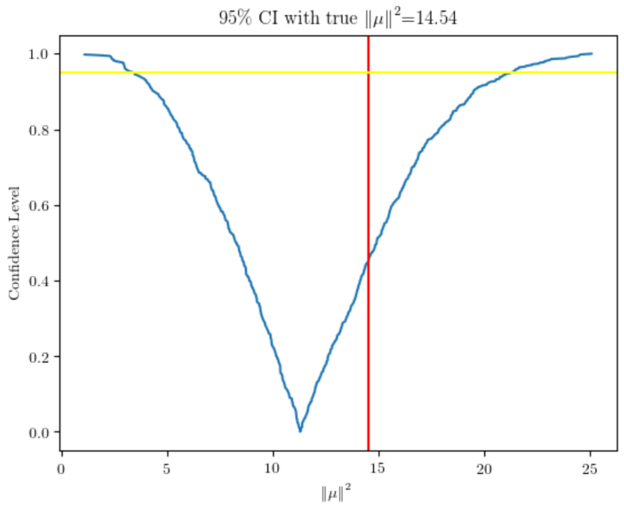


**Figure 2.1:** Confidence curve.

See Figure 2.1: a vertical line shows the true parameter value, while a horizontal line with height $\alpha$ $(0 < \alpha < 1)$ across the CC provides an $\alpha$ level, equal tailed, two-sided confidence interval.

## 2.3 Methodology

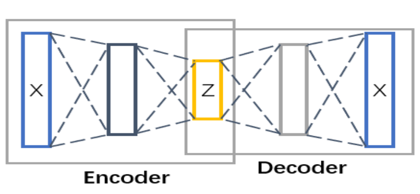
### 2.3.1 Deep fiducial inference

Before we introduce our FAE, we first discuss some terminology used within deep learning (Goodfellow et al., 2016). Artificial neural network is a function described using a directed graph. The vertices are simple functions, typically a linear function with output modified by a non-linear function called an activation function. Examples of activation functions include the identity, called linear activation function, $\max(x, 0)$, called ReLU (Nair and Hinton, 2010), sigmoid function, hyperbolic tangent function, etc. Activation functions are called squashing if their range is a compact interval. The edges of the graph describe the relationships between inputs and outputs of the simple functions. A feedforward neural network, the most basic type of neural network, is an artificial neural network wherein connections between the nodes do not form a cycle. Training neural network describes the process of fitting the coefficients of the linear functions in the nodes of the graph to training data. This is typically done using a stochastic optimization algorithm, e.g., stochastic gradient descent (SGD) (Bottou, 2010) or Adam (Kingma and Ba, 2014).

In this section, we design a fiducial FAE, a deep neural network based approximation for the inverse function (2.2), for circumstances where the inverse function might not have an analytical form or might be difficult to calculate. There are standard theoretical guarantees for a good approximation (Hornik et al., 1989) (Theorem 2) and our numerical experiments in Section 2.4 also validate this approach.

**Theorem 2** (Universal Approximation Theorem)**.** *A feedforward network with a linear output layer and at least one hidden layer with any squashing activation functions can approximate any Borel measurable function, provided that the network is given enough hidden units.*

The most basic version of autoencoder (AE) (Hinton and Zemel, 1994; Schmidhuber, 2015) often contains two parts, encoder and decoder. The encoder maps from the observation space $\mathcal{X}$ to latent coder space $\mathcal{Z}$ while the decoder does the inverse (see left panel of Figure 2.2).



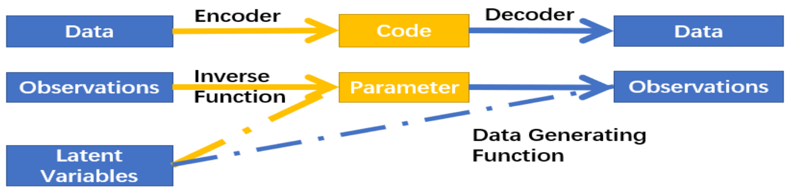


**Figure 2.2:** Autoencoder and fiducial inference. The left panel shows a schematic of a generic autoencoder while the right panel compares autoencoder with usual generalized fiducial inference.

We compare the encode-decode process of the standard AE to the usual fiducial inference in the right panel of Figure 2.2. The process by which the data generating function maps the parameter and the latent variables to the observations is similar to the mechanism by which the decoder maps the code to the data in the usual AE. Similarly, the encoder in AE plays a role akin to the inverse of the data generating algorithm in fiducial inference. Inspired by this analogy, we create the FAE framework which will use as an encoder a neural network that will approximate the inverse of the data generating algorithm $\hat{\mu} = g(\mathbf{x}, \mathbf{z})$ to be estimated, and as a decoder the data generating algorithm $\hat{\mathbf{x}} = f(\mathbf{z}, \hat{\mu})$ that is known to us.

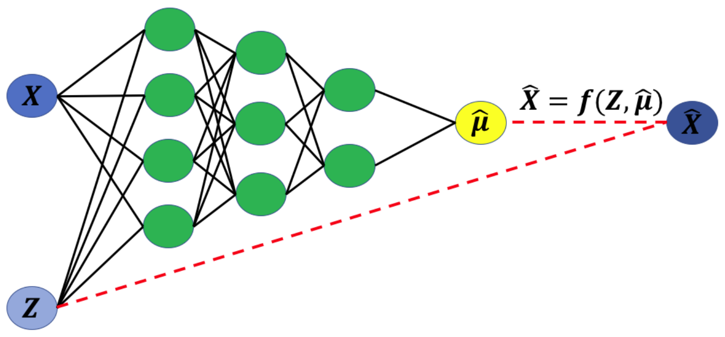


**Figure 2.3:** Standard fiducial autoencoder architecture.

More precisely, the standard FAE has two input nodes, the observation data $\mathbf{x}$ and the latent variable $\mathbf{z}$, one final output node, the prediction $\hat{\mathbf{x}}$, and one intermediate output node, the prediction $\hat{\mu}$. The FAE's encoder usually consists of a deep fully connected neural network, which maps the $\mathbf{x}$ and $\mathbf{z}$ to $\hat{\mu}$. The architecture of the encoder is flexible and should be selected based on the problem so that it can learn the inverse function well. Unlike the traditional autoencoder which uses the code layer $\hat{\mu}$ to predict the input nodes, the FAE's decoder employs the known data generating algorithm $\hat{\mathbf{x}} = f(\mathbf{z}, \hat{\mu})$. We built an additional straight connection from the input $\mathbf{z}$ node directly to the final output node $\hat{\mathbf{x}}$, which is an extreme case of the residual neural network, see Figure 2.3. Two main differences between traditional autoencoder and our FAE are summarized as follows:

- First, we use the exact data generating algorithm instead of another group of neural network as the decoder.
- Second, our decoder takes as the input not only code layer $\hat{\mu}$ but also the random component $\boldsymbol{Z}^{\star}$.

One big advantage of FAE, compared to the usual AE, is that it never suffers a lack of training data. This is because the training data is obtained by simulation from the data generating algorithm that is known to us. In particular, given a number of training $\{\mu_k, k = 1, 2, ..., n_{train}\}$, we could generate as many pairs of $\mathbf{x}_k$ and $\mathbf{z}_k$ for training FAE as needed. The limitation is that we need to make sure that the training set contains a large enough number of data sets that are similar to the observed data. For example, if the training set does not contain enough values of $\mu_k$ that are close to the true $\mu$, we cannot guarantee the FAE to provide good answers. Therefore, we recommend that enough of the training data is generated using values of $\mu_k$ close to some pilot estimator of $\hat{\mu}$.; for example, the least square estimator as in Section 2.4.4.

Since the decoder is the fixed deterministic data generating function, the loss of FAE quantifies how well the encoder approximates the inverse function. The total loss function we use for training our neural network contains two parts, the mean square error with regard to $\hat{\mathbf{x}}$ and $\hat{\mu}$: $L = w_1\|\mathbf{x} - \hat{\mathbf{x}}\|^2 + w_2\|\mu - \hat{\mu}\|^2$, where $w_1$ and $w_2$ are user-selected weights. If we set $w_1 = 0$ and $w_2 = 1$, we would in effect be training a neural network mapping directly from $X$ and $Z$ to $\mu$, and we would miss the information provided by the data generating function. Our numerical experiments show that this choice is not optimal. After incorporating the information from the data generating function, the approximation performance is greatly improved. On the other hand, if we only use the mean square error based on predicting the data, $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$ ($w_1 = 1$ and $w_2 = 0$), the FAE would still do reasonably well. However, using MSE for $\hat{\mu}$ with appropriate weights does not only increase the convergence speed but also improves the FAE's performance. In our numerical experiments we manually select $w_1, w_2$ so that the loss with regard to the observation, $w_1\|\mathbf{x} - \hat{\mathbf{x}}\|^2$, and the loss with regard to the parameter, $w_2\|\mu - \hat{\mu}\|^2$, are roughly of the same magnitude. For example, we set $w_1 = w_2 = 1$ as the default parameters in Section 2.4.2.

### 2.3.2 Approximate fiducial computation

Once the FAE converges, we simply apply the encoder with pairs of the fixed observed $\mathbf{x}$ and a number of simulated $\boldsymbol{Z}^\star$, which are the independent identical copy of $\boldsymbol{Z}$, to get the estimated $\mu^\star$. Truncating these fiducial samples to the set $\boldsymbol{Z}^\star \in \mathcal{C}_\varepsilon$, defined in (2.3), using approximate fiducial computation algorithms, one will achieve samples from the approximate generalized fiducial distribution that can be used for inference. Notice that using only $\boldsymbol{Z}^\star \in \mathcal{C}_\varepsilon$ is very natural as we are discarding those $\boldsymbol{Z}^\star$ for which FAE predicts values of $\mathbf{x}^\star$ that are far from the observed value $\mathbf{x}$.

This is similar to the approximate Bayesian computation (ABC) that have been intensively studied in the past years (Blum et al., 2013). Similar to ABC, which is designed to overcome the intractable likelihood function, AFC avoids directly calculating the fiducial density and the inverse function. One major difference is that while ABC uses a prior distribution to get candidate $\mu^\star$, the AFC algorithm uses the optimization problem (2.2) eschewing the need to select an arbitrary prior distribution.

The steps of the AFC algorithms are summarized in Algorithm 1. Note that for different problems and even for different parameters within the same problem we might need to select different thresholds ($\varepsilon$ in equation (2.3)) to efficiently get valid approximate generalized fiducial samples. If the threshold is too big, then we might get biased samples; if the threshold is too small, it would be very difficult for AFC to generate enough samples passing the threshold condition. In practice, we could use one random batch of samples to approximate the distribution of $dist(\mathbf{x}, \mathbf{x}^\star)$, and select the threshold according to the efficiency and the accuracy we expect for the AFC.

We call the samples from Algorithm 1 approximate generalized fiducial samples. Aggregating those samples into an empirical distribution provides an approximation to the GFD and form the bases for making statistical inference. In our numerical experiments, we study statistical properties of the point estimators based on the mean and median, and the $\alpha$-level approximate confidence intervals based on the $(1-\alpha)/2$ and $(1+\alpha)/2$ empirical quantiles of the generalized fiducial samples. The corresponding confidence curves are reported to visualize the approximate fiducial distribution. Finally, we remark that the AFC algorithm can be useful even in situations when we know the analytical form of the fiducial inverse function (2.2). We demonstrate this on an example in Section 2.4.1.

**Algorithm 1: Approximate Fiducial Computation (AFC)**

```
   Input  : Data generating function f; (approximate) inverse function g; distribution $F_0$ for
            generating $\boldsymbol{Z}^\star$; observation $\mathbf{x}$; threshold $\epsilon$
   Output: Generalized fiducial samples (GFS)
 1 while ($itr < max\_itr$) and (# of GFS $< N$) do
 2     Sample $\boldsymbol{Z}^\star$ from $F_0$;
 3     $\mu^\star = g(\mathbf{x}, \boldsymbol{Z}^\star)$ ;
 4     $\mathbf{x}^\star = f(\boldsymbol{Z}^\star, \mu^\star)$ ;
 5     if $dist(\mathbf{x}, \mathbf{x}^\star) < \epsilon$ then
 6         Accept $\mu^\star$;
 7     else
 8         Reject $\mu^\star$;
 9     end
10     $itr = itr + 1$;
11 end
```

## 2.4 Simulation

We report the results of four numerical experiments. The first is the location-scale Laplace distribution, a relatively straightforward example illustrating AFC when the analytical form of the inverse of the data generating algorithm is available. The second example demonstrates that FAE can learn a non-linear inverse function and shows how AFC improves the inference with the decreasing thresholds $\varepsilon$. The third example shows AFC improves the inference of a function of parameter. The fourth example compares FAE and several competing methods using a highly non-linear regression model.

### 2.4.1 Location-scale Laplace distribution

Consider the data generating algorithm

$$\mathbf{x} = f(\mathbf{z}, \mu) = \theta \times \mathbf{1} + \sigma \mathbf{z},$$

where $\mathbf{x} = (x_1, \ldots, x_m)^\top$ are the observed data, $\mu = (\theta, \sigma)$ are the unknown scalar parameters, and $\mathbf{z} = (z_1, \ldots, z_m)^\top$ is a vector of i.i.d. samples from standard Laplace distribution, i.e., density $f(z) = e^{-|z|}/2$. Straightforward calculations show that $\mu^\star = (\theta^\star, \sigma^\star) = g(\mathbf{x}, \boldsymbol{Z}^\star)$ is

$$\sigma^\star = \frac{\sum_{i=1}^m (x_i - \bar{x})(Z_i^\star - \bar{Z}^\star)}{\sum_{i=1}^m (Z_i^\star - \bar{Z}^\star)^2} \qquad \text{and} \qquad \theta^\star = \bar{x} - \sigma^\star \bar{Z}^\star, \tag{2.4}$$

where $\bar{x} = m^{-1} \sum_{i=1}^m x_i$ and $\bar{Z}^\star = m^{-1} \sum_{i=1}^m Z_i^\star$.

Since we have the analytical solution for the inverse function we can easily compute $\mu^\star = g(\mathbf{x}, \boldsymbol{Z}^\star)$ for any $\boldsymbol{Z}^\star$ following Laplace distribution. This is the baseline method without AFC algorithm. Then we use the data generating algorithm to compute the predicted observation $\mathbf{x}^\star = f(\boldsymbol{Z}^\star, \mu^\star) = \theta^\star \times \mathbf{1} + \sigma^\star \boldsymbol{Z}^\star$, and accept the proposed $\mu^\star$ if and only if $\|\mathbf{x} - \mathbf{x}^\star\| \leq \epsilon$ for some pre-selected threshold $\epsilon$. This process is repeated until we get the desired number of AFC corrected fiducial samples.

Notice that for small enough $\epsilon$ the AFC algorithm will not accept $\boldsymbol{Z}^\star$ that does not match the order of the data. Since the distribution of $\boldsymbol{Z}^\star$ is exchangeable, we will without loss of generality assume that both $x_1 < \cdots < x_m$ and $Z_1^\star < \cdots < Z_m^\star$ are ordered. This will make the algorithm more efficient. Finally, because of invariance of the location-scale model, it is enough to perform the simulation using only one value the true parameter value, say $\theta = 0$ and $\sigma = 1$.

Figure 2.4 shows an example of CC with and without AFC for the same dataset containing $m = 100$ observations. In the left panel, we see that the CC for $\theta$ with AFC (orange) is much thinner than the curve without AFC (blue). And the corresponding fiducial median with AFC is also more close to the truth 0. In the right panel, the CC for $\sigma$ with AFC provides a slightly narrower 90% confidence interval compared with that without AFC.

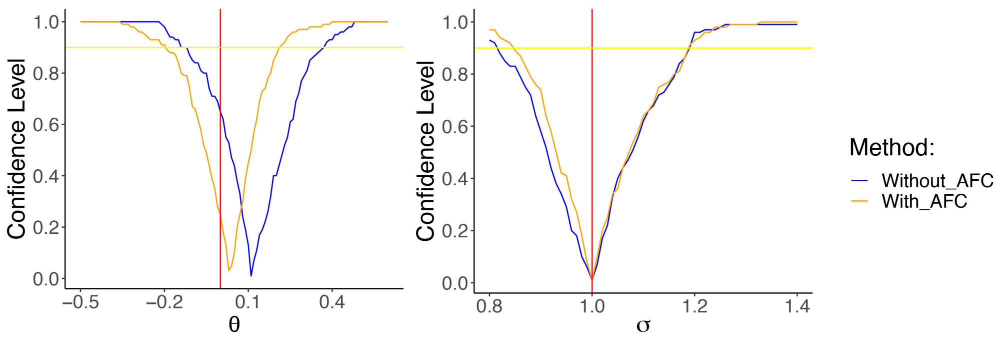


**Figure 2.4:** Confidence curves of marginal distributions for the Laplace example. The red vertical lines shows the true parameters ($\theta = 0$ and $\sigma = 1$) generated the observed data. The yellow horizontal line indicates the 90% confidence intervals.

**Table 2.1:** Inference performances without AFC for Laplace example.

| **Truth** | **Coverage** | **Expected CI Length** | **Expected Mean** | **Expected Median** |
|---|---|---|---|---|
| $\theta = 0$ | 0.835 | 0.4468 | 0.0042 | 0.0048 |
| $\sigma = 1$ | 0.935 | 0.3636 | 0.9717 | 0.9657 |

Tables 2.1 and 2.2 compare the inference performance with and without AFC. In particular, we fixed the true parameters $\theta = 0$ and $\sigma = 1$, and observed 200 data sets each containing $m = 100$ observations $\{\mathbf{x}_k = (x_{1,k}, \ldots, x_{m,k})^\top, k = 1, \ldots, 200\}$. For each $\mathbf{x}_k$, we used 1,000 random $\{\boldsymbol{Z}^\star_{j,k}, j = 1, \ldots, 1000\}$ to obtain the corresponding $\mu^\star_{j,k}$ and use them to obtain estimators of the mean, median and 90% confidence set. We report four key statistics averaged over the 200 data sets, namely, coverage, the expected length of the confidence intervals, the expected value of the fiducial mean and median. We can see that AFC provides more accurate point estimations for both mean estimator and median estimator. In addition, the length of confidence intervals with AFC at the same confidence level are shorter than those without AFC. Lastly, AFC improves the coverage for $\theta$ and $\sigma$.

**Table 2.2:** Inference performances with AFC for Laplace example.

| **Truth** | **Coverage** | **Expected CI Length** | **Expected Mean** | **Expected Median** |
|---|---|---|---|---|
| $\theta = 0$ | 0.870 | 0.4244 | 0.0033 | 0.0032 |
| $\sigma = 1$ | 0.940 | 0.3466 | 0.9872 | 0.9796 |

### 2.4.2 Nonlinear data generating algorithm

Consider the following non-linear model defined by the data generating algorithm

$$\mathbf{x} = \mu \times \mathbf{1} + \mu^{\frac{q}{2}} \times \mathbf{z},$$

where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^m (m = 3)$ and $\mu \in \mathbb{R}$, with $z_i$ as the realizations of independent standard normal random variables. Here, $\mu$ is the parameter of interest and $q/2$ is known. When $q = 0$, this becomes a standard location parameter problem: $\boldsymbol{X} = \mu \times \mathbf{1} + \boldsymbol{Z}$, $q = 2$ is a scale parameter problem $\boldsymbol{X} = \mu \times (\mathbf{1} + \boldsymbol{Z})$. and $q = 4$ leads to a one-parameter exponential family. In this numerical experiment we choose $q = 3$ in order to have a true non-linear model.

To train the FAE, we simulate 100,000 $\mu_k$ from the $Uniform(0, 6)$ distribution and randomly split them to 80,000 training samples and 20,000 validation samples. These $\mu_k$ serve as the true parameter for training purposes. We use an 11 layer fully-connected neural network as the encoder with ReLU as the activation function (Nair and Hinton, 2010) and with Adam as the optimizer (Kingma and Ba, 2014). The FAE converges after 10 epochs. All FAEs in this chapter are implemented using on Keras in Python (Chollet et al., 2015).

After the FAE has converged we will use the fitted encoder for inference. In particular we first randomly sample $\boldsymbol{Z}_j^\star, j = 1, \ldots, n_{fid}$ ($n_{fid} = 1000$) and use them together with the observed data $\mathbf{x}$ in the encoder to obtain $n_{fid}$ samples of $\mu_j^\star$. Notice that the same observed value of $\mathbf{x}$ is used for all $\mu_j^\star$. Using the $\mu_j^\star$ we can estimate an empirical distribution function of the GFD for $\mu$ and draw a corresponding confidence curve. Figure 2.5 shows confidence curves for nine different realizations of $\mathbf{x}$. Most of the confidence intervals are wide and only 7 of 9 true $\mu$ are covered by the corresponding 95% level confidence interval.

Table 2.3 presents the results of a simulation study using FAE without AFC. In particular, we fixed four true $\mu \in 1, 2, 3, 4$, and observed 200 data sets $\{\mathbf{x}_k, k = 1, \ldots, 200\}$ for each of the four values of $\mu$. For each $\mathbf{x}_k$, we use 1,000 random $\boldsymbol{Z}_j^\star$ to obtain $\mu_j^\star$ and use them to obtain fiducial mean, median and 90% confidence set. Table 2.3 reports four key statistics averaged over the 200 data sets, namely, coverage, the expected length of the confidence intervals, the expected value of

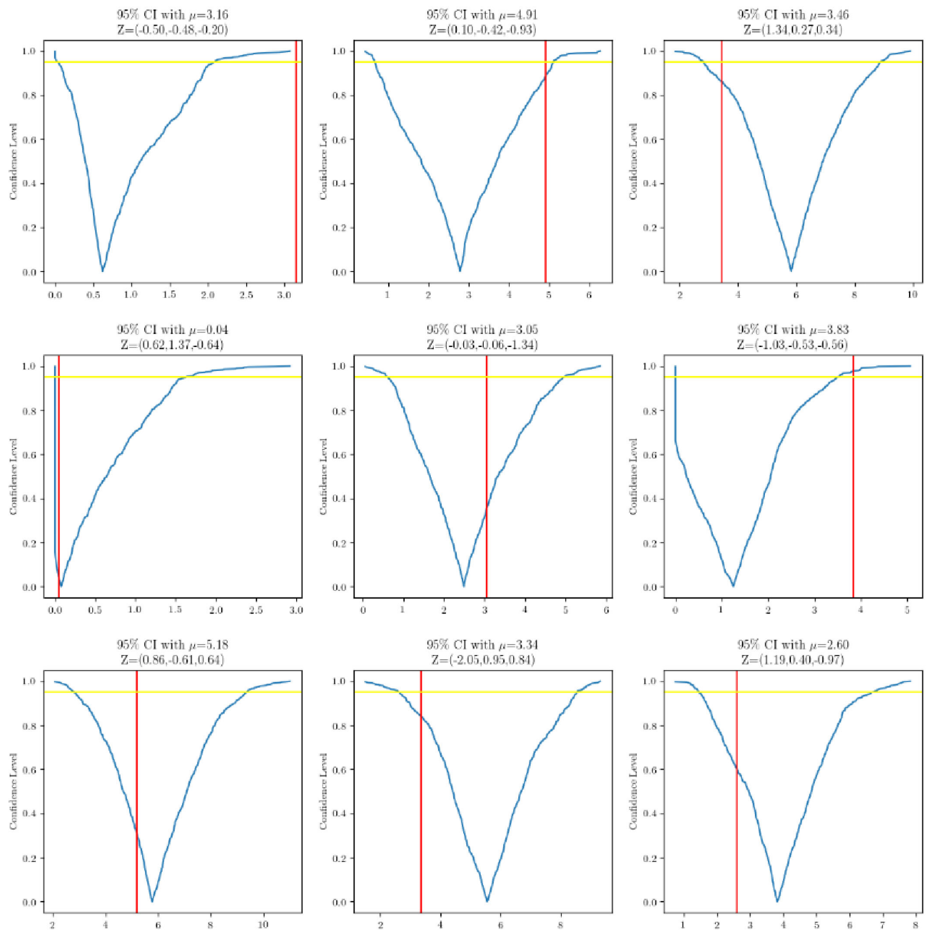


**Figure 2.5:** Nine confidence curves for nonlinear data generating algorithm of FAE without AFC. The red vertical lines correspond to the true parameters. The intersection of the confidence level (yellow horizontal line) and the blue confidence curve shows the 95% confidence interval. The values of $Z$ are the random realizations used in the data generating algorithm.

the fiducial mean and median . We can see that the confidence sets are very wide and the expected fiducial mean and the expected fiducial median are biased.

Next, we investigate the effect of AFC for a fixed observation. Figure 2.6 shows confidence curves for the same data $\mathbf{x}$ with different thresholds; the true $\mu = 3.5$. As the threshold $\epsilon$ decreases, the marginal fiducial median is getting closer to the true $\mu$. Additionally, the fiducial distribution is becoming more concentrated.

Finally, we repeated the simulation study for FAE with AFC to generate 1,000 threshold-admissible samples to form inference estimates. As shown in Table 2.4, the fiducial mean and

**Table 2.3:** Inference performances without AFC for nonlinear data generating algorithm.

| **True $\mu$** | **Coverage** | **Expected CI Length** | **Expected Mean** | **Expected Median** |
|---|---|---|---|---|
| 1 | 0.985 | 2.03 | 1.07 | 0.90 |
| 2 | 0.905 | 3.50 | 2.64 | 2.49 |
| 3 | 0.865 | 4.25 | 3.85 | 3.81 |
| 4 | 0.870 | 4.28 | 4.48 | 4.45 |

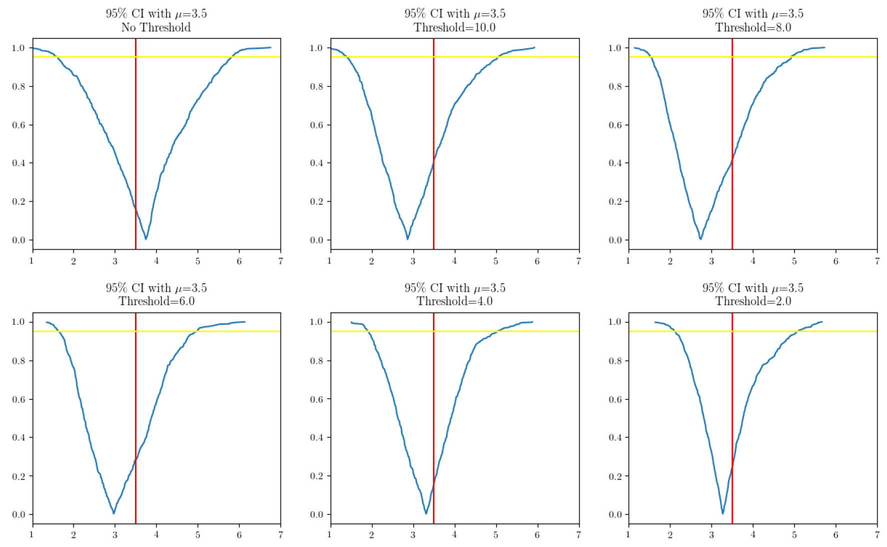


**Figure 2.6:** Confidence curves with different thresholds. The red vertical lines correspond to the true parameter ($\mu = 3.5$).

median with AFC is less biased compared to the inference performance without AFC shown in Table 2.3. The empirical coverage is greater than the true 90% confidence level for all 4 settings.

**Table 2.4:** Inference performances with AFC for nonlinear data generating algorithm.

| **True $\mu$** | **Coverage** | **Expected CI Length** | **Expected Mean** | **Expected Median** |
|---|---|---|---|---|
| 1 | 0.95 | 3.10 | 1.44 | 1.06 |
| 2 | 0.95 | 4.07 | 2.55 | 2.18 |
| 3 | 0.97 | 3.43 | 3.22 | 2.99 |
| 4 | 0.94 | 3.30 | 3.98 | 3.89 |

### 2.4.3 Many means

For this simulation, we are able to calculate the analytical form of the inverse function. We show the improvement of AFC by comparing the inference performances with and without AFC to select the valid threshold fiducial samples.

Considering the following data generate equation: $\boldsymbol{x} = \mu + \boldsymbol{z}$, with $\boldsymbol{x} \in \Re^m$, $\mu \in \Re^m$, and $\boldsymbol{z} \in \Re^m$. And the parameter of interests is the square of the $l$-2 norm of $\mu$, $\|\mu\|^2$, instead of $\mu$ itself. We will first generated fiducical samples of $\mu$, and further generated admissible $\|\mu\|^2$. For this case, the inverse function of is simple: $\mu = g(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x} - \boldsymbol{z}$. But when plug in the true $\boldsymbol{x}$ and $\boldsymbol{z}^*$, an independent copy of $\boldsymbol{z}$, into the inverse function $g$ and carried the AFC algorithms, we will accept every proposed $\mu$, since $\hat{\boldsymbol{x}} = f(\hat{\mu}, \boldsymbol{z}^*) = \hat{\mu} + \boldsymbol{z}^* = g(\boldsymbol{x}, \boldsymbol{z}^*) + \boldsymbol{z}^* = \boldsymbol{x} - \boldsymbol{z}^* + \boldsymbol{z}^* = \boldsymbol{x}$. We proposed the following estimation $\hat{\mu} = \|\tilde{\mu}\| * (\boldsymbol{x}/\|\boldsymbol{x}\|)$ with $\tilde{\mu} = \boldsymbol{x} - \boldsymbol{z}^*$.

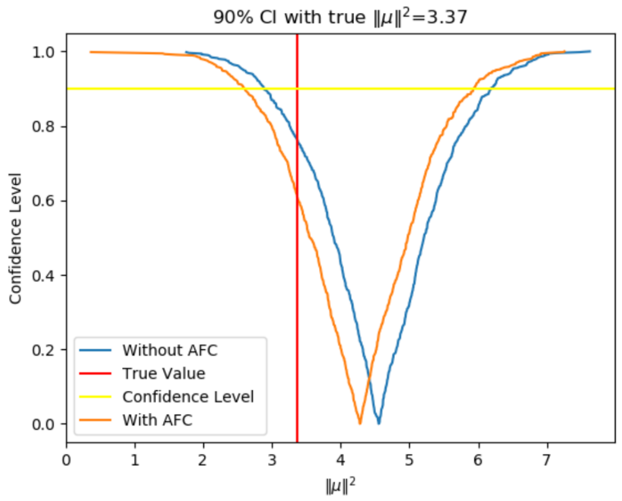


**Figure 2.7:** Confidence curves with and without AFC.

**Table 2.5:** Inference performances for many means example. ($\|\mu\|^2 = 3.37$)

| If AFC | Coverage | Expected CI Length | Expected Mean | Expected Median |
|---|---|---|---|---|
| No | 0.85 | 3.12 | 3.91 | 3.91 |
| Yes | 0.95 | 3.29 | 3.65 | 3.64 |

As shown in Table 3, the empirical coverage increase 10%, and the expected median and the expected mean is more accurate with AFC.

### 2.4.4 Biological oxygen demand problem

Here we further test our AFC-corrected FAE solution on a non-linear regression algebraic model for Biological Oxygen Demand (BOD) (Bardsley et al., 2014) . The BOD model is often used for simulating the saturation of the growth of the observed response y that corresponds to a given variable x. The corresponding data generating algorithm is

$$\mathbf{y} = f(\mu, \mathbf{x}) = t_0(1 - e^{-t_1\mathbf{x}}) + \mathbf{z} \tag{2.5}$$

where the parameters $\mu = (t_0, t_1)$ are two scalars to estimate; the observed synthetic data have five design points $\mathbf{x} = (2, 4, 6, 8, 10)$ and dependent observations $\mathbf{y} = (0.152, 0.296, 0.413, 0.482, 0.567)$. The observation errors $\mathbf{z} = (z_1, \ldots, z_5)$ are assumed to be independent and identically distributed mean zero Gaussian errors with the standard deviation $\sigma > 0$. Following (Bardsley et al., 2014) we fixed $\sigma = 0.015$.

To train the FAE for the BOD model, we simulate 120,000 samples, with 96,000 as training dataset and 24,000 as validation dataset. Each data frame indexed by k contains three parts, the parameters $\mu_k = (t_{0,k}, t_{1,k})$, with $t_{0,k}$ from the Uniform$(0.4, 1.2)$ distribution and $t_{1,k}$ from the Uniform$(0.000001, 0.2)$ distribution, the random error $\mathbf{z}_k$ from $N_5(\mathbf{0}, \sigma^2\mathbf{I}_5)$, and the corresponding $\mathbf{y}_k$ generated from the data generating algorithm (2.5). Notice that these $\mu_k$ serve as the true parameter for training purpose and are generated around the least square estimators of the parameters $(\hat{t_0} = 0.90110, \hat{t_1} = 0.09863)$. Our encoder consists of 8 shared fully-connected layers that learn code from the observed data $(\mathbf{x}, \mathbf{y}_k)$ and the corresponding $\mathbf{z}_k$; then 3 fully-connected layers map from the code space to $t_0$, and in parallel one fully-connected layer maps from the code to $t_1$. We use ReLU as the activation function and Adam as the optimizer. The FAE converges after 10 epochs.

We used four different methods to analyze the BOD data with the resulting distributions reported in Figure 2.8. The FAE with AFC is in panel a); the GFD using Theorem 1 implemented using Hamiltonian Monte Carlo in STAN (Stan Development Team, 2020) is in panel b); the parametric bootstrap (Tibshirani and Efron, 1993) is in panel c); and the Bayesian solution of (Bardsley et al., 2014) using Metropolis-Hastings algorithm with a simplified fiducial-like proposal,

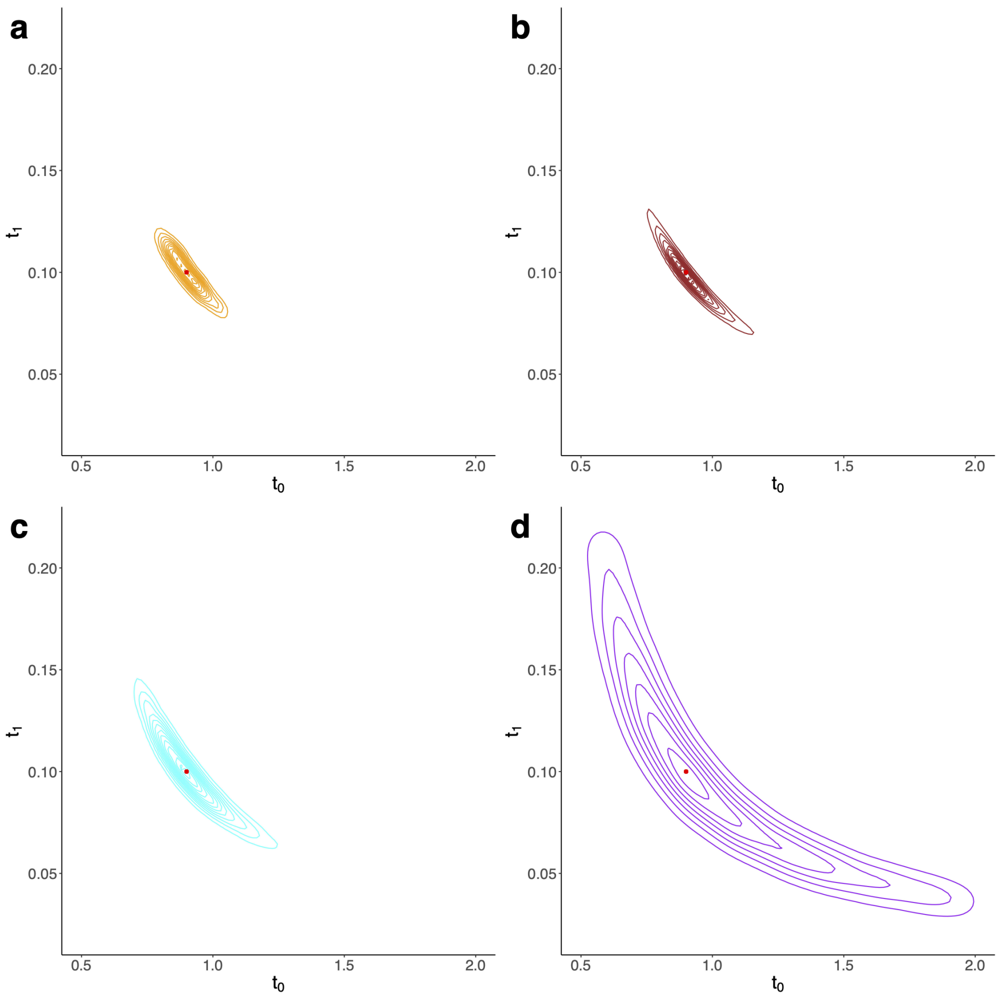


**Figure 2.8:** Contour plots of the joint posteriors for the BOD example. (a) FAE estimation with AFC algorithm (orange); (b) fiducial estimation implemented via HMC (brown); (c) parametric bootstrap (cyan); (d) RTO (Bayesian) solution (magenta). The red dots near the center of the contours represent the true parameters ($t_0 = 0.9, t_1 = 0.1$) that generated the observed data.

called randomize-then-optimize (RTO), is in panel d). Notice that all methods yield a "banana-shaped" distribution centered around the true parameters ($t_1 = 0.9, t_2 = 0.1$), indicated by the red dots in Figure 2.8. The AFC-corrected FAE solution (orange contour) achieve the thinnest banana-shape among all, while the RTO solution is the widest. Furthermore, we report confidence curves based on the marginal distributions of the two parameters separately in Figure 2.9. Again our AFC-corrected FAE solution is the thinnest among the four methods. Both the thinnest banana shape and confidence curve indicate that our AFC-corrected FAE solution is the most concentrated near the truth.

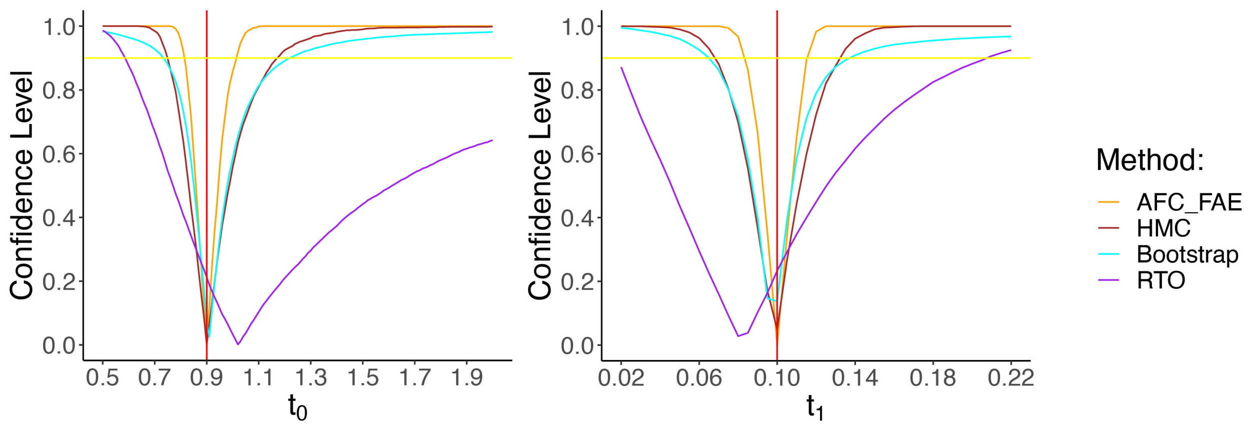


**Figure 2.9:** Confidence curves of marginal distributions for the BOD example. The colors associated with the methods are the same as in Figure 2.8. The red vertical lines are the true parameters ($t_0 = 0.9, t_1 = 0.1$) that generated the observed data. The yellow horizontal line indicates the 90% confidence intervals.

In summary, we have shown how to use AFC with and without the analytical form of the inverse function, and by implementing AFC we improved the inference performances of GFI. Our FAE provides an accurate approximation of the inverse function.

## 2.5 Discussion

In this chapter, we first proposed a fiducial autoencoder for the circumstance in which the analytical form of the inverse function is not available or the marginal fiducial density is intractable. The universal approximation theorem provides theoretical guarantees for the approximation performance of our FAE, and our simulations further validate our approach. The proposed FAE can accurately approximate the inverse function, and it can be efficiently combined with the AFC algorithm to provide valid and accurate inferences of the true parameters. The AFC algorithm is similar to ABC and provides an insight into the relationship between Bayesian and fiducial distribution; the use of a prior versus solving an optimization problem when proposing $\mu$.

For modern machine learning and deep learning communities, data are usually implicitly modeled, while for fiducial inference, we explicitly model the modeling mechanism behind the data through our data generating algorithm. Under the FAE framework, we are combining those two approaches: we incorporate the data generating algorithm as a decoder (explicitly model the forward process); we keep deep neural network (implicit model) as an encoder to computationally solve the inverse problem. Furthermore, FAE might help the deep learning community to understand the neural work through our fiducial autoencoder. For example, a neural network is often regarded as a non-linear regression tool, but one main difficulty for DL is that the true function is unknown. When the inverse function has an analytical solution, we can quantify the biases in a certain sense. Thus studying the theoretical properties of FAE could potentially be helpful to understand the neural network.

One limitation of our AFC-corrected FAE solution is that the inference results are sensitive to the training data FAE learned from. Neural networks are known for their ability to perform well on the similar data it has seen before. If most of the parameters of our simulated training data are far from the true parameter of the observed data, or FAE does not see enough similar training data as the true parameters of the observed data, FAE, even with AFC, might provide a biased estimation. A simple remedy for that is to generate the training data around the least square estimation of the parameter. In other words, we only require the FAE to learn the inverse function around the true parameter, instead of the whole parameter space, which can be much harder.

We have focused on cases with completely known data-generating algorithm and our pre-defined neural network architecture (fully connected neural networks) in the numerical experiments. The main purpose of these studies is to demonstrate the approximation performance of the FAE and the validity of the AFC algorithm. Thus we did not tune the best neural network architecture for the encoder part of the FAE. Note we do not have a specific requirement for the network structure of encoder. Any standard deep neural networks can be used to construct an encoder. However, to learn a complex inverse function inevitably requires a more sophisticated construction of encoder. This can be a potential issue, but rapid development of deep learning should provide new elegant architectures that increase the FAE applicability. The competitive performance of AFC corrected FAE solution both in terms of efficiency and accuracy suggests that this is a promising area for future research.

CHAPTER 3

# SMNN: Batch Effect Correction for Single-cell RNA-seq data via Supervised Mutual Nearest Neighbor Detection

## 3.1 Introduction

An ever-increasing amount of single cell RNA-sequencing (scRNA-seq) data has been generated as scRNA-seq technologies mature and sequencing costs continue dropping. However, large scale scRNA-seq data, for example, those profiling tens of thousands to millions of cells (such as the Human Cell Atlas Project (Rozenblatt-Rosen et al., 2017), almost inevitably involve multiple batches across time points, laboratories, or experimental protocols. The presence of batch effect renders joint analysis across batches challenging (Chen and Zhou, 2017; Stegle et al., 2015). Batch effect, or systematic differences in gene expression profiles across batches, not only can obscure the true underlying biology, but also may lead to spurious findings. Thus, batch effect correction, which aims to mitigate the discrepancies across batches, is crucial and deemed indispensable for the analysis of scRNA-seq data across batches (Stuart and Satija, 2019a).

Because of its importance, a number of batch effects correction methods has been recently proposed and implemented. Most of these methods, including limma (Smyth, 2005), ComBat (Johnson et al., 2007), and svaseq (Leek, 2014) , are regression-based. Among them, limma and ComBat explicitly model known batch effect as a blocking term. Because of the regression framework adopted, standard statistical approaches to estimate the regression coefficients corresponding to the blocking term can be conveniently employed. In contrast, svaseq is often used to detect underlying unknown factors of variation, for instance, unrecorded differences in the experimental protocols. svaseq first identifies these unknown factors as surrogate variables and subsequently corrects them. For these regression-based methods, once the regression coefficients are estimated or the unknown factors are identified, one can then regress out these batch effects accordingly, obtaining residuals that will serve as the batch-effect corrected expression matrix for further analyses. These methods have become standard practice in the analysis of bulk RNA-seq data. However, when it comes to scRNA-seq data, one key underlying assumption behind these methods, that the cell composition within each batch is identical, might not hold. Consequently, estimates of the coefficients might be inaccurate. As a matter of fact, when applied to scRNA-seq data, the corrected results derived from these methods widely adopted for bulk RNA-seq data might be even inferior to raw data without no correction, in some extreme cases (Haghverdi et al., 2018).

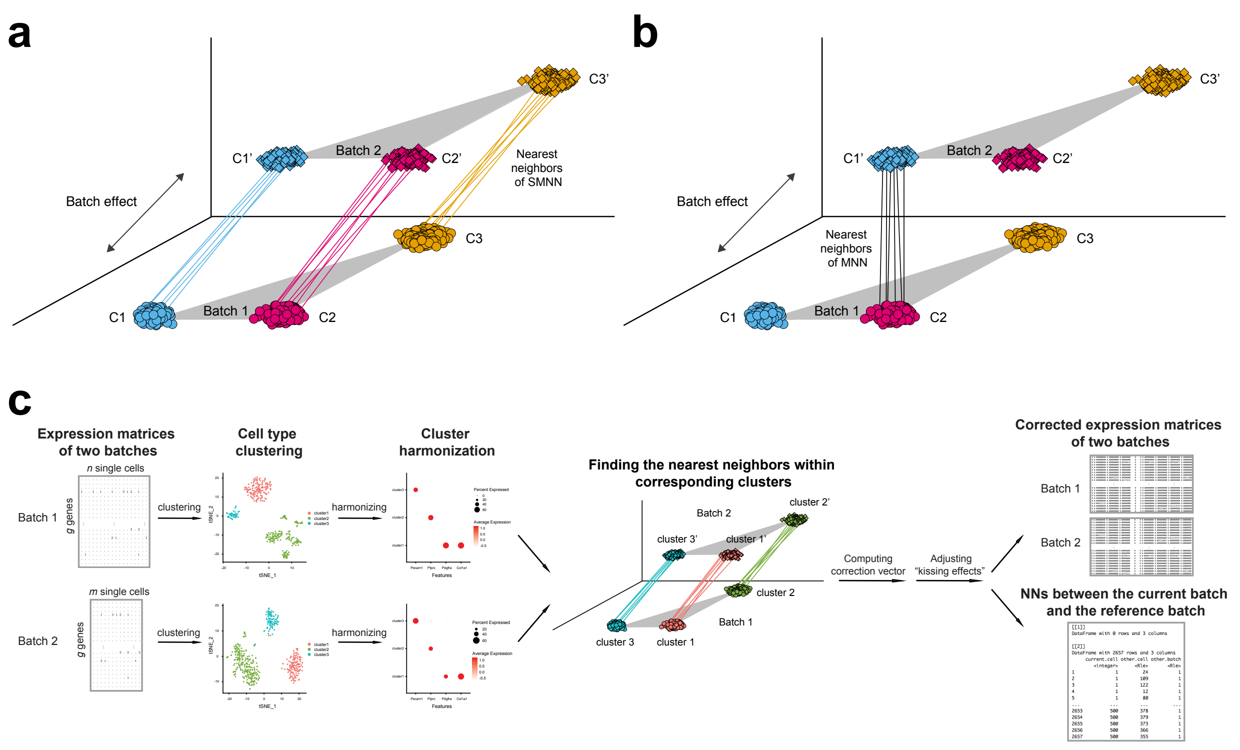
To address the heterogeneity and high dimensionality of complex data, several dimension-reduction approaches have been adopted. An incomplete list of these strategies includes principal component analysis (PCA), autoencoder, or force-based methods such as t-distributed stochastic neighbor embedding (t-SNE) (Van Der Maaten, 2014). Through those dimension reduction techniques, one can project new data onto the reference dataset using a set of landmarks from the reference (Haghverdi et al., 2018; Nestorowa et al., 2016; Spitzer et al., 2015; Stuart and Satija, 2019b) to remove batch effects between any new dataset and the reference dataset. Such projection methods require the reference batch contains all the cell types across batches. As one example, Spitzer et al. (Spitzer et al., 2015) employed force-based dimension reduction and showed that leveraging a few landmark cell types from bone marrow (the most appropriate tissue in that it provides the most complete coverage of immune cell types) allowed mapping and comparing immune cells across different tissues and species. When applied to scRNA-seq data, however, these methods suffer when cells from a new batch fall out of the space inferred from the reference. Furthermore, determining the dimensionality of the low dimensional manifolds is still an open and challenging problem. To address the limitations of existing methods, two recently developed batch effect correction methods, MNN and Seurat v3, adopt the concept of leveraging information of mutual nearest neighbors (MNN) across batches (Haghverdi et al., 2018; Stuart and Satija, 2019b), and demonstrate superior performance over alternative methods (Haghverdi et al., 2018; Stuart and Satija, 2019b). However, this MNN-based strategy ignores cell type information and suffers from potentially mismatching cells from different cell types/states across batches, which may lead to undesired correction results. For example, under the scenario depicted in Fig. 3.1b, MNN leads to cluster 1 (C1) and cluster 2 (C2) mis-corrected due to mismatching single cells in the two clusters/cell-types across batches.

To address the above issue, here we present SMNN, a supervised machine learning method that explicitly incorporates cell type information. SMNN performs nearest neighbor searching within the same cell type, instead of global searching ignoring cell type labels (Fig. 3.1a). Cell type information, when unknown a priori, can be inferred via clustering methods (Duò et al., 2018; Kiselev et al., 2019; Sun et al., 2019; Zhu et al., 2019).

## 3.2 Methodology

### 3.2.1 SMNN framework

The motivation behind our SMNN is that single-cell cluster or cell type information has the potential aid the identification of most relevant nearest neighbors and subsequently improve batch effect correction. A preliminary clustering before any correction can provide knowledge regarding cell composition within each batch, which serves as the cellular correspondence across batches (Fig. 3.1a). With this clustering information, we can refine the nearest neighbor searching space within a certain population of cells that are of the same or similar cell type(s) or state(s) across all batches.



```
[[1]]
DataFrame with 0 rows and 3 columns

[[2]]
DataFrame with 2657 rows and 3 columns
     current.cell other.cell other.batch
        <integer>      <Rle>       <Rle>
1               1         24           1
2               1        109           1
3               1        122           1
4               1         12           1
5               1         80           1
...           ...        ...         ...
2653          500        378           1
2654          500        379           1
2655          500        373           1
2656          500        366           1
2657          500        355           1
```

**Figure 3.1: Overview of SMNN.** Schematics for detecting mutual nearest neighbors between two batches under a non-orthogonal scenario (a) in SMNN; and (b) in MNN. (c) Workflow of SMNN. Single cell clustering is first performed within each batch using Seurat v3; and then SMNN takes user-specified marker gene information for each cell type to match clusters/cell types across batches. With the clustering and cluster-specific marker gene information, SMNN searches mutual nearest neighbors within each cell type and performs batch effect correction accordingly.

SMNN takes a natural two-step approach to leverage cell type label information for enhanced batch effect correction (Fig. 3.1c). First, it takes the expression matrices across multiple batches as input, and performs clustering separately for each batch. Specifically, in this first step, SMNN uses Seurat v3 (Butler et al., 2018) where dimension reduction is conducted via principal component analysis (PCA) to the default of 20 PCs, and then graph-based clustering follows on the dimension-reduced data with resolution parameter of 0.9 (Huh et al., 2019; Yang et al., 2019). Obtaining an accurate matching of the cluster labels across batches is of paramount importance for subsequent nearest neighbor detection. SMNN requires users to specify a list of marker genes and their corresponding cell type labels to match clusters/cell types across batches. We hereafter refer to this cell type or cluster matching as cluster harmonization across batches. Because not all cell types are necessarily shared across batches, and no prior knowledge exists regarding the exact composition of cell types in each batch, SMNN allows users to take discretion in terms of the marker genes to include, representing the cell types that are believed to be shared across batches.

Based on the marker gene information, a harmonized label is assigned to every cluster identified across all the batches according to two criteria: the percentage of cells in a cluster expressing a certain marker gene and the average gene expression levels across all the cells in the cluster. After harmonization, cluster labels are unified across batches. This completes step one of SMNN. Note that if users have a priori knowledge regarding the cluster/cell-type labels, the clustering step could be bypassed completely.

With the harmonized cluster or cell type label information obtained in the first step, SMNN, in the second step, searches mutual nearest neighbors only within each matched cell type between the first batch (which serves as the reference batch) and any of the other batches (the current batch), and performs batch effect correction accordingly. Compared to MNN or Seurat v3, where the mutual nearest neighbors or anchor cells are searched globally, SMNN identifies neighbors from the same cell population or state. After mutual nearest neighbors are identified, similar to MNN, SMNN first computes batch effect correction vector for each identified pair of cells, and then calculates, for each cell, the cell-specific correction vectors by exploiting a Gaussian kernel to obtain a weighted average across all the pair-specific vectors with mutual nearest neighbors of the cell under consideration. The correction vectors obtained from shared cell-types will be applied to correct all cells including those belonging to batch-specific cell types.

The correction vector obtained from shared cell-types will be applied to correct all the cells, including the cells in batch-specific clusters. Specifically, for each cell $x$ in the target batch (which will be corrected), whether it falls into a batch-shared cluster or a batch-specific cluster, SMNN calculates a cell-specific correction vector using the same formula below. The cell-specific correction vector $\vec{u}$ for any target cell $x$ is a weighted sum of vector differences $\vec{v}_{ml}$ across all identified MNN pairs (without loss of generosity, we assume m is from the target batch and l from the reference batch. Note that in SMNN, the two cells m and l in each MNN pair belong to the same cluster/cell-type across the two batches).

$$\vec{u}(x) = \frac{\sum_{MNN} \text{pairs}\, \vec{v}_{ml} W(x,m)}{\sum_{MNN} \text{pairs}\, W(x,m)}$$

The Gaussian kernel weights $W(x,m)$ gives higher weights to vector differences ($\vec{v}_{ml}$'s) involving a cell $m$ in closer proximity with the target cell $x$. The smaller the distance, the larger of the weight. Note that a cell $m$ may appear multiple times and contributes to multiple terms in the weighted sum if it has multiple MNNs identified by our SMNN. Also note that if cell $x$ belongs to a batch-specific cluster, itself would not contribute to any of the terms because it would not have any MNNs.

Each cell's correction vector is further scaled according to the cell's location in the space defined by the correction vector, and standardized according to quantiles across batches, in order to eliminate "kissing effects". "Kissing effects" refer to the phenomenon that only the surfaces of cell-clouds across batches are brought in contact (rather than entire clouds fully merged), commonly observed with naïve batch effect correction, (Haghverdi et al., 2018). At the end of the second step, SMNN returns the batch-effect corrected expression matrix matrix including all genes from the input matrix for each batch, as well as the information regarding nearest neighbors between the reference batch and the current batch under correction. This step is carried out for every batch other than the reference batch so that all batches are corrected to the same reference batch in the end.

SMNN is implemented in R, and freely available at `https://yunliweb.its.unc.edu/SMNN/` and `https://github.com/yycunc/SMNN`.

## 3.3 Simulation

We simulated two scenarios, orthogonal and non-orthogonal, to compare the performance of MNN and SMNN. The difference between the two scenarios lies in the directions of the true underlying batch effect vectors with respect to those of the biological effects (see details in Section 3.3.3).

### 3.3.1 Baseline simulation

Our baseline simulation framework, similar to that adopted in the MNN paper, contains two steps: Firstly, data are initially generated in low (specifically three) dimensional biological space. Data in each batch independently generated from a Gaussian mixture model to represent a low dimensional biological space, with each component in the mixture corresponding to one cell type. Specifically, we considered two batches $X_k$ and $Y_l$, each of which follows a three-component Gaussian mixture model in a three-dimensional space, representing the low (here three) dimensional biological space.

$$X_k \sim \sum_{i=1}^{3} w_{1i} N\left(\mu_{1i}, \boldsymbol{I}_3\right), \text{ with } \sum_{i=1}^{3} w_{1i} = 1, \text{ and } w_{11}, w_{12}, w_{13} \geq 0, \text{ for } k = 1, 2, \ldots, n_1$$
$$Y_l \sim \sum_{j=1}^{3} w_{2j} N\left(\mu_{2j}, \boldsymbol{I}_3\right), \text{ with } \sum_{j=1}^{3} w_{2j} = 1, \text{ and } w_{21}, w_{22}, w_{23} \geq 0, \text{ for } l = 1, 2, \ldots, n_2$$

where $\mu_{1i}$ is the three-dimensional vector specifying cell-type specific means for the i-th three cell types in the first batch, reflecting the biological effect; similarly for $\mu_{2j}$; $n_1$ and $n_2$ is the total number of cells in the first and second batch, respectively; $w_{1i}$ and $w_{2j}$ are the different mixing coefficients for the three cell types in the two batches; and $\boldsymbol{I}_3$ is the three dimensional identity matrix with diagonal entries as ones and the rest entries as zeros. In our simulations, we set $n_1 = 1000$, $n_2 = 1100$ and

$$(w_{11}, w_{12}, w_{13}) = (0.3, 0.5, 0.2)$$
$$(w_{21}, w_{22}, w_{23}) = (0.25, 0.5, 0.25)$$

Secondly, we projected the low dimensional data with batch effect to the high dimensional gene expression space. We mapped both datasets to $G = 50$ dimensions by linear transformation using the same random Gaussian matrix $\boldsymbol{P}$, to simulate high-dimensional gene expression profiles.

$$\widetilde{X_k} = \boldsymbol{P} X_k, \text{ for } k = 1, 2, \ldots, n_1$$
$$\widetilde{Y_l} = \boldsymbol{P} Y_l, \text{ for } l = 1, 2, \ldots, n_2$$

$\boldsymbol{P}$ is a $G \times 3$ Gaussian random matrix with each entry simulated from the standard normal distribution.

### 3.3.2 Introduction of batch effects

In the MNN paper (Haghverdi et al., 2018), batch effects were directly introduced in the high dimensional gene expression space. Specifically, a Gaussian random vector $b = (b_1, b_2, \ldots, b_G)^T$ was simulated and added to the second dataset via the following:

$$X_{Observed,k} = \widetilde{X_k} + \varepsilon_{1,k}, \text{ for } k = 1, 2, \ldots, n_1$$

$$Y_{Observed,l} = \widetilde{Y}_l + b + \varepsilon_{2,l}, \text{ for } l = 1, 2, \ldots, n_2$$

where $\epsilon_{1,k}$ and $\epsilon_{2,l}$ are independent random noises added to the expression of each "gene" for each cell in the two batches.

In our simulations, we adopted a slightly different approach: we introduced batch effects in the low dimensional biological space. Specifically, we simulated a bias vector $c = (c_1, c_2, c_3)^T$ in the biological space:

$$X_{Observed,k} = \widetilde{X_k} + \varepsilon_{1,k} = \boldsymbol{P} X_k + \varepsilon_{1,k}, \text{ for } k = 1, 2, \ldots, n_1$$

$$Y_{Observed,l} = \widetilde{Y_{SMNN,l}} + \varepsilon_{2,l} = \boldsymbol{P}\left(Y_l + c\right) + \varepsilon_{2,l} = \boldsymbol{P} Y_l + Pc + \varepsilon_{2,l}, \text{ for } l = 1, 2, \ldots, n_2$$

Our simulation framework can be viewed as a reparametrized version of the model in Haghverdi et al. (Haghverdi et al., 2018). For each batch effect b of the model in Haghverdi et al. (Haghverdi et al., 2018), there exist multiple pairs of projection matrix P and vector c such that b=Pc, and for any vector c in our model, there is a corresponding vector b=Pc given a fixed projection matrix P. In particular, $(b)_l = (\boldsymbol{P}c)_l = \sum_{i=1}^{G} P_{li} c_i \sim N\left(0, \sum_{i=1}^{G} c_i^2\right)$. In other words, for any simulated setting in Haghverdi et al. (Haghverdi et al., 2018), we can find at least one equivalent setting in our model; and vice versa. Although our simulation framework is largely similar to that in Haghverdi et al. (Haghverdi et al., 2018), the two differ in the following two aspects:

First, the low-dimensional biological space is three-dimensional in ours and two-dimensional in Haghverdi et al. (Haghverdi et al., 2018). Second, we introduce batch effects c in low dimensional biological space and then projected to high dimensional space, while Haghverdi et al. (Haghverdi et al., 2018) directly introduce batch effects b in the high dimensional gene expression space. We made such changes so that we can simulate both the orthogonal and non-orthogonal scenarios in a more straightforward manner the extent of orthogonality can be controlled. The orthogonality is defined in the sense that biological differences (that is, mean difference between any two clusters/cell-types), are orthogonal to those from batch effects. Our framework allows flexible modeling of the biological effects and batch effects in the same low dimensional biological space and allow us to control the extent of orthogonality. Specifically, the batch effect c is added to mean vectors of three cell types in batch 1 to get the mean vectors of three cell types for batch

2.

$$\mu_{2i} = \mu_{1i} + c, \text{ for } i = 1, 2, 3$$

Note that $(\mu_{1j} - \mu_{1i})\, c = 0,$ for $i \neq j \in \{1, 2, 3\}$ represents the orthogonal scenario that variation from batch effect is orthogonal to mean difference between any two clusters/cell-types, and $(\mu_{1j} - \mu_{1i})\, c \neq 0,$ for $i \neq j \in \{1, 2, 3\}$ in the non-orthogonal case.

### 3.3.3 The two scenarios

As aforementioned, we considered two scenarios, orthogonal case and non-orthogonal case. Orthogonality is defined in the sense that biological differences (that is, mean difference between any two clusters/cell-types), are orthogonal to those from batch effects.

Leveraging the simulation framework described before (Section 3.3.2), we simulated two scenarios via the following:

a) In the orthogonal case, we set $c = (0, 0, 2)^T$

$$\mu_{11} = (5, 0, 0)^T, \mu_{12} = (0, 0, 0)^T, \mu_{13} = (0, 5, 0)^T$$
$$\mu_{21} = (5, 0, 2)^T, \mu_{22} = (0, 0, 2)^T, \mu_{23} = (0, 5, 2)^T$$

b) In the non-orthogonal case, we set $c = (0, 5, 2)^T$

$$\mu_{11} = (5, 0, 0)^T, \mu_{12} = (0, 0, 0)^T, \mu_{13} = (0, 5, 0)^T$$
$$\mu_{21} = (5, 5, 2)^T, \mu_{22} = (0, 5, 2)^T, \mu_{23} = (0, 10, 2)^T$$

### 3.3.4 Performance evaluation

MNN and SMNN share the goal to correct batch effects. Mathematically, using the notations introduced in Section 3.3.2, the goal translates into de-biasing vector c (which would be effectively reduced to b in the orthogonal case). Without loss of generality and following MNN, we treated the first batch as the reference and corrected the second batch $Y_{Observed,l} : l = 1, \ldots, n_2$ to the first batch $X_{Observed,k} : k = 1, \ldots, n_1$. Denote the corrected values from MNN and SMNN as $\left\{\widehat{Y_{MNNN}} : l = 1, \ldots, n_2\right\}$ and $\left\{\widehat{Y_{SMNNN}} : l = 1, \ldots, n_2\right\}$, respectively.

To measure the performance of the two correction methods, we utilize the Frobenius norm to define the loss function:

$$L(\hat{Y}, \tilde{Y}) = \|\widetilde{\boldsymbol{Y}} - \hat{Y}\|_F = \sqrt{\sum_{l=1}^{n_2} \left\| \widetilde{Y_l} - \hat{Y_l} \right\|^2} = \sqrt{\sum_{l=1}^{n_2}\sum_{g=1}^{G} \left| \widetilde{Y_{l,g}} - \widehat{Y_{l,g}} \right|^2}$$

Note that $\tilde{Y}$ is the simulated true profiles (from section 3.3.1) before batch effects and noises are introduced in section 3.3.2. Since MNN conducts a cosine normalization to the input and the output, we use cosine-normalized $\tilde{Y}$ when calculating the above loss function.

### 3.3.5 Simulation results

Since MNN has been shown to excel alternative methods (Haghverdi et al., 2018; Stuart and Satija, 2019a) , we here focus on comparing our SMNN with MNN. We first compared the performance of SMNN to MNN in simulated data. In our simulations, SMNN demonstrates superior performance over MNN under both orthogonal and non-orthogonal scenarios (Fig. 3.2 - Fig. 3.6).

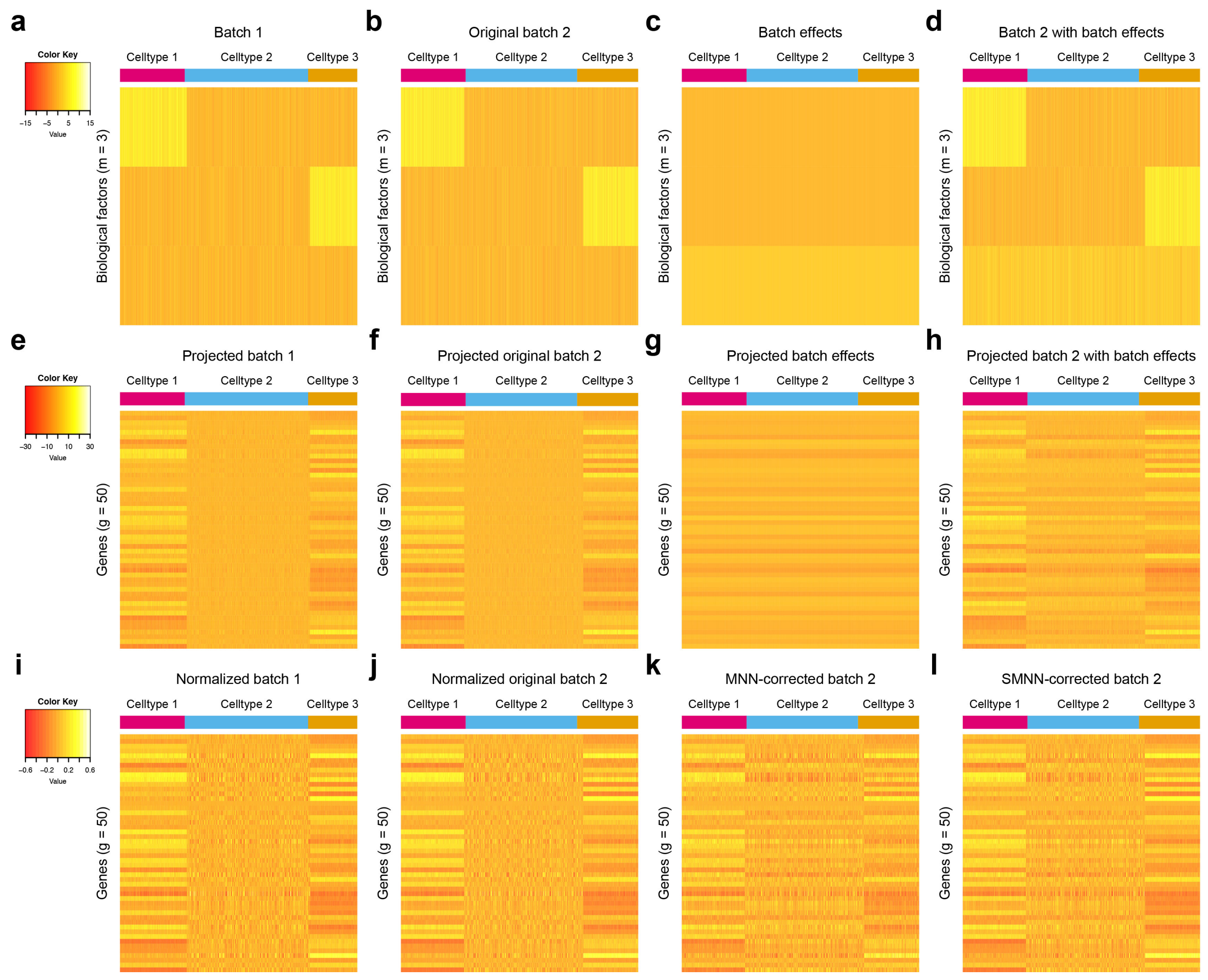


**Figure 3.2: Heatmap of gene expression matrices for simulated data under orthogonal scenario.** (a), (b), (c) and (d) show the 3-dimensional biological space with rows of each heatmap representing biological factors and columns corresponding to single cells. (e), (f), (g) and (h) show the high dimensional gene expression profiles with rows corresponding to genes and columns again representing single cells. (a), (e) and (i) correspond to the batch 1, and (b), (f) and (j) correspond to batch 2. (c) and (g) provide a visualization for the direction of batch effects in low-dimension biological space and high-dimension gene expression spaces, respectively. (d) and (h), sum of (b) and (c) and sum of (f) and (g) respectively, are "observed" data for cells in batch 2 in low and high dimensional space respectively. (i) and (j) are the cosine-normalized data for batch 1 and original batch 2. Note "original" is in the sense that no batch effects have been introduced to the data yet. (k) and (l) are the MNN and SMNN corrected results, respectively.

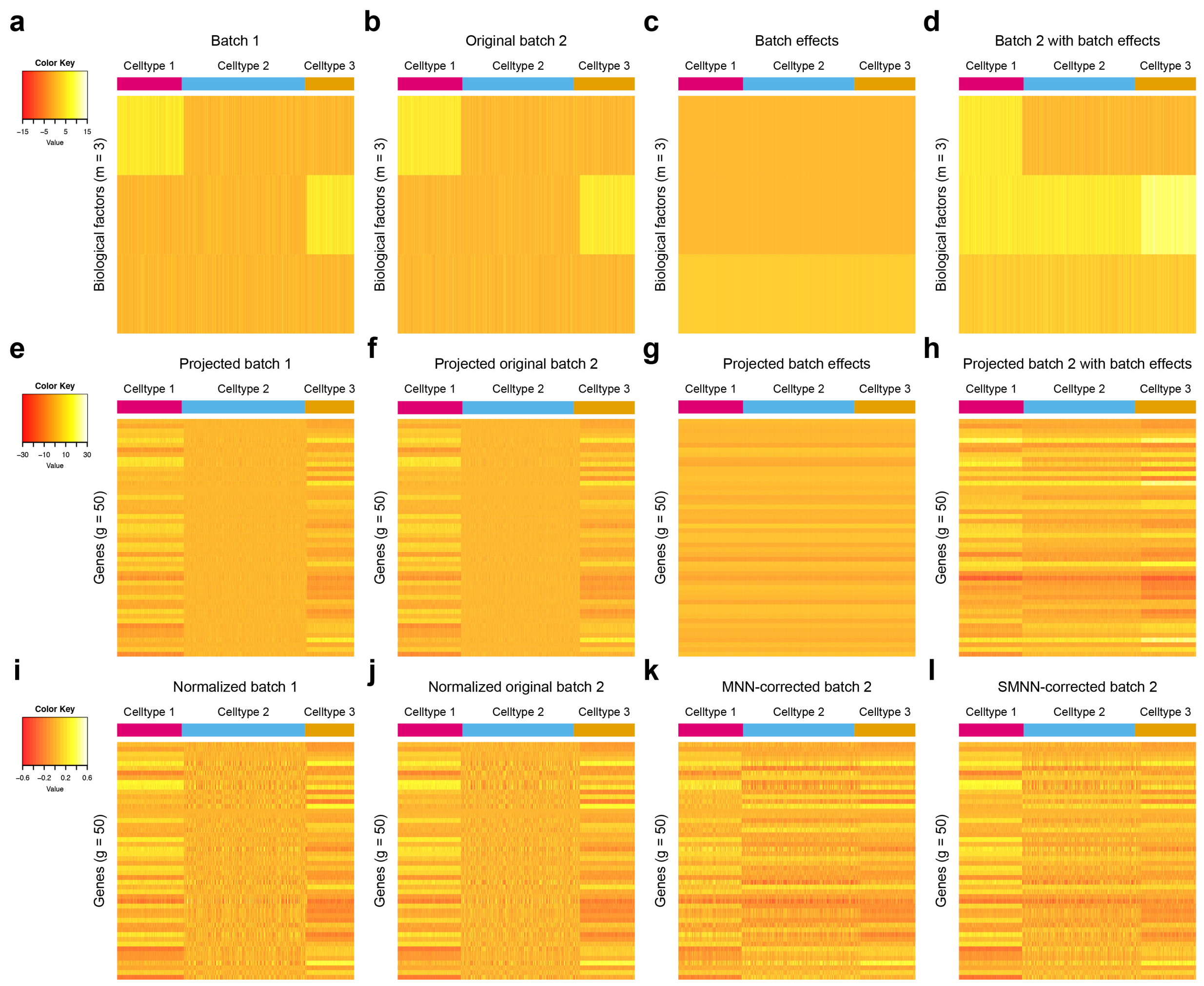


**Figure 3.3: Heatmap of gene expression matrices for simulated data under non-orthogonal scenario.** (a), (b), (c) and (d) show the 3-dimensional biological space with rows of each heatmap representing biological factors and columns corresponding to single cells. (e), (f), (g) and (h) show the high dimensional gene expression profiles with rows corresponding to genes and columns again representing single cells. (a), (e) and (i) correspond to the batch 1, and (b), (f) and (j) correspond to batch 2. (c) and (g) provide a visualization for the direction of batch effects in low-dimension biological space and high-dimension gene expression spaces, respectively. (d) and (h), sum of (b) and (c) and sum of (f) and (g) respectively, are "observed" data for cells in batch 2 in low and high dimensional space respectively. (i) and (j) are the cosine-normalized data for batch 1 and original batch 2. Note "original" is in the sense that no batch effects have been introduced to the data yet. (k) and (l) are the MNN and SMNN corrected results, respectively.

We show t-SNE plot for each cell type before and after MNN and SMNN correction under both the orthogonal and non-orthogonal scenarios. Under orthogonality, the two batches partially overlapped in the t-SNE plot before correction, suggesting that the variation due to batch effect was indeed much smaller than that due to biological effect. Both MNN and SMNN successfully mixed single cells from two batches (Fig. 3.4). However, for cell types 1 and 3, there were still some cells from the second batch left unmixed with those from the first batch after MNN correction (Fig.

3.4a and c). Under the non-orthogonal scenario, the differences between two batches were more pronounced before correction, and SMNN apparently outperformed MNN (Fig. 3.5), especially in cell type 1 (Fig. 3.5a).

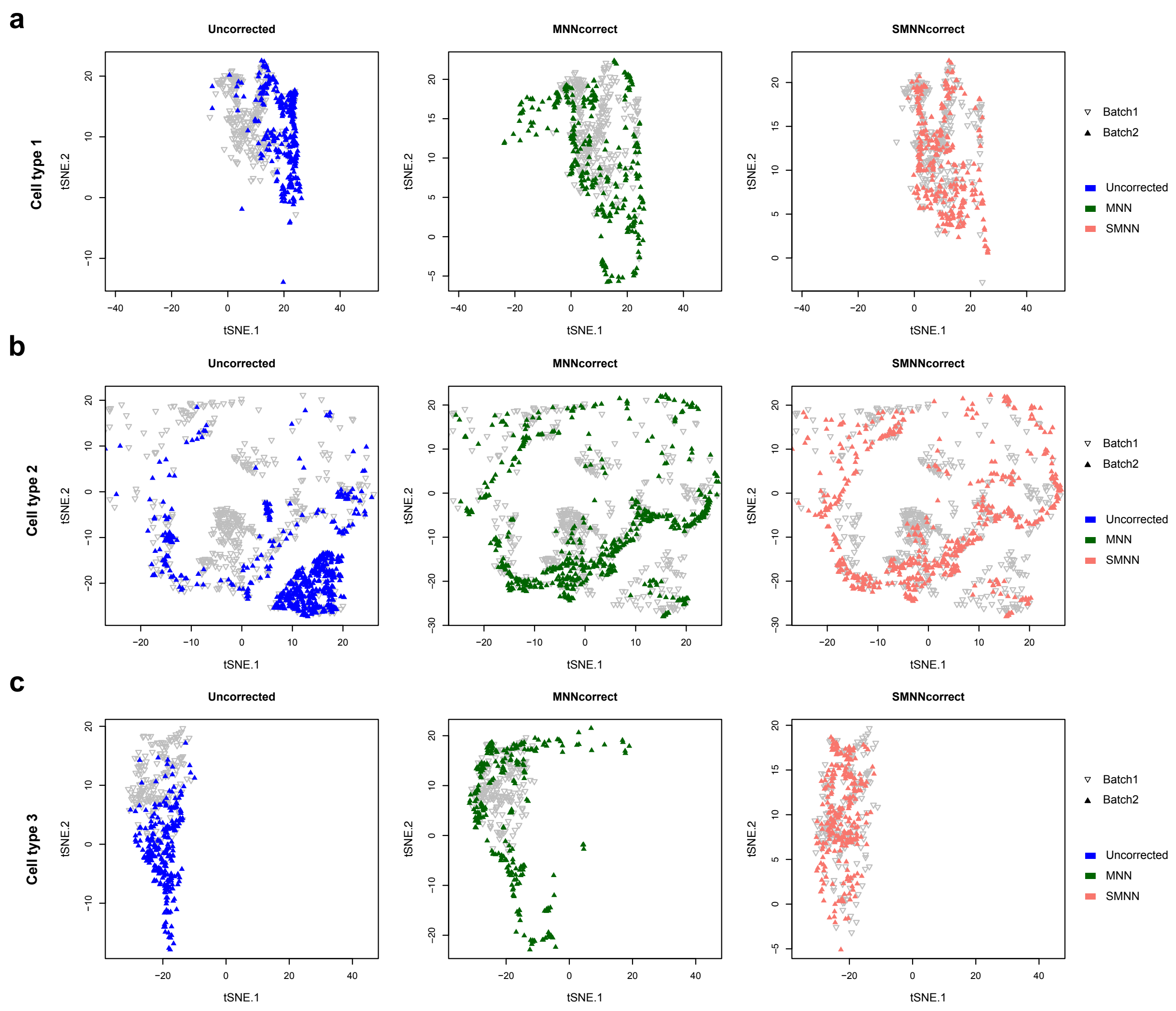


**Figure 3.4: t-SNE plots by cell type under the orthogonal scenario.** The first column shows the uncorrected data after cosine normalization; the second column shows the MNN corrected results and the last column shows the SMNN corrected results. (a), (b) and (c) correspond to three different cell types.

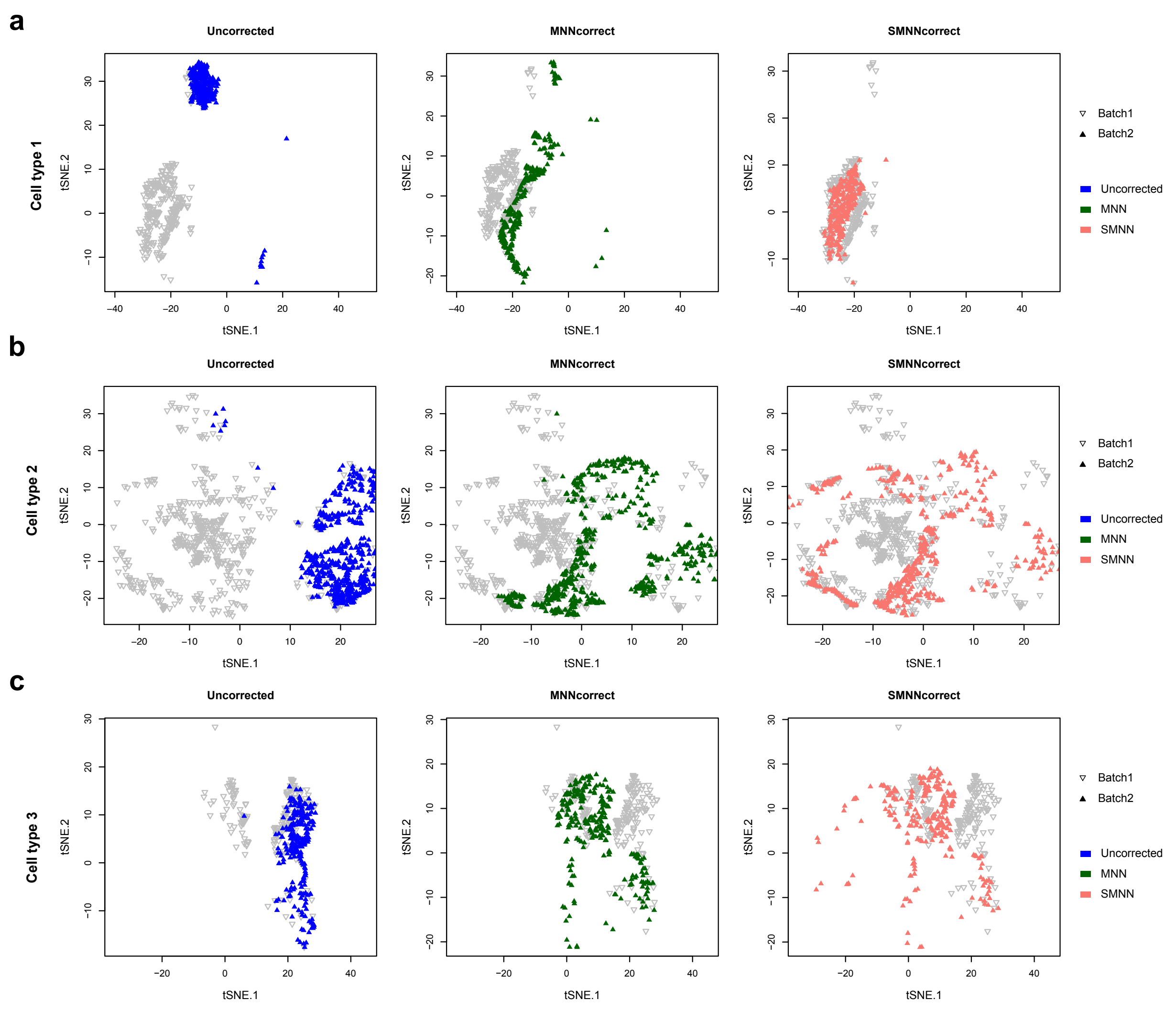


**Figure 3.5: t-SNE plots by cell type under the non-orthogonal scenario.** The first column shows the uncorrected data after cosine normalization; the second column shows the MNN corrected results and the last column shows the SMNN corrected results. (a), (b) and (c) corresponds to three different cell types.

Moreover, we also computed Frobenius norm distance (Van Loan and Golub, 1983) for each cell between its simulated true profile before introducing batch effects and after SMNN and MNN correction. The results showed an apparently reduced deviation from the truth after SMNN correction than MNN (Fig. 3.6). We have also simulated data using the original simulation framework in Haghverdi et al. (Haghverdi et al., 2018), which does not allow precise control of orthogonality and seems to simulate data closer to those under orthogonal cases (Appendix B: Fig. B.1a). Applying SMNN and MNN to such simualted data, we also found that SMNN showed slight advantages (Appendix B: Fig. B.1b) These results suggest that SMNN provides improved batch effect correction over MNN under both orthogonal and non-orthogonal scenarios.

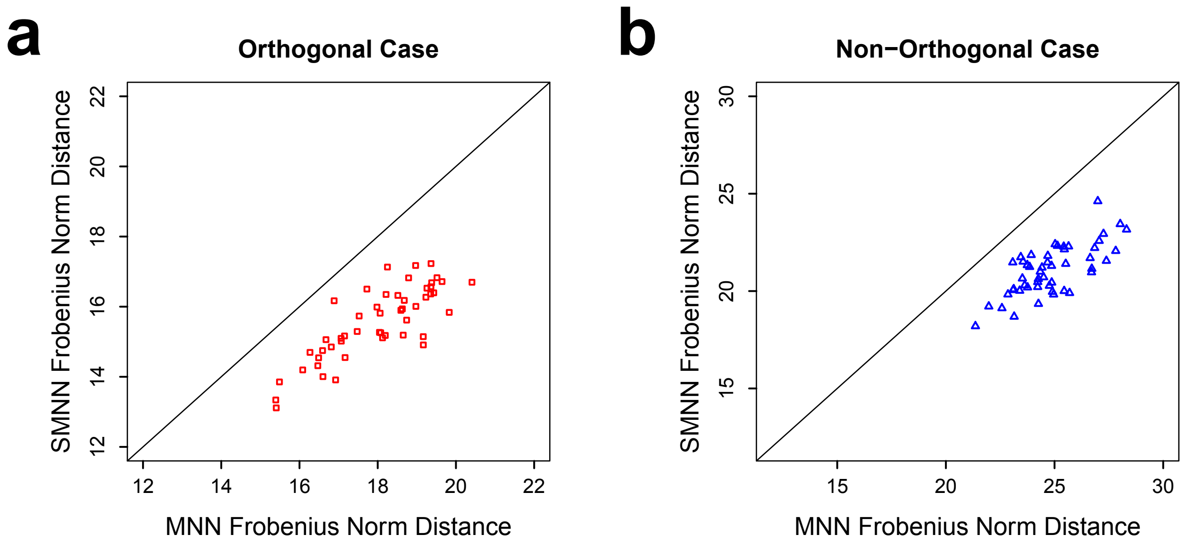


**Figure 3.6:** Frobenius norm distance between two batches after SMNN and MNN correction in simulation data under orthogonal (left) and non-orthogonal scenarios (right).

## 3.4 Real data benchmarking

To assess the performance of SMNN in real data, we first compared SMNN to alternative batch effect correction methods: MNN (Haghverdi et al., 2018), Seurat v3 (Butler et al., 2018), and LIGER (Welch et al., 2019) on two hematopoietic scRNA-seq datasets, generated using different sequencing platforms, MARS-seq and SMART-seq2 (Table 3.1) (Nestorowa et al., 2016; Paul et al., 2015). The first batch produced by MARS-seq consists of 1920 cells of six major cell types, and the second batch generated by SMART-seq2 contains 2730 of three cell types, where three cell types, CMP, GMP and MEP cells, are shared between these two batches (here the two datasets). Batch effect correction was carried out using all four methods, following their default instructions. Cell type labels were fed to SMNN directly according to the annotation from the original papers. To better compare the performance between MNN and SMNN, only the three cell types shared between the two batches were extracted for our downstream analyses. The corrected results of all the three cell types together, as well as for each of them separately, were visualized by UMAP using umap-learn method (McInnes et al., 2018). In order to qualify the mixture of single cells using both batch correction methods, we calculated: 1) F statistics under two-way multivariate analysis of variance (MANOVA) for merged datasets of the two batches. F statistics quantifies differences between batches, where smaller values indicating better mixing of cells across batches; and 2) the distance for the cells within each cell type in batch 2 to the centroid of the corresponding cell group in batch 1.

To measure the separation of cell types after correction, we additionally attempted to detect differentially expressed genes (DEGs) between different cell types in both SMNN and MNN corrected datasets. The corrected expression matrices of the two batches were merged and DEGs were detected by Seurat v3 using Wilcoxon rank sum test (Butler et al., 2018). Genes with an adjusted p-value $< 0.01$ were considered as differentially expressed. Gene ontology (GO) enrichment analysis was performed for the DEGs exclusively identified by SMNN using clusterProfiler (Yu et al., 2012). Because there is no ground truth for DEGs, we further identified DEGs between different cell types within corrected batch 2 and then compared them to those identified in uncorrected batch 1 and uncorrected batch 2, which supposedly are not affected by the choice of batch effect correction method. Precision was computed for each comparison. Furthermore, we performed batch effect correction on another two tissues/cell lines, pancreas (Grün et al., 2016; Muraro et al., 2016) human peripheral blood mononuclear cells (PBMCs) (Zheng et al., 2017), again using both SMNN and MNN. DEGs were detected between T cells and B cells in the merged PBMC and T cell datasets after SMNN and MNN correction, respectively. Furthermore, single cell clustering was applied to batch-effects corrected gene expression matrices in all the three real datasets following the pipeline described in MNN paper (Haghverdi et al., 2018). Cell type labels before correction were considered as ground truth and Adjusted Rand Index (ARI) (Hubert and Arabie, 1985) was employed to measure the clustering similarity before and after correction:

$$ARI\left(L_q, L_s\right) = \frac{\sum_{q,s}\begin{pmatrix} n_{qs} \\ 2 \end{pmatrix} - \left[\sum_q \begin{pmatrix} n_q \\ 2 \end{pmatrix} \sum_s \begin{pmatrix} n_s \\ 2 \end{pmatrix}\right] / \begin{pmatrix} n \\ 2 \end{pmatrix}}{\frac{1}{2}\left[\sum_q \begin{pmatrix} n_q \\ 2 \end{pmatrix} + \sum_s \begin{pmatrix} n_s \\ 2 \end{pmatrix}\right] - \left[\sum_q \begin{pmatrix} n_q \\ 2 \end{pmatrix} \sum_s \begin{pmatrix} n_s \\ 2 \end{pmatrix}\right] / \begin{pmatrix} n \\ 2 \end{pmatrix}}$$

where $n_q$ and $n_s$ are the single cell numbers in cluster q and s, respectively; $n_{qs}$ is the number of single cells shared between clusters q and s; and n is the total number of single cells. ARI ranges from 0 to 1, where a higher value represents a higher level of similarity between the two sets of cluster labels.

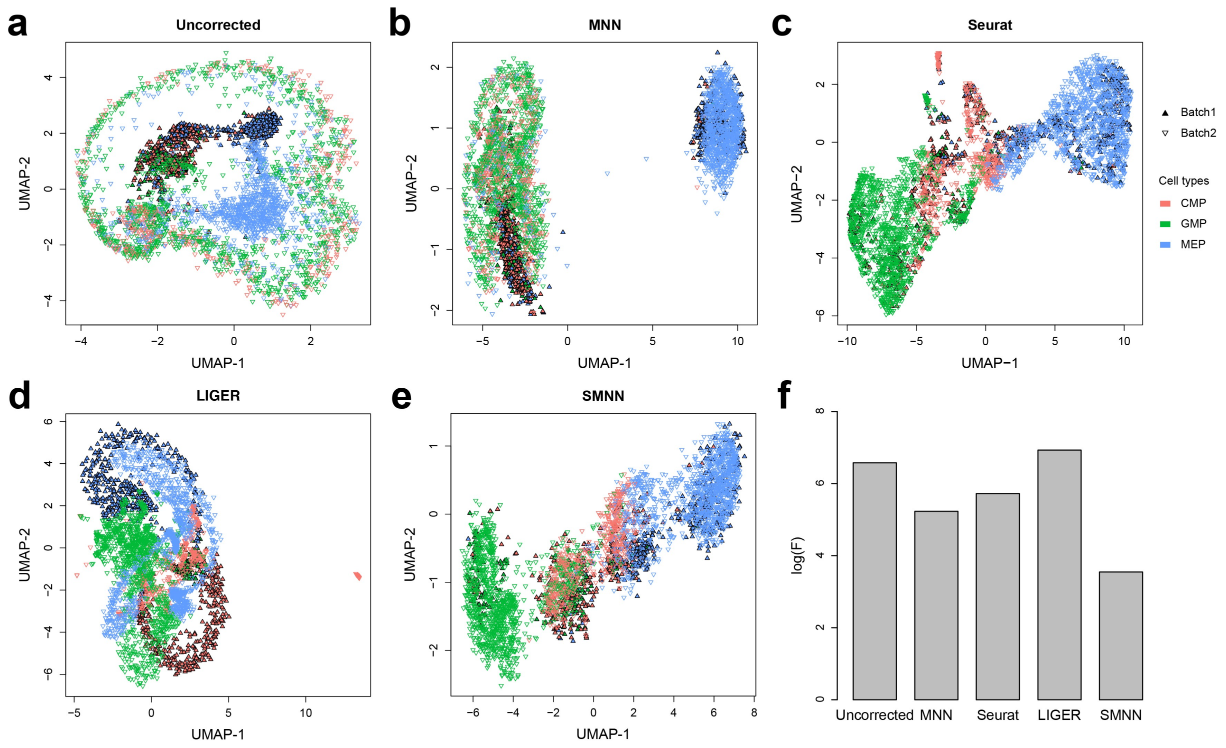


**Figure 3.7: Performance comparison between SMNN and MNN in two hematopoietic datasets.** (a) UMAP plots for two hematopoietic datasets before batch effect correction. Solid and inverted triangle represent the first and second batch, respectively; and different cell types are shown in different colors. (b-e) UMAP plots for the two hematopoietic datasets after correction with MNN, Seurat v3, LIGER, and SMNN. (f) Logarithms of F-statistics for merged data of the two batches.

For performance evaluation on two hematopoietic datasets using four methods: our SMNN, published MNN, Seurat v3 and LIGER. Fig. 3.7a-e shows UMAP plot before and after correction. Notably, all four methods can substantially mitigate discrepancy between the two datasets. Comparatively, SMNN better mixed cells of the same cell type across batches than the other three methods, and seemed to better position cells from batch-specific cell types with respect to other biologically related cell types (Fig. 3.11), especially for common myeloid progenitor (CMP) and megakaryocyte-erythrocyte progenitor (MEP) cells, which were wrongly corrected by MNN due to sub-optimal nearest neighbor search ignoring cell type information (Fig. 3.8).

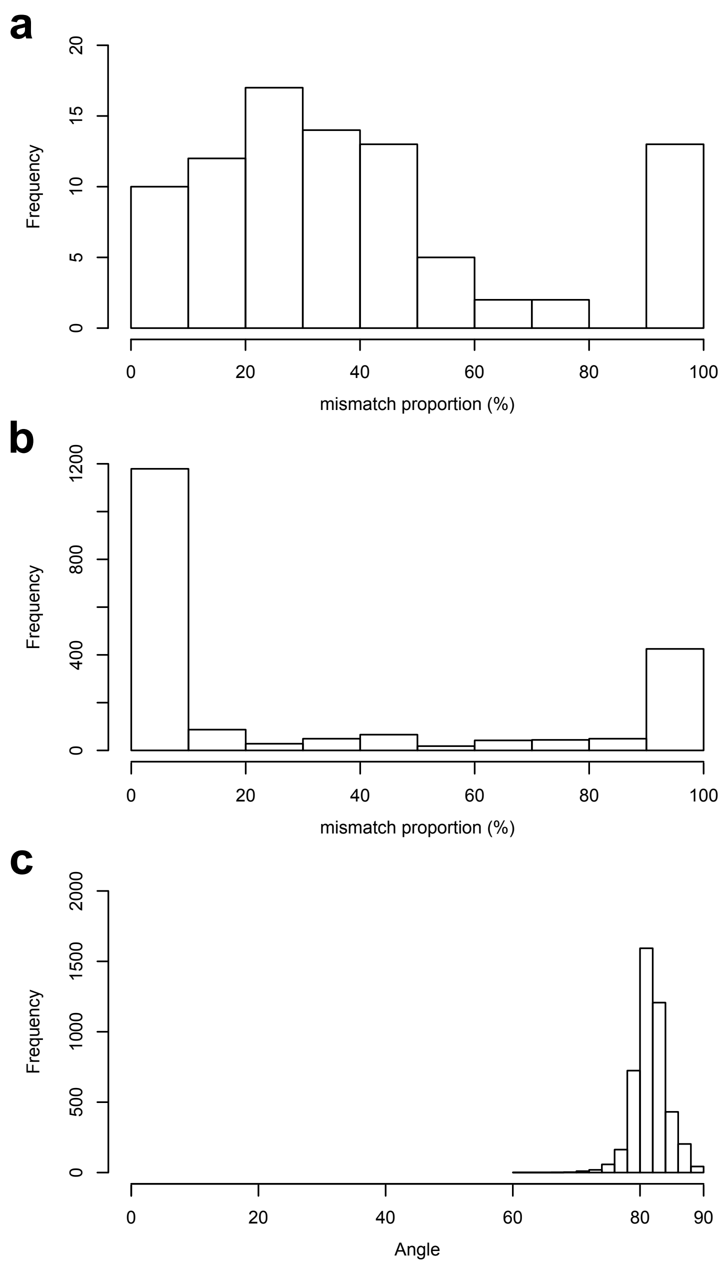


**Figure 3.8: Quantification of cell-type mismatching in MNN identified nearest neighbors.** (a) Histogram of the proportion of nearest neighbors from a mismatching cell type (for each cell), for two hematopoietic datasets. (b) Histogram of the proportion of nearest neighbors from a mismatching cell type (for each cell), for the 10X Genomics datasets. (c) Histogram of the angles (surrogate for orthogonality) for the 10X Genomics datasets.

Correspondingly, SMNN corrected data exhibits the lowest F value than that from the other three methods. Specifically, F value is with reduced by 81.5 - 96.6% on top of MNN, Seurat v3, and LIGER, respectively (Fig. 3.7f). Furthermore, we compared the distance for the cells between batch 1 and 2, and found that, compared to data before correction, both MNN and SMNN reduced the Euclidean distance between the two batches (Fig. 3.9). In addition, SMNN further decreased the distance by up to 8.2% than MNN (2.8%, 4.3% and 8.2% for cells of type CMP, MEP and granulocyte-monocyte progenitor (GMP) cells, respectively). ). Under scenarios where we only have partial cell type information, SMNN still better mixed cells of the same cell type across batches (detailed in Appendix B; Appendix B: Fig. 3.10a-c and e-g), and manifested the best/lowest F values, compared with uncorrected and MNN-corrected data (Appendix B: Fig. 3.10d and h). These results suggest improved batch effect correction by SMNN, compared to unsupervised correction methods.

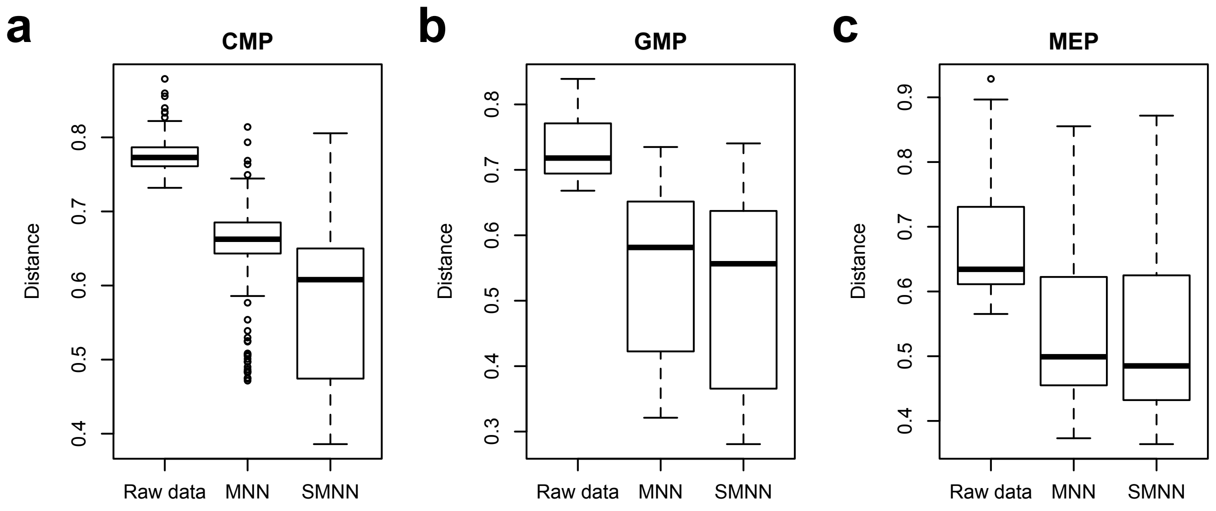


**Figure 3.9: Boxplot of distances by cell type.** For each of the three different cell types, CMP (a), GMP (b) and MEP (c), we calculate the distance for every cell in batch 2 to its corresponding centroid in batch1.

### 3.4.1 SMNN identifies differentially expressed genes that are biologically relevant

We then compared the DEGs among different cell types identified by SMNN and MNN. After correction, in the merged hematopoietic dataset, 1012 and 1145 up-regulated DEGs were identified in CMP cells by SMNN and MNN, respectively, when compared to GMP cells, while 926 and 1108 down-regulated DEGs were identified by the two methods, respectively (Fig. B.1a and Appendix Fig. B.4a). Of them, 736 up-regulated and 842 down-regulated DEGs were shared between SMNN and MNN corrected data. GO enrichment analysis showed that, the DEGs detected only by SMNN were overrepresented in GO terms related to blood coagulation and hemostasis, such as platelet activation and aggregation, hemostasis, coagulation and regulation of wound healing (Fig. B.1b).

Similar DEG detection was carried out to detect genes differentially expressed between CMP and MEP cells. 181 SMNN-specific DEGs were identified out of the 594 up-regulated DEGs in CMP cells when compared to MEP cells (Fig. B.1c), and they were found to be enriched for GO terms involved in immune cell proliferation and differentiation, including regulation of leukocyte proliferation, differentiation and migration, myeloid cell differentiation and mononuclear cell proliferation (Fig. B.1d). Lastly, genes identified by SMNN to be up-regulated in GMP when compared to MEP cells, were found to be involved in immune processes; whereas up-regulated genes in MEP over GMP were enriched in blood coagulation (Appendix Fig. B.4e-h). Comparatively, the GO terms enriched for MNN-specific DEGs seem not particularly relevant to corresponding cell functions (Fig. B.5). These cell-function-relevant SMNN-specific DEGs indicate SMNN can maintain some cell features that are missed by MNN after correction.

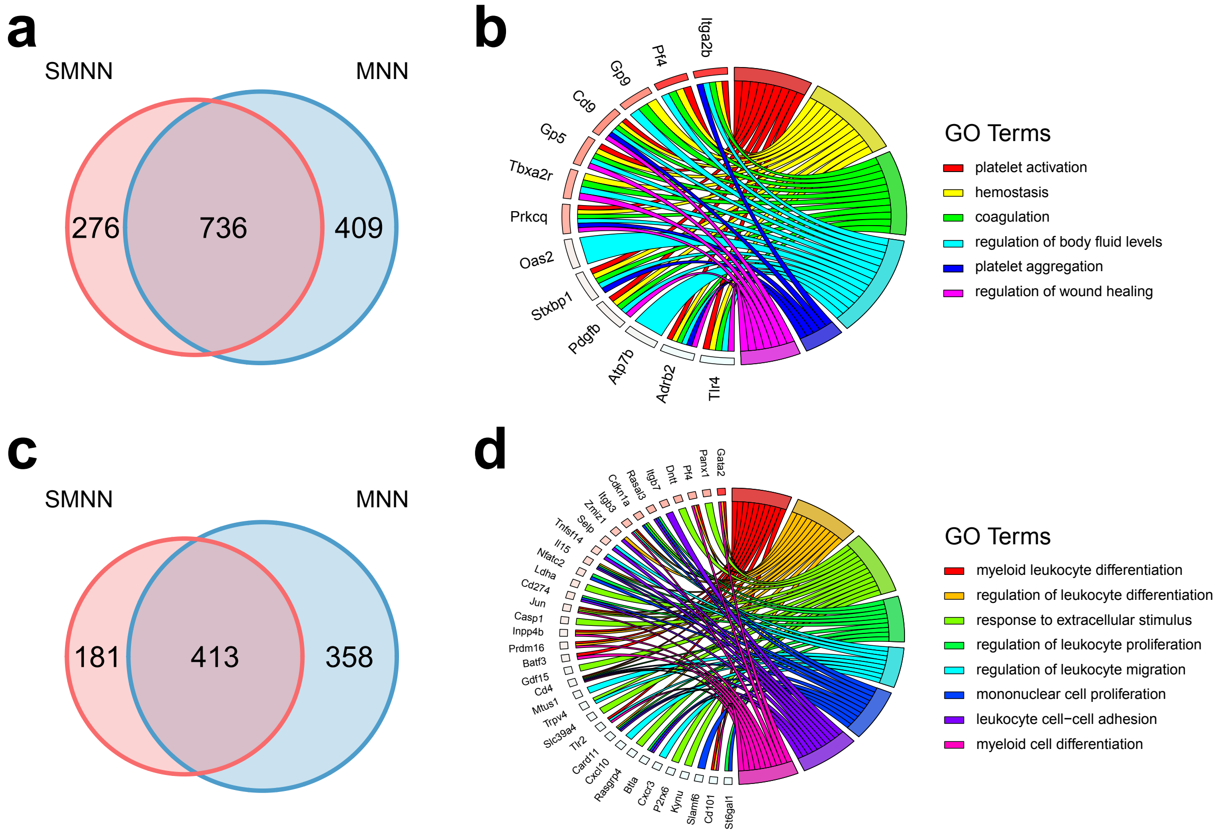


**Figure 3.10: Comparison of differentially expressed genes (DEGs), identified in the merged dataset by pooling batch 1 data with batch 2 data after SMNN and MNN correction.** (a) Overlap of DEGs up-regulated in CMP over GMP after SMNN and MNN correction. (b) Feature enriched GO terms and the corresponding DEGs up-regulated in CMP over GMP. (c) Overlap of DEGs up-regulated in CMP over MEP after SMNN and MNN correction. (d) Feature enriched GO terms and the corresponding DEGs up-regulated in CMP over MEP.

In addition, we considered two sets of "working truth": first, DEGs identified in uncorrected batch 1; second DEGs identified in batch 2, and we compared SMNN and MNN results to both sets of working truth. The results showed that, in both comparisons (one comparison for each set of working truth), fewer DEGs were observed in SMNN-corrected batch 2, but higher precision and lower false negative rate in each of the three cell types than those in MNN results. When compared to the uncorrected batch 1, 3.6% - 841% improvements in precision were observed in SMNN results than MNN (Fig. 3.11 and Appendix Fig. B.7). Similarly, SMNN increased the precision by 6.2% - 54.0% on top of MNN when compared to uncorrected batch 2 (Appendix Fig. B.6-B.8). We also performed DEG analysis at various adjusted p-value thresholds and the results showed that the better performance of SMNN is not sensitive to the p-value cutoff we used for DEG detection. Such an improvement in the accuracy of DEG identification indicates that higher amount of information regarding cell structure was retained after SMNN correction than MNN.

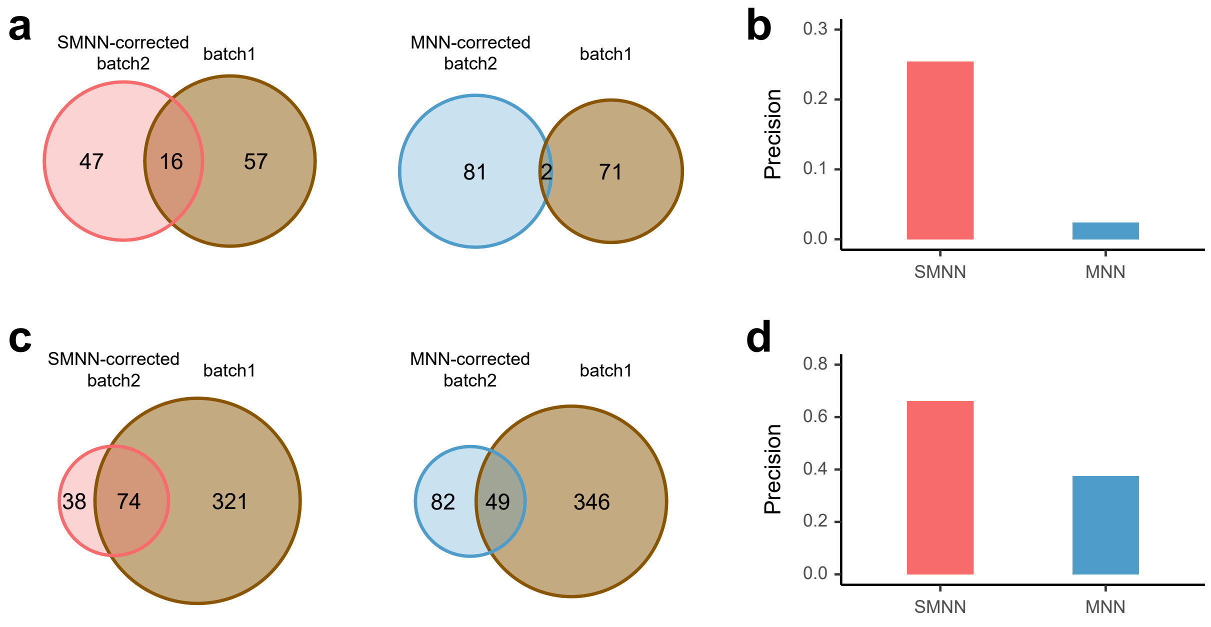


**Figure 3.11: Reproducibility of DEGs (between CMP and GMP), identified in uncorrected batch 1 and in SMNN or MNN-corrected batch 2.** (a) Reproducibility of DEGs up-regulated in CMP over GMP, detected in batch 1, versus SMNN (left) or MNN-corrected (right) batch 2. (b) Precision of the DEGs (between CMP and GMP) identified in batch 2 after SMNN and MNN correction. (c) Reproducibility of DEGs up-regulated in GMP over CMP, identified in the uncorrected batch 1, and in SMNN (left) or MNN-corrected (right) batch 2. (d) Precision of the DEGs up-regulated in GMP over CMP identified in batch 2 after SMNN and MNN correction.

We also identified DEGs between T cells and B cells in the merged PBMC and T cell datasets after SMNN and MNN correction, respectively. Compared to B cells, 3213 and 4180 up-regulated

DEGs were identified in T cells by SMNN and MNN, respectively, 2203 of which were shared between the two methods (Appendix Fig. B.10e). GO enrichment analysis showed that, the SMNN-specific DEGs were significantly enriched for GO terms relevant to the processes of immune signal recognition and T cell activation, such as T cell receptor signaling pathway, innate immune response-activating signal transduction, cytoplasmic pattern recognition receptor signaling pathway and regulation of autophagy (Appendix Fig. B.10f). In B cells, 5422 and 3462 were found to be up-regulated after SMNN and MNN correction, where 2765 were SMNN-specific (Appendix Fig. B.10g). These genes were overrepresented in GO terms involved in protein synthesis and transport, including translational elongation and termination, ER to Golgi vesicle-mediated transport, vesicle organization and Golgi vesicle budding (Appendix Fig. B.10h). These results again suggest that SMNN more accurately retains or rescues cell features after correction.

### 3.4.2 SMNN more accurately identifies cell clusters

**Table 3.1:** Major characteristics of the three benchmarking datasets.

| Dataset | Batch ID | Tissue origin | Num. of cells | Technical platform | Ref |
|---|---|---|---|---|---|
| Hematopoiesis | batch 1 | Mouse hematopoiesis | 2,729 | MARS-seq | Paul et al. 2015 |
| | batch 2 | Mouse hematopoiesis | 1,920 | SMART-seq2 | Nestorowa et al. 2016 |
| Pancreas | batch 1 | Human cadaveric pancreata | 1,007 | CEL-seq | Grün et al. 2016 |
| | batch 2 | Human cadaveric pancreata | 1,595 | CEL-seq2 | Muraro et al. 2016 |
| Droplet | batch 1 | PBMC | 68,580 | 10X Genomics GemCode | Zheng et al. 2017 |
| | batch 2 | T Cells | 4,459 | 10X Genomics GemCode | Zheng et al. 2017 |

Finally, we examined the ability to differentiate cell types after SMNN and MNN correction in three datasets (Table 3.1). In all three real datasets, ARI after SMNN correction showed 7.6 - 42.3% improvements over that of MNN (Fig. 3.12), suggesting that SMNN correction more effectively recovers cell-type specific features.

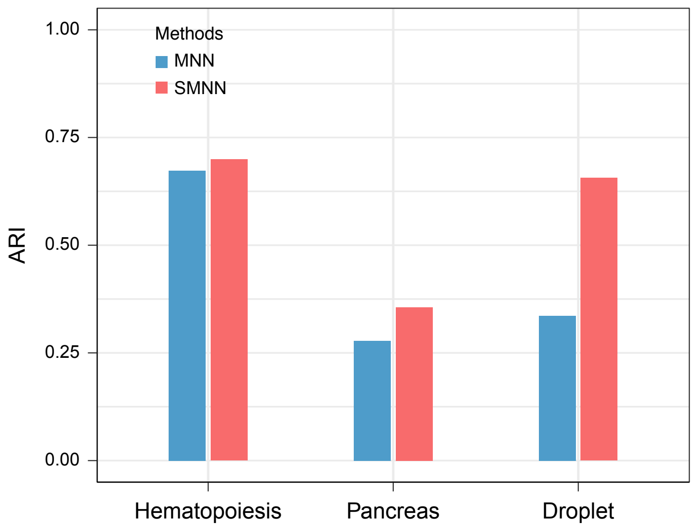


**Figure 3.12: Clustering accuracy in three datasets after batch effect correction.** Adjusted Rand Index (ARI) is employed to measure the similarity between clustering results before and after batch effect correction.

## 3.5 Discussion

In this study, we present SMNN, a batch effect correction method for scRNA-seq data via supervised mutual nearest neighbor detection. Our work is built on the recently developed method MNN, which has showed advantages in batch effect correction than existing alternative methods. On top of MNN, our SMNN relaxes a strong assumption that underlies MNN: that the biological differentiations are orthogonal to batch effects (Haghverdi et al., 2018). When this fundamental assumption is violated, especially under the realistic scenario that the two batches are rather different, MNN tends to err when searching nearest neighbors for cells belonging to the same biological cell type across batches. Our SMNN, in contrast, explicitly considers cell type label information to perform supervised mutual nearest neighbor matching, thus empowered to extract only desired neighbors from the same cell type.

A notable feature of our SMNN is that it can detect and match the corresponding cell populations across batches with the help of feature markers provided by users. SMNN performs clustering within each batch before merging across batches, which can reveal basic data structure, i.e. cell composition and proportions of contributing cell types, without any adverse impact due to batch effects. Cells of each cluster are labeled by leveraging their average expression levels of certain marker(s), thus enabling us to limit the mutual nearest neighbor detection within a smaller search space (i.e., only among cells of the same or similar cell type or status). This supervised approach eliminates the correction biases incurred by pairs of cells wrongly matched across cell types. We benchmarked SMNN together with three state-of-the-art batch effect correction methods, MNN, Seurat v3 and LIGER, on simulated and three published scRNA datasets. Our results clearly show the advantages of SMNN in terms removing batch effects. For example, our results for the hematopoietic datasets show that SMNN better mixed cells of all the three cell types across the two batches (Fig. 3.7a-e), and reduced the differentiation between the two batches by up to 96.6% on top of the corrected results from the three unsupervised methods (Fig. 3.7f), demonstrating that our SMNN method can more effectively mitigate batch effect. Additionally, cell population composition also can be a critical factor in batch effect correction (Fig. B.11). Our results by analyzing batches with varying cell type compositions suggest that our SMNN is robust to differential cell composition across batches.

More importantly, the wrongly matched cell pairs may wipe out the distinguishing features of cell types. This is mainly because, for a pair of cells from two different cell types, the true biological differentiations between them would be considered as technical biases and subsequently removed in the correction process. Compared to MNN, SMNN also appears to more accurately recover cell-type specific features: clustering accuracy using SMNN-corrected data increases substantially in all the three real datasets (by 7.6 to 42.3% when measured by ARI) (Fig. 3.12). Furthermore, we observe power enhancement in detecting DEGs between different cell types in the data after SMNN correction than MNN (Fig. 3.10 and 3.11 and Appendix Fig. B.4-B.8). Specifically, the precision of the DEGs identified by SMNN were improved by up to 841% and 54.0% than those by MNN when compared to the two set of working truth, respectively (Fig. 3.11c and d and Appendix Fig. B.7-B.8). Moreover, GO term enrichment results show that, the up-regulated DEGs identified only in SMNN-corrected GMP and MEP cells were involved in immune process and blood coagulation, respectively (Appendix Fig. B.4f and h), which accurately reflect the major features of these two cell types (Lieu and Reddy, 2012). Similarly, DEGs identified between T and B cells after SMNN correction are also biologically more relevant than those identified after MNN correction (Fig. B.10). These results suggest that SMNN can eliminate the overcorrection between different cell types and thus maintains more biological features in corrected data than MNN. Efficient removal of batch effects at reduced cost of biological information loss, manifested by SMNN in our extensive simulated and real data evaluations, empowers valid and more powerful downstream analysis.

In summary, extensive simulation and real data benchmarking suggest that our SMNN can not only better rescue biological features and thereof provide improved cluster results, but also facilitate the identification of biologically relevant DEGs. Therefore, we anticipate that our SMNN is valuable for integrated analysis of multiple scRNA-seq datasets, accelerating genetic studies involving single-cell dynamics.

CHAPTER 4

# CUE: CpG impUtation Ensemble for DNA Methylation Imputation Across Platforms

## 4.1 Introduction

DNA methylation of cytosine residues at CpG dinucleotides is one of the most extensively studied epigenetic marks. Rich recent literature provides evidence regarding its important role not only in normal development but also in risk and progression to many diseases (Bird, 2002; Gonzalo, 2010; Joubert et al., 2016; Klutstein et al., 2016; Iurlaro et al., 2017; Horvath and Raj, 2018; Turecki and Meaney, 2016; Bakusic et al., 2017). A wide range of biological processes are dependent on DNA methylation status, including gene transcription, X-chromosome inactivation, cell differentiation, cancer progression and other critical life events or processes such as aging (Bird, 2002; Gonzalo, 2010). Therefore, studying DNA methylation is of great interest and importance. However, such studies also provide great challenges for a number of reasons including but not limited to the following four. First, as an epigenetic marker, DNA methylation level is dynamic, varying over time, across different molecular environment, in different developmental stages, or across different tissues or cell lines. Second, correlation of methylation levels between CpG sites decreases dramatically with distance, for example typically $< 0.5$ when two CpG sites are merely >500bp apart. Third, there are multiple options to measure DNA methylation (see section 2 for a more detailed review). Lastly, even using the same measurement technology, batch effect and/or various other technical biases and noises are almost inevitable (Bird, 2002; Gonzalo, 2010).

With the emergence of powerful technologies such as DNA methylation microarray (Bibikova et al., 2011b) and bisulfite sequencing, geneticists are able to profile DNA methylation at increasingly higher resolutions. Taking methylation microarrays for an example, we have witnessed new platforms replacing old ones every few years (Moran et al., 2016; Bibikova et al., 2009; Dedeurwaerder et al., 2014). However, different platforms (for example, the widely used Illumina Human-

Methylation27, HumanMethylation450, MethylationEPIC BeadChips) target different CpG sites and have different marker densities. In addition, different biochemical or experimental techniques can be used to quantify methylation levels (e.g., type-I versus type-II assays adopted by the Illumina methylation arrays), further hindering joint analysis of data from multiple platforms. Two aforementioned microarrays, the Illumina HumanMethylationEPIC (HM850) BeadChip and HumanMethylation450 (HM450) BeadChip, are the most commonly used microarrays to measure DNA methylation levels. While HM450 investigates $485,577$ probes spanning 96% of CpG islands and 92% of CpG shores across a moderate number of genes (Bibikova et al., 2011b) , HM850 provides much more comprehensive coverage with the additional $413,743$ CpG sites located farther outside CpG islands. Because HM450 is no longer commercially available, we start to encounter data generated from a mixture of two different arrays. Such data have largely constrained joint analysis, where investigators typically focus on the probes shared between the two platforms. This is a prudent and convenient approach, without having to reevaluate all samples using HM850 (which is not only time consuming but also cost prohibitive). However, such an approach implies an unfortunate waste of HM850 data where more than 40% of data will not be used. In this study, we present CUE, CpG imputation Ensemble, an ensemble learning framework which leverages several machine learning algorithms and traditional statistical models, to efficiently utilize data generated from different platforms. While several existing methods designed for imputing sequencing-density methylation levels require hundreds of genomic features (e.g., those from the ENCODE project) to impute each missing CpG methylation site, we consider a relatively simple and more widely-applicable imputation regime where we only require methylation measurements from the HM450 BeadChip.

Because of the practical needs of imputation in DNA methylation data as well as the success of imputation methods in other genetic settings, a number of DNA methylation imputation methods has been proposed in the recent literature. Among them, support vector machines (SVM) and hybrid of SVM and other models predominated the DNA methylation imputation literature (Bhasin et al., 2005; Das et al., 2006; Bock et al., 2006; Fang et al., 2006; Zhou et al., 2012; Zhou and Tuck, 2007; Liu et al., 2015).Most of these methods assumed that methylation status is binary. In other words, a CpG site is either methylated or unmethylated for an individual and thus imputation becomes a classification problem. Almost all methods proposed prior to 2014 predicted the average methylation status for genomic regions, where each region encompasses multiple CpG sites (Zhang et al., 2015). All of the studies reported accuracy exceeding 90%.

Dichotomizing methylation status, adopted by these SVM based methods, is not ideal in that it can lead not only to general information loss but also to losing biologically meaningful information carried by intermediate raw beta values ($\beta$). Beta value is the ratio of intensities between methylated and unmethylated alleles, one standard quantitative measure of DNA methylation levels. $\beta$ values range from 0 to 1 with 0 being completely unmethylated and 1 completely methylated. With the prospering data science field, especially in areas such as machine learning and particularly deep learning (Bengio, 2009), several algorithms have been successfully employed and reported (see table) to outperform the earlier SVM based methods. For example, Zhang et al. (Zhang et al., 2015) in 2015 employed a random forest (RF) classifier to predict methylation levels with five groups of features selected from the ENCODE Project, achieving 96% accuracy. For another example, Angermueller et al. (Angermueller et al., 2017) in 2017 adopted a deep learning method to provide an accurate prediction of single-cell DNA methylation states, which achieved performances similar to the previous SVM or RF based methods. In the same year, BoostMe (Zou et al., 2018), based on the state-of-the-art boosting algorithm XGBoost, achieved the same level of accuracy as RF, but is computationally much more efficient.

In this study, we aim to impute an HM450 dataset up to an HM850 dataset, for increased coverage of this epigenomic landscape. We first seek solutions among the imputation methods in previous literatures. Our goal was to develop a general imputation framework to leverage the merits of different models and improve the imputation accuracy. We also examined the imputation results and filtering out the low-quality probes. These accurately imputed methylation values could subsequently improve power in downstream analysis, for example for associating methylation profiles with the phenotypic trait(s) of interest, widely referred to as epigenome wide association studies (EWAS).

## 4.2 Materials (data)

Recent advances in methylation microarrays and sequencing technologies have enabled us to gauge DNA methylation profiles genome-wide at single base-pair resolution (Laird, 2010). There are four major types of DNA methylation data commonly generated by different technologies (Laird, 2010).

a) Whole-genome bisulfite sequencing (WGBS) data This is the current gold standard for quantifying single-site DNA methylation levels across the genome. WGBS can theoretically quantify DNA methylation levels at 26 million (out of 28 million in total) CpG sites for the human genome (Laurent et al., 2010; Lister et al., 2011; Sandoval et al., 2011). The number of CpG sites measured by WGBS by far exceeds that of the remaining three methods to be reviewed. Despite its most comprehensive coverage, WGBS is not the standard practice mainly because of its prohibitively high costs and partly also because of various experimental biases such as those induced by bisulfite conversion. In addition, WGBS is difficult to apply to a particular region(s) of interest.

b) Methylation microarrays data The Illumina HumanMethylation27 and HumanMethylation450 BeadChip have been most widely used to measure DNA methylation levels at preselected CpG sites across the genome. They assay 28,000 and 480,00o CpG sites respectively. The latest Illumina MethylationEpic BeadChip encompasses over 850,000 sites, offering even broader coverage of the human methylome.

c) Methylated DNA immunoprecipitation sequencing data Generating this type of data is not only expensive but also experimentally difficult. For example, antibodies to pool down methylated DNA segments with high sensitivity and specificity are largely non-existing (Bryk et al., 2002; Ruike et al., 2010; Down et al., 2008).

d) Reduced representation bisulfite sequencing (RRBS) data Combining restriction enzyme digestion and bisulfite conversion, RRBS targets certain regions of the genome by enriching for areas with a high CpG content (Meissner et al., 2005).

For our ensemble imputation model, We analyzed data from two population cohorts measured both by both HumanMethylationEPIC (HM850) and HumanMethylation450 (HM450) BeadChip: the Extremely Low Gestational Age Newborns (ELGAN) study (Santos et al., 2019)($n = 127$) and the Posttraumatic Stress Disorder (PTSD) genetics data repository (Logue et al., 2017) ($n = 144$). We assess three datasets: ELGAN and PTSD separately, and a combined dataset with batch effects corrected via the ComBat R function. Samples from the PTSD genetics repository were from the Translational Research Center for TBI and Stress Disorders (TRACTS), a VA Rehabilitation Research and Development National Center for TBI Research at VA Boston Healthcare System. Informed consent was obtained from all PTSD subjects at the time of study inclusion. ELGAN study enrolled infants born $< 28$ weeks of gestion during 2002-2004, in five states and 14 hospitals in the United states (Santos et al., 2019). Detailed procedure regarding sample recruitment, main characteristics of study samples, and methylation measurements are presented in previous publications: Logue el al. for PTSD (Logue et al., 2017) and Hudson et al. for ELGAN (Santos et al., 2019).

The data used in this study have been pre-preprocessed previously. Logue el al. have previously published on the 145 samples measured both by HM450 and HM850 (Logue et al., 2017). The PTSD dataset was first corrected for the individual-level background noise by used GenomeStudio and then cleaned with the CpGassoc package and the ChAMP package in R (Team, 2008). The detailed data cleaning and processing of PTSD dataset can be referred to Mark's paper. In this dissertation, we further excluded 1 sample and keep 144 complete samples due to its missing of a lot of probes. Additionally, 127 subjects from ELGAN study (Santos et al., 2019) were selected based on the availability of placental samples with DNA methylation data by both HM450 and HM850. DNA methylation data for ELGAN dataset are first pre-processed by minfi package (Aryee et al., 2014). Then functional normalization is used for background subtraction and dye normalization. Hudson et al. finally used ComBat function from sva package to adjust for batch effects from two platforms (Johnson et al., 2007). The detailed placenta tissue collection and other assessments of DNA methylation for ELGAN dataset can be referred to Hudson's paper.

We aim to impute HM450 up to HM850, for increased coverage of this epigenomic landscape. To make our data better follow a Gaussian distribution (PFR model assumption) (Network, 2012, 2015), we employ M values, defined as

$$M = logit(\beta) = log_2(\frac{\beta}{1-\beta})$$

instead of the raw beta ($\beta$) values. Again, beta values are the ratios of intensities of the methylated probes over unmethylated probes.

Since imputing sporadic missing data is not the focus of our work, we removed all probes with any missing values for convenience. One could apply methods similar to those developed for gene expression data (Troyanskaya et al., 2001; Bø et al., 2004; Kim et al., 2006; Liew et al., 2010) to impute the sporadic missing values at directly assayed CpG sites. After removing sporadic missing values, we further filtered the probes to keep the common complete (no missing data) probes shared between two cohorts, which would make the assessments of different imputation models on two cohorts comparable. This left us 248,421 probes for HM450 and 587,454 probes for HM850 for ELGAN and PTSD cohort respectively. We used 248K as explanatory variables while the 339K HM850 specific probes as response variables.

### 4.3 Methodology

### 4.3.1 Penalized functional regression

Here we present a penalized functional regression model (Goldsmith et al., 2011) with minor modifications. We also observe $X_i(t)$, indexed by $T_i = t$ representing sample-specific density function of the DNA methylation levels measured by HM450. Previous work has been shown that by incorporating non-local density information we could improve the imputation accuracy. We consider the following functional linear regression model:

$$Y_i = \alpha + \int_0^1 X_i(\mathrm{t})\beta(t)dt + Z_i\gamma + \varepsilon_i$$

with $\beta(t) \in \mathcal{L}^2[0,1]$ characterizes the effect of density function $X_i(t)$ when $T_i = t$. $\alpha$ is the grand mean and $\gamma$ denotes the vector of regression coefficients corresponding to the vector of covariates $Z_i$, 25 downstream probes and 25 upstream probes to each target probe as the local covariates. $\epsilon_i \sim N\left(0, \sigma^2\right)$

**Estimation of $X_i(t)$:** In order to improve imputation accuracy, we incorporated functional predictors $X_i(t)$ into our model to capture methylome-wide information, besides methylation levels from local probes encapsulated in $Z_i$. According to the probe's relative location to a CpG island, we first defined five groups: "CpG Island," "North Shore," "South Shore," "North Shelf," and "South Shelf" (Bibikova et al., 2011a) (Fig. 4.1).

Then, we estimated the DNA methylation function $X_i(t)$ for a particular target probe using the DNA methylation data from all HM450 probes falling in the same group as the target probe. Assuming that probes with similar properties tend to have similar methylation profiles, we use $X_i(t)$ to borrow information from the nonlocal probes falling in the same group. The observed DNA methylation data in group g are denoted as $\tau_i^g = (\tau_1^g, \ldots, \tau_q^g)$, where q is the number of CpG sites falling into group g, and $\tau_j^g$ is the DNA methylation value at jth HM450 probe in group g with $j = 1, \ldots, q$. Rather than estimating $X_i(t)$ by expanding into the principal component basis obtained from its covariance matrix (Goldsmith et al., 2011), we employed kernel density estimation to obtain $X_i(t)$ with $\tau_i^g$ so that it is specific to group g.

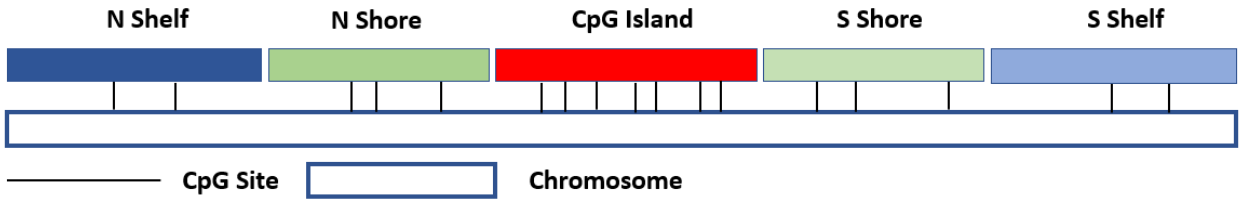


**Figure 4.1:** Five groups of CpG sites according to their relative location to CpG islands.

**Estimation of $\beta(t)$:** To perform model fitting, we projected the functional term $\beta(t)$ onto a linear spline basis:

$$\beta(t) = b_1 + b_2 t + \sum_{k=3}^{K_b} b_k \left(t - \delta_k\right)_+ \text{ with } \delta_k \in [0, 1]$$

where $\delta_k$ is the $k$ th knot along the interval $[0, 1]$ and $(t - \delta_k)_+$ is the positive part function:

$$(t - \delta_k)_+ = \begin{cases} t - \delta_k, & \text{if } t \geq \delta_k \\ 0, & \text{if } t < \delta_k \end{cases}$$

We further defined our spline basis vector:

$$\varphi(\mathrm{t}) = \{\varphi_1(\mathrm{t}), \varphi_2(\mathrm{t}), \ldots, \varphi_{K_b}(\mathrm{t})\} = \left\{1, \mathrm{t}, (\mathrm{t} - \delta_3)_+, (\mathrm{t} - \delta_4)_+, \ldots, (\mathrm{t} - \delta_{K_b})_\mathrm{t}\right\}$$

and a coefficient vector: $b = (b_1, b_2, .., b_{K_b})^T$ so that we may induce smoothing by assuming $b \sim N(0, \boldsymbol{D})$, where $\boldsymbol{D}$ is a penalty matrix corresponding to the particular spline basis $\varphi(\mathrm{t})$. Since

$\beta(t) = \varphi(t) \cdot b$, we had:

$$\int_0^1 X_i(\mathrm{t})\beta(t)dt = \int_0^1 f_{T_i}(\mathrm{t})\varphi(t) \cdot bdt = \int_0^1 f_{T_i}(\mathrm{t})\varphi(t)dt \cdot b$$

For ease of notation, we denoted $J_{X_\phi}$ as the $n \times K_b$ matrix with the $(i,k)^{th}$ entry equal to $\int_0^1 f_{T_i}(\mathrm{t})\varphi_k(t)dt$ and $\boldsymbol{Z}$ as the $n \times p$ matrix with the $i^{th}$ row equal to $Z_i$, where p is the number of covariates. The model can be written in matrix format as:

$$Y|X(t) = \left[1_n, J_{X_\varphi}, Z\right]\left[\alpha^T, b^T, \gamma^T\right] + \varepsilon$$
$$b \sim \mathrm{N}(0, D)$$

This is a mixed effect model with $K_b$ random effects b and penalty matrix:

$$D = \begin{bmatrix} 0_{2\times 2} & 0_{2\times(K_b-2)} \\ 0_{(K_b-2)\times 2} & \sigma_b^2 I_{(K_b-2)\times(K_b-2)} \end{bmatrix}$$

The advantage of the PFR approach is it borrows the info from non-local probes and one limitation is the running time of PFR is relative long compare to other single imputation approach.

### 4.3.2 CUE: CpG impUtation Ensemble

Denote the different imputation models by $f_k$, with $k = \{1,..,K\}$. Then the imputed values for the $i^{th}$ probe with the $k^{th}$ imputation model are denoted by $\hat{Y}_{ik} = f_k(Z_i)$ (or $f_k(Z_i, X_i(t))$ if we use our penalized functional regression approach). The proposed ensemble prediction estimator:

$$\hat{Y}_{i.} = \sum_{k=1}^{K} w_{i,k}\hat{Y}_{ik} \text{ , with weights } w_{i,k} \in [0,1]\ k = \{1,..,K\}$$

Note with $\beta$ methylation values, we have the natural constraint $Y_i, \hat{Y}_{ik} \in [0,1]$.

**Selection of Weights:** One challenge problem is how to select weights for the ensemble imputation methods. Here we list three different approaches to select the weights. First, equal weights: $w_{j,1} = w_{j,2} = \cdots = w_{J,K} = \frac{1}{K}$; Second, best-single-method weights (0-1 weights): $w_{j,\text{best}} = 1$, the other weights $= 0$; Third, theoretical optimal weights: given the predictions from different models, we need to solve the following probe specific optimization problem to learn the optimal weights for each probe.

$$\begin{cases} \min\limits_{w_i \in [0,1]^p} & L_i(w_i) = ||Y_i - \hat{Y}_{i.}||^2 = ||Y_i - \sum_{k=1}^{K} w_{i,k} \hat{Y}_{ik}||^2 \\ \text{subject to} & w_i = [w_{i,1}, w_{i,2}, .., w_{i,K}]^T \end{cases}$$

In general, the first one is simple and robust but could not guarantee the improvement of the performance. The second one and the third one would be guaranteed to be no worse than any single imputation tools by design. The third one is actually the best linear fitting on the training data, which tends to overfits on the training data. The above convex optimization can be efficiently solved by standard convex optimizer. However, in the real application examples, the true $Y_i$ could not be observed. One possible remedy is that we solve the optimal weights for the training dataset where we observe the true $Y_i$ and directly applied learning weights to the test dataset. If the training dataset and the test dataset are generated from same distribution, then the optimal weights we learned from training datasets can be generalized to the test dataset easily. If not, other domain adaption and transfer learning techniques can be incorporated to get more accurate estimation of the optimal weights for the test dataset. But in this chapter, we would assume that the training and test dataset are generated from the same dataset.

In this study, we adopted the second approach to seek for the balance between imputation performances and robustness. Based on the training results, we select the best method for imputation at each CpG site and employ that model for the final prediction. Here the model comparison criterion is out-of-bag predicted MSE. Suppose the $k-th$ model outperforms other methods on a CpG site (i.e., with lowest out-of-bag prediction MSE), then $w_k = 1$ and $w_i = 0$ for $i \neq k$. Namely, $\widehat{y_{Ens}} = \widehat{y_k}$ for this CpG site if the $k-th$ model performs the best. Consequently, the performance of the ensemble method outperforms other single methods by design.

### 4.3.3 Imputation quality assessment and control

Six-fold cross-validation was used to assess imputation quality. For each split, the full dataset was randomly divided into a training set, consisting of 5/6 of the total samples, and a testing set (1/6 of the total samples). For each testing set, we only kept data at the probes common to

HM450 and HM850 (shared probes, or predictor probes), and masked methylation values of HM850-specific probes. For the training set, we employed methylation measurements on the shared probes as predictors to impute methylation values at HM850 specific probes. Since most HM450 probes are measured by both HM850 and HM450 platforms, the predictors used in our model can be methylation levels for these shared probes measured from either array. Note that our prediction model was built under the realistic and thus more challenging scenario where we used as predictors the measurements from HM450 array instead of those from HM850 array, which would require the training dataset had measurements from both arrays. Specifically, we first fitted our PFR model, learning the relationship between the methylation values of the shared and HM450-specific probes. Second, we used the fitted model to impute the masked values of HM850 probes from the HM450 data in the testing set. In the end, we evaluated the imputation performance by integrating imputation results from all 6 splits.

As measures of the imputation quality, we employ both the predicted root mean squared error (RMSE) and the accuracy with 0.5 as the threshold. Conventionally, if the raw methylation value is above the threshold, we call it methylated, and unmethylated otherwise. We employ two quality control criteria: the probe-level predicted RMSE $> 0.05$ and the probe-level predicted accuracy $> 95\%$ when dichotomizing DNA methylation level at a cutoff of 0.5.

## 4.4 Real data results

In this study, we used DNA methylation data of both HM450 and HM850 from two cohorts, namely the Posttraumatic Stress Disorder (PTSD) genetics repository (Logue et al., 2017) (144 whole blood samples) and Extremely Low Gestational Age Newborns (ELGAN) study (Santos et al., 2019) (127 placenta samples). We first comprehensively assessed five methods: three traditional statistical methods, k-nearest-neighbors (KNN), logistic regression (Logistic) and penalized functional regression (PFR) model (Goldsmith et al., 2011; Zhang et al., 2016); and two modern machine learning algorithms, random forest (RF) and XGBoost. Their performances were systematically evaluated using six-fold cross-validation on the two cohorts separately.

### 4.4.1 Cross validation results on ELGAN and PTSD

In this dissertation, we mainly focused on imputation within the same tissue, because for most studies, samples are usually collected for the same tissue and more importantly because methylation profiles tend to differ profoundly, preventing accurate or even meaningful imputation across tissues. We assessed imputation quality within each cohort by conducting the six-fold cross validation, separately on whole blood samples from PTSD and placenta samples from ELGAN. Note that all samples in the two cohorts (ELGAN and PTSD) in this study have both 850K and 450K data and thus could be used to evaluate our CUE method using the aforementioned cross-validation strategy.

Note the time complexity of our method is O(n) and we can easily impute each target probe in parallel to decrease clock computation time, where n is the number of the HM850-specific probes (in this case $n = 339,033$ HM850-specific probes).

For the ELGAN dataset, RF achieved the fastest speed, the smallest root mean square error (RMSE) (0.099) and the highest accuracy (measured by dichotomizing DNA methylation level at a cutoff of 0.5) (94.60%) among the five imputation tools we compared (Table 1). KNN, PFR and XGBoost performed slightly worse than RF with regard to RMSE (decreases by 0.004-0.025), and had 0.34%-1.96% loss in terms of classification accuracy for dichotomous methylation status. Logistic regression can achieve an accuracy higher than 90% but performed the worst in RMSE.

**Table 4.1:** Imputation performances in the ELGAN dataset and the PTSD dataset. For all the computational results reported in tables of this chapter, we used 15 CPUs. Logistic regression did not converge (D.N.C.) for ELGAN dataset.

| | PTSD performance | | | ELGAN performance | | |
|---|---|---|---|---|---|---|
| | Accuracy | RMSE | Time | Accuracy | RMSE | Time |
| KNN | 98.02% | 0.054 | **3hrs** | 92.64% | 0.124 | 2.5hrs |
| Logistic | D.N.C. | D.N.C. | D.N.C. | 91.76% | 0.263 | **2.5hrs** |
| PFR | 98.41% | 0.044 | 18.5hrs | 93.50% | 0.114 | 6hrs |
| RF | 98.02% | 0.054 | 5hrs | **94.60%** | **0.099** | 3hrs |
| XGBoost | **98.59%** | **0.040** | 3hrs | 94.26% | 0.103 | 3hrs |

Similarly, we obtained the six-fold cross validation results in the PTSD dataset. Among the five single imputation results, the winner in this PTSD dataset is XGBoost, in contrast to random forest for the ELGAN dataset. Specifically, XGBoost achieved the smallest RMSE (0.04) and the highest accuracy (98.59%) (Table 4.1). In fact, random forest was the not even the second best. Penalized functional regression approach achieved a 0.39% higher accuracy and a lower (by 0.01)

RMSE than random forest. Compared with previous results from the ELGAN dataset, this set of results suggests that there was no uniformly best single imputation method across different tissues or datasets, which inspired our ensemble imputation framework.

We further reported the proportions of CpG sites where each imputation method outperformed all others in PTSD (Fig. 4.2). Results showed that all four viable imputation methods (we excluded logistic regression since it failed to converge) are champion at some CpG sites, which again motivates the development of an ensemble imputation framework. For more than 42% of the CpG sites, random forest achieved the lowest RMSE (Fig. 4.2). In total, random forest and XGBoost outperformed the other methods for  70% of CpG sites. In contrast, penalized functional regression was the best imputation model for 26% of the CpG sites while XGBoost for  30% CpG sites (Fig. 4.2). Last but not the least, for 1.7% of the probes (5,596 probes) KNN performed the best (Fig. 4.2).

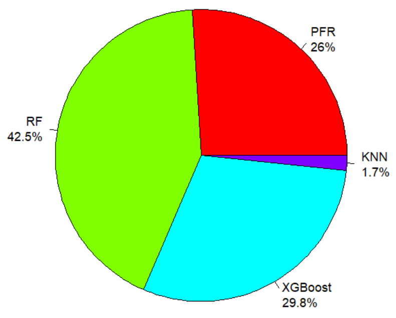


**Figure 4.2:** Proportions of Best Method for Each Targeted Site in PTSD.

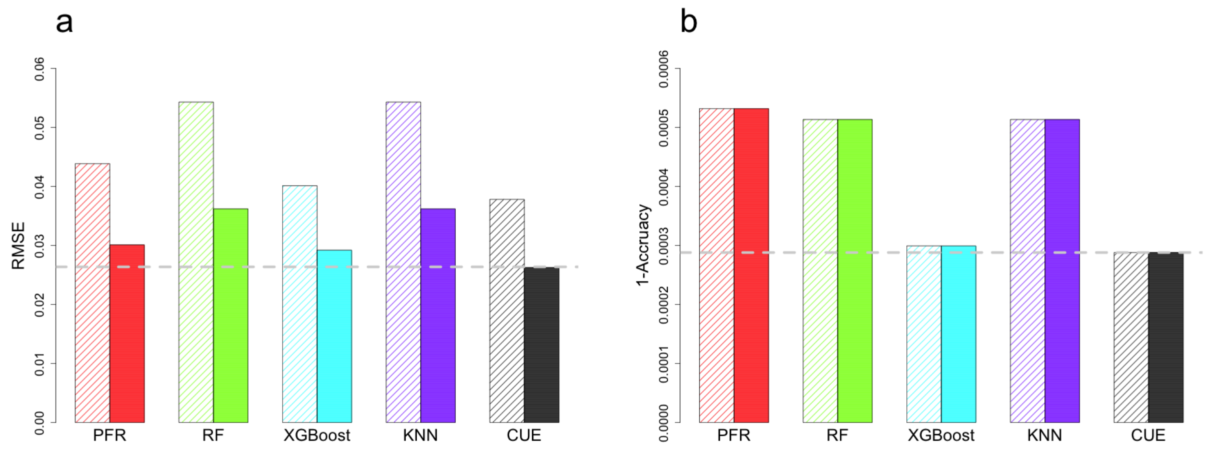


**Figure 4.3:** Imputed Performances Before (Left Bar) and After QC (Right Bar) for QCed Probes in PTSD. (a) RMSE comparisons. (b) 1- Accuracy (classification error) comparisons. Different colors represent different methods. The horizontal dash line is the lowest value corresponding to the best method.

We developed CUE, CpG imputation Ensemble, which employed an ensemble approach to improve prediction of methylation values at HM850 specific CpG sites. We also filtered out the probes failed to pass the quality control (QC) criteria (RMSE $< 0.05$ and accuracy $> 95\%$ at CpG site level). Before QC, all the methods' predicted RMSE are below 0.06 and CUE performed best (Fig. 4.3). The predicted RMSE of individual tools can be reduced by 5.8%-30.3% with CUE. After post-imputation QC, the predicted RMSE of all the tools were reduced by $37.9\% - 50.0\%$. Among all the 339,033 HM850-only CpG sites shared between ELGAN and PTSD, CUE out-performed all individual methods at 289,604 (85.4%) sites (Table C.1). Specifically, CUE achieved the lowest predicted RMSE (0.026) and the highest accuracy (99.97%), compared with individual methods with RMSE ranging 0.029-0.036 (improved by $10.0\% - 27.4\%$) and with accuracy $99.95\% - 99.97\%$
.

When evaluating prediction accuracy, we use 0.5 as the conventional cut-off for the methylation states (labeled as 1 [or methylated] if the beta methylation level is above 0.5; 0 [or unmethylated] otherwise). Our CUE method is robust to different cut-offs (thresholds) and achieves the highest accuracy across all thresholds (Fig. C.2). Logistic regression seems sensitive to the threshold probably because it is trained based on labels defined at the cutoff of 0.5.

### 4.4.2 Independent validation results: cross-dataset performances

Although six-fold cross-validation experiments above provide useful information, in reality, imputation will be performed in dataset(s) distinct from the one based on which training models are built. To provide more honest performance estimates and to assess the transportability of models trained by our CUE and individual methods, we examined their performances across the two datasets. With the presence of many systematic differences between the two datasets, we first attempted to correct for batch effects. Specifically, we employed Combat (Johnson, Li et al. 2007) to generate a harmonized dataset after pooled together data from the two cohorts. We then trained methylation prediction models with the harmonized ELGAN dataset and tested on the harmonized PTSD dataset. Among the five single imputation methods, random forest achieved the highest accuracy (95.23%) and the lowest predicted RMSE (0.07). KNN is the fastest model with 1.4% loss in accuracy and 0.02 loss.

### 4.4.3 DNA methylation varies across different tissues

DNA methylation data varies across different tissues inherently. Separately for each cohort (tissue), we classified DNA methylation probes into three categories: unmethylated probe if $\beta$ values across all samples in the cohort are less than 0.5; methylated probe if $\beta$ values across all samples in the cohort are greater than 0.5; bi-modal probe otherwise. We reported the proportions of three different categories probes for 339,033 HM850-targeted probes (Fig. 4.4). For PTSD cohort, almost 91% probes are either methylated or unmethylated, while only 54% for ELGAN. In general, bi-modal probes are more difficult to impute because of their complexity inherently. One indicator is that their variances tend to be larger than the rest probes.

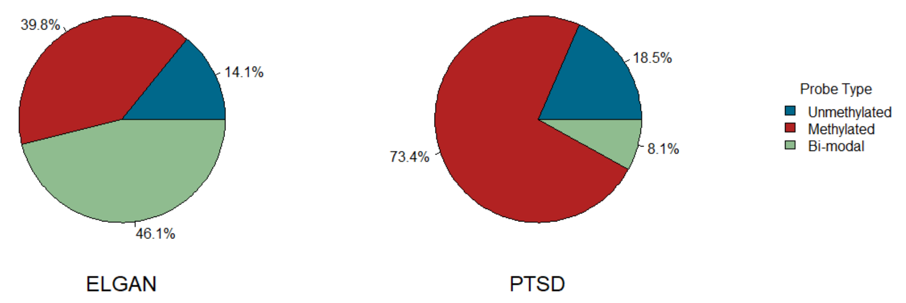


**Figure 4.4:** Proportions of Three-Category HM850-specific Probes in Two Studies.

## 4.5 Discussion

In this study, we developed an ensemble method, CUE, to enhance prediction accuracy when imputing methylation values across different platforms. Although our initial goal is to extende our previously developed penalized functional regression (PFR) framework, and systematically evaluated its performance with multiple alternative methods. Results from real data analyses suggest that there is no uniformly single best imputation method across different datasets or tissues, which motivates our ensemble method, CUE. Under three different scenarios with data from two different cohorts (one with samples from placenta and the other with samples from whole blood), CUE outperforms all the single imputation tools in terms of both predicted root mean square error (measuring methylation values with a continuous scale) and predicted accuracy (dichotomizing methylation values). CUE also leads to a larger number of probes well imputed (that is, passing post-imputation quality control) than any single imputation method.

CUE can produce accurate imputation results when the training and testing data characterize the same tissue under similar conditions. With our CUE imputation framework, we can combine data from multiple platforms, enabling higher resolution and more powerful downstream analysis. For example, the combined dataset can be used to boost the power not only for epigenome wide association (EWAS) study, but also for mQTL analysis, as well as multi-omics integrative analysis. Regardless of the epigenetic architecture underlying phenotype(s) of interest, we expect our method to facilitate more efficient utilization of methylation data from multiple platforms and to foster advances in understanding the impact of DNA methylation on phenotype(s) of interest.

To further assess the generalizability of prediction model trained on data from the same tissue but from different institutions, we would benefit from the availability of such data. However, we are not aware of such data despite our keen efforts to assemble such datasets. Future studies are highly warranted when data become available.

In summary, findings in this study suggest that our CUE ensemble methylation imputation method is valuable for imputing from HM450 to HM850. DNA methylation data inherently vary across tissues and ours and others' results (Zhang et al., 2016) suggest that it would be prudent to train separate imputation prediction models for different tissues. From this study, we provide two sets of imputation models: one for whole blood and the other for placenta. Our study is the first to impute from HM450 to HM850 for two different tissues. We believe our CUE method as well as the pre-trained imputation models across the two tissues will be of value to many investigators, facilitating more powerful epigenetics studies with either HM450 data or a mixture of HM450 and HM850 data.

# Summary

Machine learning methods are of increasing importance in both foundations of statistical inference and modern genetics applications. In the dissertation, we first focus on the foundation of the inference problem in Chapter 2. Specifically, we introduce a new ML-based inference framework, deep fiducial inference, accompanied by a computational algorithm, approximate fiducial computation. Then, we consider two ML applications, one in genetics in Chapter 3 and one in epigenetics in Chapter 4.

In Chapter 2, we propose a fiducial autoencoder for the circumstance in which the analytical form of the inverse function is not available or the marginal fiducial density is intractable. The universal approximation theorem provides theoretical guarantees for the approximation performance of our FAE, and our simulations further validate our approach. The proposed FAE can accurately approximate the inverse function, and it can be efficiently combined with the AFC algorithm to provide valid and accurate inferences of the true parameters. The competitive performance of AFC corrected FAE solutions both in terms of efficiency and accuracy suggests that this is a promising area for future research.

In Chapter 3, we present SMNN, a batch effect correction method for scRNA-seq data via supervised mutual nearest neighbor detection. Our SMNN explicitly considers cell type label information to perform supervised mutual nearest neighbor matching, thus empowered to extract only desired neighbors from the same cell type. Extensive simulation and real data benchmarking suggest that our SMNN can better rescue biological features and thereof provide improved cluster results. Therefore, we anticipate that our SMNN is valuable for the integrated analysis of multiple scRNA-seq datasets.

In Chapter 4, we develop an ensemble method, CUE, to enhance prediction accuracy when imputing methylation values across different platforms. CUE can produce accurate imputation results when the training and testing data characterize the same tissue under similar conditions. With our CUE imputation framework, we can combine data from multiple platforms, enabling higher resolution and more powerful downstream analysis. Findings in this study suggest that our CUE ensemble methylation imputation method is valuable for imputing from HM450 to HM850.

# APPENDIX A
# PYTHON CODES FOR FAE

**Listing A.1:** Python code for BOD example

```
import keras
import tensorflow as tf
from keras import backend as K
import numpy as np

from keras.layers import Input, Dense
from keras.models import Model, Sequential
from keras.models import model_from_json
from keras.optimizers import RMSprop, Adam

import matplotlib.pyplot as plt
import sys
import os

class FAE():
        def __init__(self):
                # Input shape
                self.samples = 120000
                self.noise_var = 0.015
                self.dim = 5
                self.channels = 1
                self.optimizer = Adam()

                # Initialzie input
                x_input = Input(shape=(self.dim, self.channels), name='x')
                y_input = Input(shape=(self.dim, self.channels), name='y')
                z_input = Input(shape=(self.dim, self.channels), name='z')

                # Build encoder
                self.encoder = self.build_encoder()
                self.encoder0 = self.build_encoder0()
                self.encoder1 = self.build_encoder1()
                code = self.encoder([x_input, y_input, z_input])
                t0 = self.encoder0([code])
                t1 = self.encoder1([code])

                # Build decoder
                self.decoder = self.build_decoder()
                y_hat = self.decoder([t0, t1, x_input, z_input])

                # The combined model (conect encoder and decoder)
                self.autoencoder = Model(inputs=[x_input,y_input,z_input],
                    outputs=[y_hat,t0,t1])
                self.autoencoder.compile(optimizer = self.optimizer,
                loss={'decoder': 'mean_squared_error',
                'encoder0': 'mean_squared_error',
                'encoder1': 'mean_squared_error'},
                loss_weights={'decoder': 5,
                'encoder0': 10,
                'encoder1': 3})
                self.autoencoder.summary()

        def build_encoder(self):
                # this is our input placeholder
                x_input = Input(shape = (self.dim, self.channels),name='x')
                y_input = Input(shape = (self.dim, self.channels),name='y')
                z_input = Input(shape=(self.dim,  self.channels), name='z')



                # "encoded" is the encoded representation of the input
                encoded=keras.layers.concatenate([x_input,y_input,z_input])
                encoded = Dense(32, activation='relu')(encoded)
                encoded = Dense(64, activation='relu')(encoded)
                encoded = Dense(128, activation='relu')(encoded)
                encoded = Dense(512, activation='relu')(encoded)
                encoded = Dense(128, activation='relu')(encoded)
                encoded = Dense(512, activation='relu')(encoded)
                encoded = Dense(128, activation='relu')(encoded)
                encoded = Dense(32, activation='relu')(encoded)
                encoder = Model(inputs=[x_input,y_input,z_input], outputs=
                    encoded, name="encoder")
                encoder.summary()
                return encoder

        def build_encoder0(self):
                encoded0 = Input(shape=(self.dim,  32), name='code0')
                encoded = Dense(128, activation='relu')(encoded0)
                encoded = Dense(64, activation='relu')(encoded)
                encoded = Dense(32, activation='relu')(encoded)
                t0_candidate = Dense(1, activation='relu',name='t0_hat')(
                    encoded)
                et0 = keras.layers.AveragePooling1D(pool_size=self.dim,
                    strides=None, padding='valid',name='et0')(t0_candidate)
                encoder0 = Model(inputs=encoded0, outputs=et0, name="encoder0"
                    )
                encoder0.summary()
                return encoder0

        def build_encoder1(self):
                # define encoder1
                encoded = Input(shape=(self.dim,  32), name='code1')
                t1_candidate = Dense(1, activation='relu',name='t1_hat')(
                    encoded)
                et1 = keras.layers.AveragePooling1D(pool_size=self.dim,
                    strides=None, padding='valid',name='et1')(t1_candidate)
                encoder1 = Model(inputs=encoded, outputs=et1, name="encoder1")
                encoder1.summary()
                return encoder1



        def build_decoder(self):
                # this is our input placeholder
                dt0 = Input(shape = (1,  self.channels),name='dt0')
                dt1 = Input(shape = (1,  self.channels),name='dt1')
                x_input = Input(shape = (self.dim,  self.channels),name='x')
                z_input = Input(shape=(self.dim, self.channels), name='z')
                def dg(ip):
                    tt0 = ip[0]
                    tt1 = ip[1]
                    tx2 = ip[2]
                    tz2 = ip[3]
                    ty = tt0 * (1-K.exp(-tt1*tx2)) + tz2
                    return ty
                y_hat = keras.layers.Lambda(dg)([dt0, dt1, x_input, z_input])
                encoder = Model(inputs=[dt0,dt1, x_input, z_input], outputs=
                    y_hat, name="decoder")
                encoder.summary()
                return encoder

        def load_data_BOD(self,sort = True):
                # Generate BOD Data
                m = self.dim
                n = self.samples
                x=np.array([2.0, 4.0, 6.0, 8.0, 10.0])

                def DataGenerateFunction(z,t0,t1,m,n):
                    y = z + np.diag(t0) @ (1 - np.exp(- (t1.reshape(n,1) @
                        x.reshape((1,m))) ) )
                    return y

                def model1(m = 5, n = 30):
                    t0 = np.random.uniform(0.4, 1.2, n)
                    t1 = np.random.uniform(0.000001, 0.2, n)
                    z = np.random.normal(0, self.noise_var, (n,m))
                    y = DataGenerateFunction(z,t0,t1,m,n)
                    return (m, n, y, z, t0, t1)

                # setup the random seed
                np.random.seed(20200521)
                (m, n, y, z, t0, t1) = model1(m = m, n = n)
                if sort == True:
                    p = y.argsort(axis=1)
                    y.sort(axis=1)
                    z=np.array([z[i,p[i,]] for i in range(n)])
                y=y[:,:,np.newaxis]
                z=z[:,:,np.newaxis]
                t0=t0[:,np.newaxis,np.newaxis]
                t1=t1[:,np.newaxis,np.newaxis]
                r=0.8 # ratio of train and validation
                train_y=y[0:int(n*r),:,:]
                train_z=z[0:int(n*r),:,:]
                valid_y=y[int(n*r):n,:,:]
                valid_z=z[int(n*r):n,:,:]
                train_t0=t0[0:int(n*r),:,:]
                train_t1=t1[0:int(n*r),:,:]
                valid_t0=t0[int(n*r):n,:,:]
                valid_t1=t1[int(n*r):n,:,:]
                x = np.tile(x,(n,1))
                x = x[:,:,np.newaxis]
                train_x = x[0:int(n*r),:,:]
                valid_x = x[int(n*r):n,:,:]
```

```
        print(train_y.shape[0], 'train samples')
        print(valid_y.shape[0], 'test samples')
        return (train_x,train_y,train_z,train_t0,train_t1,valid_x,
            valid_y,valid_z,valid_t0,valid_t1,r)

    def train_AE(self, epochs, batch_size=256, sort = True):
        (train_x,train_y,train_z,train_t0,train_t1,valid_x,valid_y,
            valid_z,valid_t0,valid_t1,r) =  self.load_data_BOD(sort =
            True)
        self.train = self.autoencoder.fit({'x': train_x, 'y': train_y,
             'z': train_z},
                {'decoder': train_y,
                'encoder0': train_t0,
                'encoder1': train_t1},
                batch_size=batch_size,epochs=epochs,verbose=1,
                validation_data=({'x': valid_x, 'y': valid_y, 'z':
                    valid_z},
                                                {'decoder': valid_y,
                                                'encoder0': valid_t0,
                                                'encoder1': valid_t1}
                                                ))
        pred_train = self.autoencoder.predict_on_batch({'x': train_x,
            'y': train_y, 'z': train_z})
        pred_valid = self.autoencoder.predict_on_batch({'x': valid_x,
            'y': valid_y, 'z': valid_z})
        # np.savez('pred_train_valid.npz', pred_train=pred_train,
            pred_valid=pred_valid,
#           train_y=train_y,train_t0=train_t0,train_t1=train_t1)
        # evaluate the model
        scores = self.autoencoder.evaluate(x=[valid_x, valid_y,
            valid_z],y=[valid_y,valid_t0,valid_t1], verbose=0)
        print("%s: %.2f" % (self.autoencoder.metrics_names[0], scores
            [0]))
        print("%s: %.2f" % (self.autoencoder.metrics_names[1], scores
            [1]))
        print("%s: %.2f" % (self.autoencoder.metrics_names[2], scores
            [2]))
        print("%s: %.2f" % (self.autoencoder.metrics_names[3], scores
            [3]))

def test_AE(self, t0=0.8, t1=1.0, n_test=1000, sort = True):
        m = self.dim
        x=np.array([2.0, 4.0, 6.0, 8.0, 10.0])
        def model_test():
                z = np.random.normal(0, self.noise_var, (1,m))
                y = z + t0 * (1 - np.exp(- (t1  * x ) ) )
                return (m, n_test, y, z, t0, t1)

        np.random.seed(20200526) # 0504
        (m, n_test, y_test, z_test, t0, t1) = model_test()
        y_test = np.array([0.1522071, 0.29667172, 0.41254479,
            0.48237946, 0.56707723])
        p_test = np.array([0,1,2,3,4])
        print(y_test)
        print(p_test)
        y_test = np.tile(y_test,(n_test,1))
        y_test = y_test[:,:,np.newaxis]
        np.random.seed(20200504)
        z_test = np.random.normal(0, self.noise_var, (n_test,m))
        if sort == True:
                z_test=np.array([z_test[i,p_test] for i in range(
                    n_test)])
        z_test = z_test[:,:,np.newaxis]
        x_test = np.tile(x,(n_test,1))
        x_test = x_test[:,:,np.newaxis]
```

```
                pred = self.autoencoder.predict_on_batch({'x': x_test, 'y':
                    y_test, 'z': z_test})
                t0_fae = pred[1]
                t1_fae = pred[2]
                np.save("BOD_t0_%.3f.npy"%t0, t0_fae)
                np.save("BOD_t1_%.3f.npy"%t1, t1_fae)
                np.save("BOD_y_hat.npy", pred[0])


if __name__ == '__main__':
        fae = FAE()
        fae.train_AE(epochs=10, batch_size=500, sort = False)
        fae.test_AE(t0=0.9, t1=0.1, n_test=10000, sort = False)
```

**Listing A.2:** Python code for non-linear data generating example

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 30 09:19:36 2018

@author: GangLi
"""


import keras
import numpy as np
from keras import backend as K
from keras.layers import Input, Dense
from keras.models import Model
from keras.optimizers import RMSprop, Adam
import matplotlib as mpl
#mpl.use('macOsX')

#################################################################
```

```
# Generate Data
##############################################################

def DataGenerateFunction(z,mu,m):
    x = np.transpose(np.tile(mu,(m,1))) + np.matmul(np.diag(np.power(mu,(3/2))
        ),z)
    return x

def model1(m = 10, n = 30):
    mu = np.random.uniform(0, 6, n)
    z = np.random.normal(0, 1, (n,m))
    x = DataGenerateFunction(z,mu,m)
    return (m, n, x, z, mu)

(m, n, x, z, mu0) = model1(m = 3, n = 100000)
x=x[:,:,np.newaxis]
z=z[:,:,np.newaxis]
r=0.8 # ratio of train and validation
train_X=x[0:int(n*r),:,:]
train_z=z[0:int(n*r),:,:]
valid_X=x[int(n*r):n,:,:]
valid_z=z[int(n*r):n,:,:]
print(train_X.shape[0], 'train␣samples')
print(valid_X.shape[0], 'test␣samples')

##########################################################
# Define the FAE
##########################################################
inChannel = 1
# this is our input placeholder
x_input = Input(shape = (m, inChannel),name='x')
```

```
z_input = Input(shape=(m,  inChannel), name='z')



# "encoded" is the encoded representation of the input
encoded=keras.layers.concatenate([x_input,z_input])
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(128, activation='relu')(encoded)
encoded = Dense(512, activation='relu')(encoded)
encoded = Dense(128, activation='relu')(encoded)
encoded = Dense(512, activation='relu')(encoded)
encoded = Dense(128, activation='relu')(encoded)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(128, activation='relu')(encoded)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
mu = Dense(1, activation='relu',name='mu_hat')(encoded)
mu1 = keras.layers.AveragePooling1D(pool_size=3, strides=None, padding='valid'
    )(mu)

def dg(ip):
    mu1 = ip[0]
    z2 = ip[1]
    mu2=K.repeat_elements(mu1,3,1)
    return mu2 + K.pow(mu2,3/2)*z2

decoded = keras.layers.Lambda(dg)([mu1,z_input])
FAE = Model(inputs=[x_input,z_input], outputs=[decoded,mu1])
FAE.compile( optimizer = Adam(),
              loss={'lambda_1': 'mean_squared_error',
                    'average_pooling1d_1': 'mean_squared_error'},
                loss_weights={'lambda_1': 1,
                              'average_pooling1d_1': 1})
```

```
FAE.summary()

###### Model Fitting
batch_size = 250
epochs = 10
FAE_train=FAE.fit({'x': train_X, 'z': train_z},
                  {'lambda_1': train_X,
                   'average_pooling1d_1': mu0[0:int(n*r),np.newaxis,np.newaxis
                       ] },
                  batch_size=batch_size,epochs=epochs,verbose=1,
                  validation_data=({'x': valid_X, 'z': valid_z},
                                   {'lambda_1': valid_X,
                                    'average_pooling1d_1':mu0[int(n*r):n,np.
                                        newaxis,np.newaxis] }))
[FAE_pred_X,FAE_pred_mu]=FAE.predict({'x': valid_X, 'z': valid_z})
```

# APPENDIX B
# ADDITIONAL PERFORMANCE EVALUATIONS FOR SMNN

In this study, we further evaluated several other computing and performance aspects of SMNN. First, we evaluated SMNN's robustness to the availability of only partial cell type information. Specifically, we performed SMNN correction for two hematopoietic datasets by feeding cluster labels for either one or two of the three cell types. The results showed that, under such partial label information, SMNN still better mixed cells of the same cell type across batches than MNN method (Appendix B: Fig. B.2a-c and e-g), consistent with the best/lowest F values from the SMNN-corrected results F value, compared with uncorrected and MNN-corrected data. Specifically, SMNN reduced the differentiation between the two batches by 89.7% to 94.0% (73.9% to 96.7%) on top of the MNN corrected results when using label information for only one (two) of the three cell types, respectively (Appendix B: Fig. B.2d and h). These results indicate that SMNN is robust and superior even with partial cell type annotation information.

Furthermore, to assess whether the differentially expressed genes (DEGs) identified in SMNN and MNN corrected data were sensitive to the significance threshold selected, we show ROC curves to illustrate the ability of identifying DEGs among CMP, GMP and MEP cells after SMNN and MNN correction, at various adjusted p-value threshold, treating those identified in uncorrected batch 1 as working truth. The results showed that, in almost all cases, SMNN outperformed MNN (Appendix B: Fig. B.9). The only exception is for DEGs up-regulated in GMP when compared to MEP, where the ROC curves of SMNN and MNN are comparable (Appendix B: Fig. B.9e). These results suggest that our DEG analysis is not sensitive to the p-value cutoff we use for DEG detection.

Lastly, when the cell population compositions are unbalanced between two batches, we extracted the three major cell types out from two hematopoietic datasets, in order to construct two new batches with a more apparent difference in cell group composition. In the new batch 1, the proportion of CMP, GMP and MEP cells are 40.4%, 15.1% and 44.5%, while those are 17.6%, 42.3% and 40.1% in new batch 2. Then we performed both MNN and SMNN correction across the new batches. The results showed that SMNN outperformed MNN that SMNN reduced the differentiation between the two batches (measured by the F value) by 11.3% on top of the MNN corrected results (Appendix B: Fig. B.11), indicating that SMNN is robust to the cell population composition.

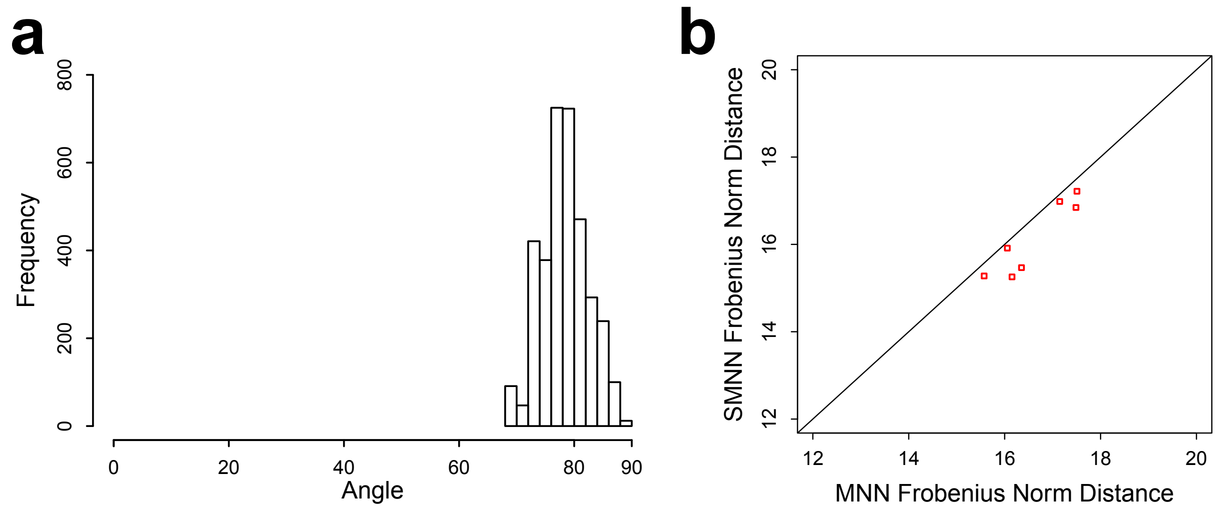


**Figure B.1: Performance of SMNN and MNN in the simulation data using the model in Haghverdi et al. (2018).** (a) Histogram of the angles (surrogate for orthogonality) for the simulation data using the model in Haghverdi et al. (2018). (b) Frobenius norm distance between two batches after SMNN and MNN correction in simulation data under orthogonal (left) and non-orthogonal scenarios (right).

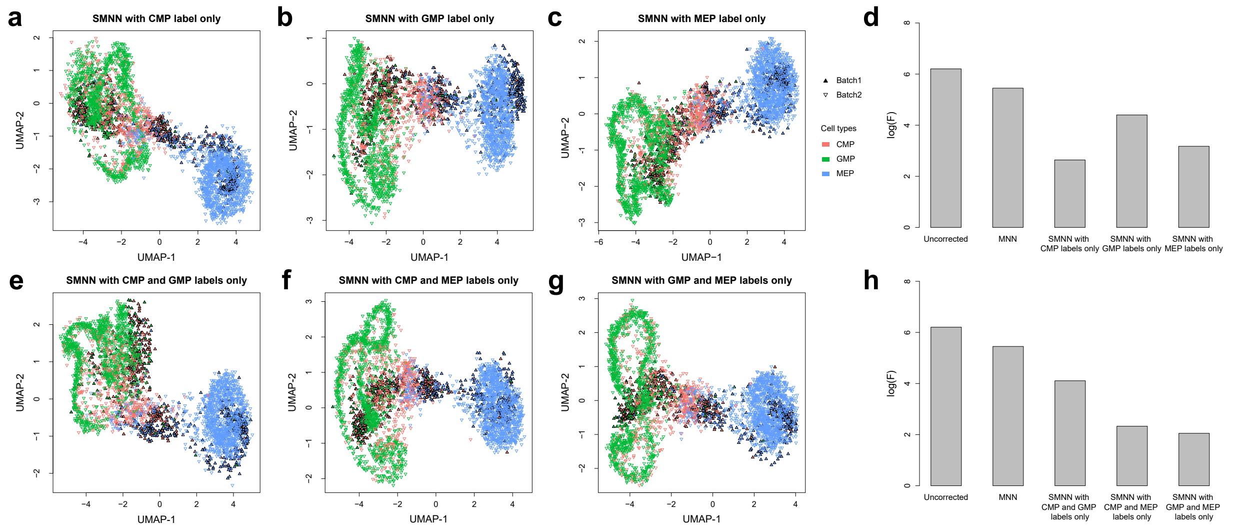


**Figure B.2: Performance of SMNN in two hematopoietic datasets with partial cell type labels.** (a-c) UMAP plots for two hematopoietic datasets after SMNN correction with the cell type label only from CMP, GMP and MEP cells, respectively. Solid and inverted triangle represent the first and second batch, respectively; and different cell types are shown in different colors. (d) Logarithms of F-statistics for merged data of the two batches before and after correction with MNN and SMNN. SMNN correction was performed with the cell type label only from CMP, GMP and MEP cells, respectively. (e-g) UMAP plots for the two hematopoietic datasets after SMNN correction with cell type label only from any two of the three cell types, CMP, GMP and MEP cells. (h) Logarithms of F-statistics for merged data of the two batches before and after correction with MNN and SMNN. SMNN correction was performed with the cell type label only from any of the three cell types, CMP, GMP and MEP cells.

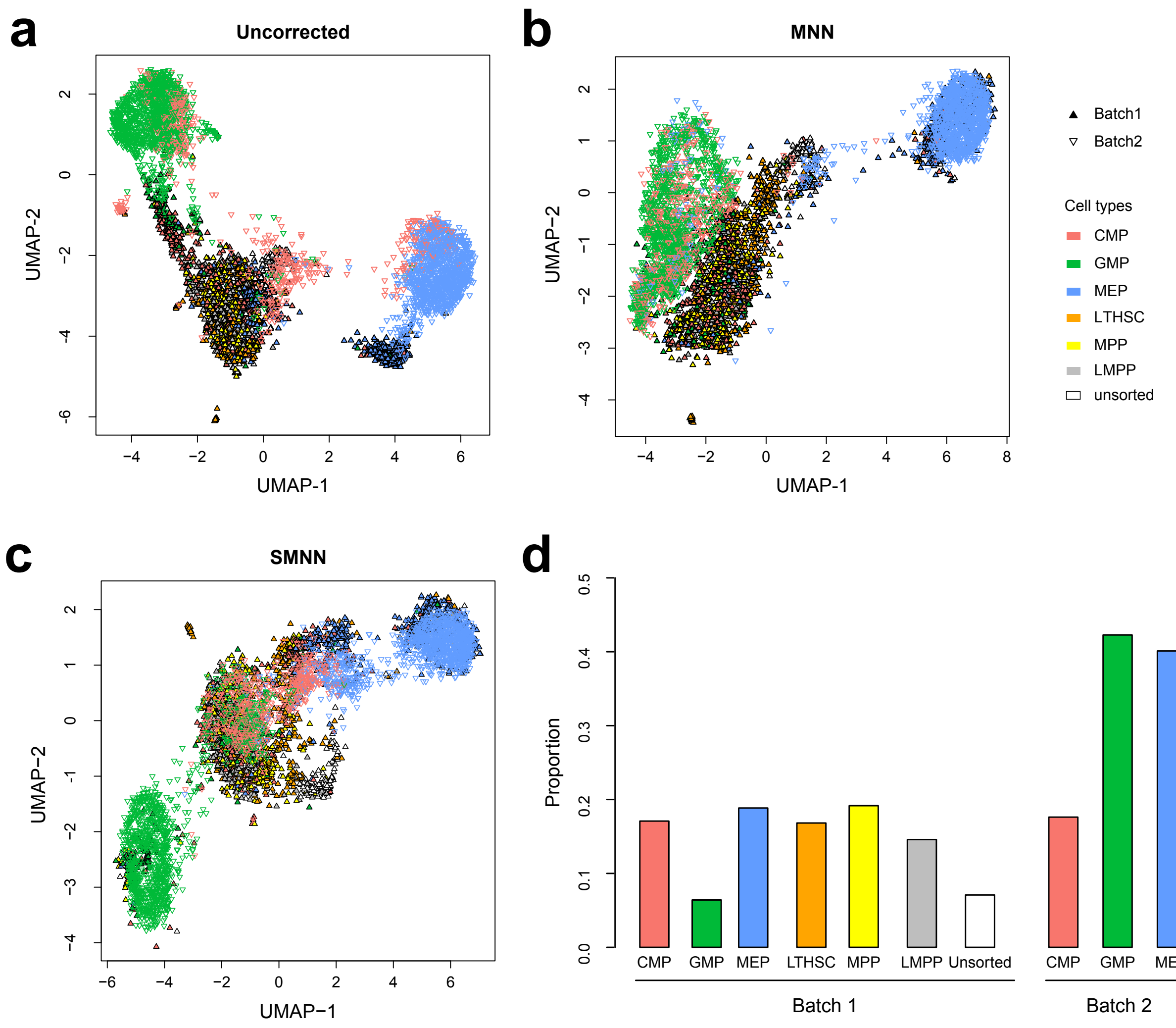


**Figure B.3: UMAP plots for all the cell types in two hematopoietic datasets.** (a), (b) and (c) corresponds to uncorrected, MNN-corrected and SMNN corrected results, respectively. (d) Cell type proportions in the two batches.

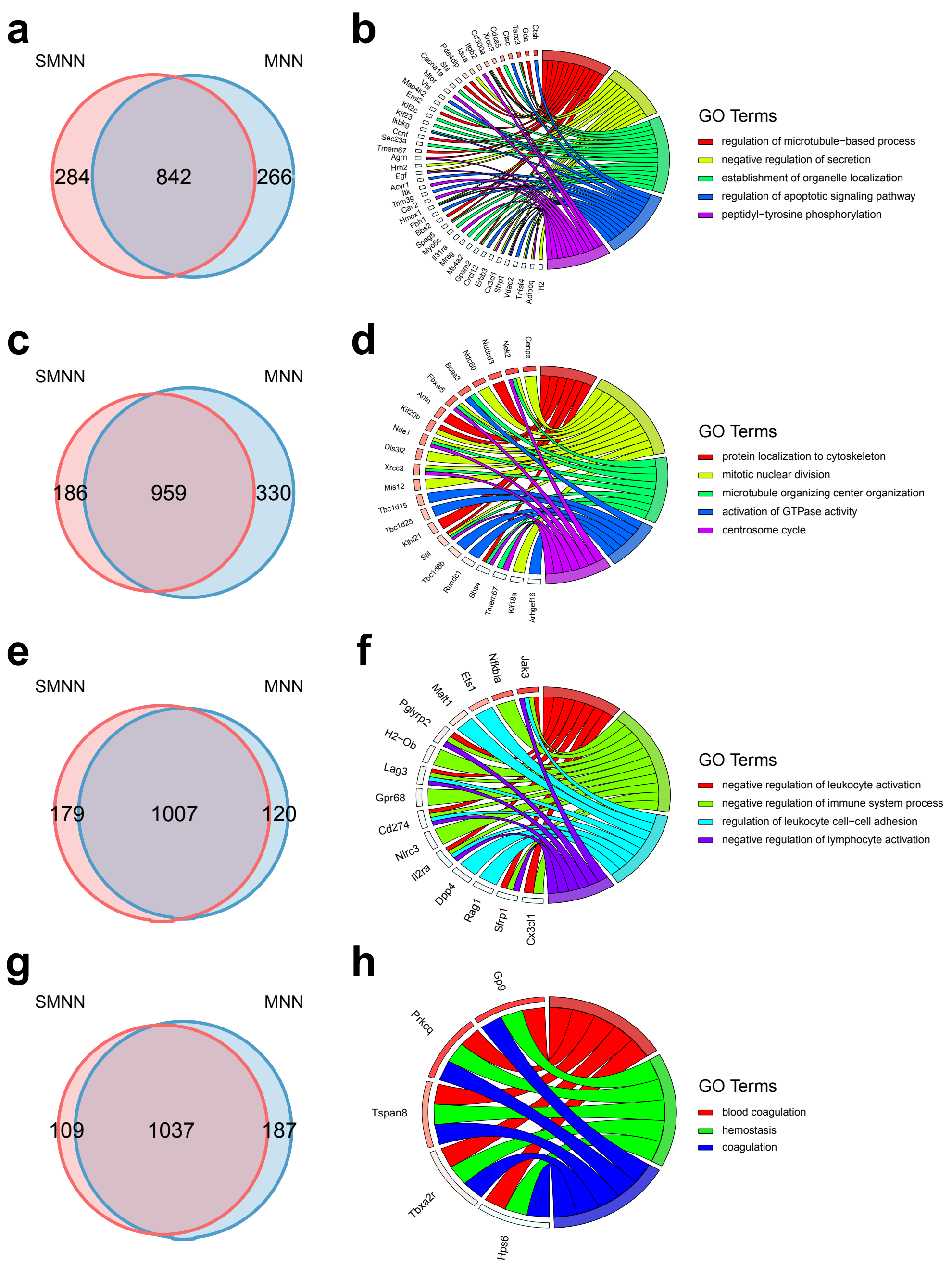


**Figure B.4: Comparison of differentially expressed genes (DEGs), identified in the merged dataset by pooling batch 1 data with batch 2 data after SMNN and MNN correction.** (a) Overlap of DEGs up-regulated in GMP over CMP after SMNN and MNN correction. (b) Feature enriched GO terms and the corresponding DEGs up-regulated in GMP over CMP. (c) Overlap of DEGs up-regulated in MEP over CMP after SMNN and MNN correction. (d) Feature enriched GO terms and the corresponding DEGs up-regulated in MEP over CMP. (e) Overlap of DEGs up-regulated in GMP over MEP after SMNN and MNN correction. (f) Feature enriched GO terms and the corresponding DEGs up-regulated in GMP over MEP. (g) Overlap of DEGs up-regulated in MEP over GMP after SMNN and MNN correction. (h) Feature enriched GO terms and the corresponding DEGs up-regulated in MEP over GMP.

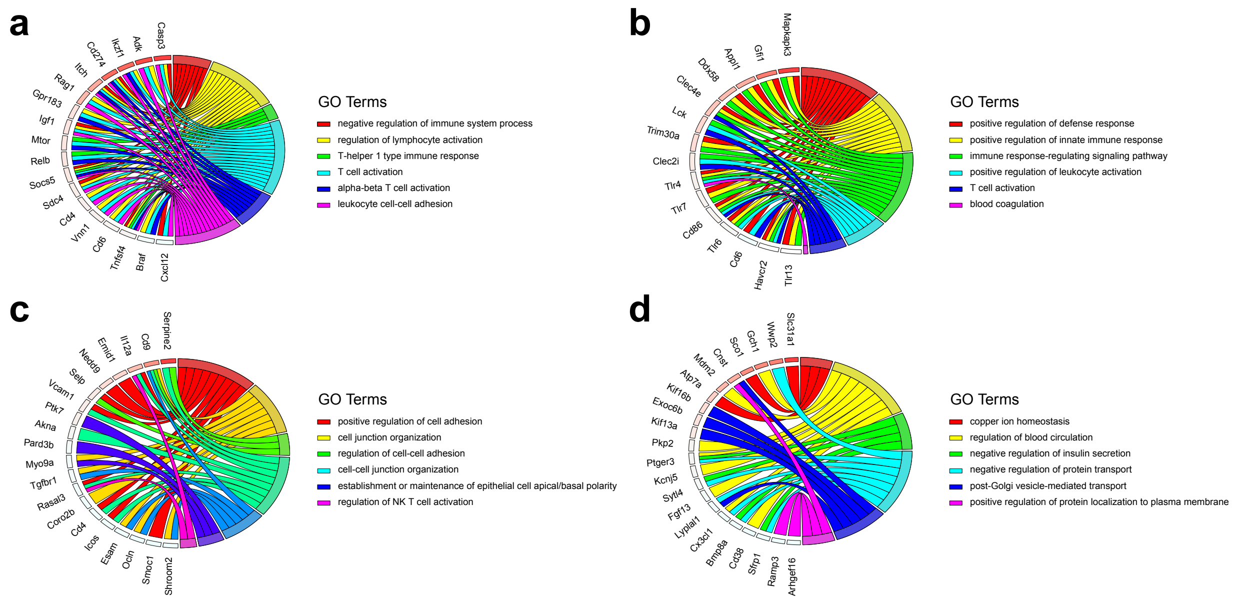

a
Casp3
Adk
Ikzf1
Cd274
Itch
Rag1
Gpr183
Igf1
Mtor
Relb
Socs5
Sdc4
Cd4
Vnn1
Cd6
Tnfsf4
Braf
Cxcl12
GO Terms
negative regulation of immune system process
regulation of lymphocyte activation
T-helper 1 type immune response
T cell activation
alpha-beta T cell activation
leukocyte cell-cell adhesion
b
Mapkapk3
Gfi1
Appl1
Ddx58
Clec4e
Lck
Trim30a
Clec2i
Tlr4
Tlr7
Cd86
Tlr6
Cd6
Havcr2
Tlr13
GO Terms
positive regulation of defense response
positive regulation of innate immune response
immune response-regulating signaling pathway
positive regulation of leukocyte activation
T cell activation
blood coagulation
c
Serpine2
Cd9
Il12a
Emid1
Nedd9
Selp
Vcam1
Ptk7
Akna
Pard3b
Myo9a
Tgfbr1
Rasal3
Coro2b
Cd4
Icos
Esam
Ocln
Smoc1
Shroom2
GO Terms
positive regulation of cell adhesion
cell junction organization
regulation of cell-cell adhesion
cell-cell junction organization
establishment or maintenance of epithelial cell apical/basal polarity
regulation of NK T cell activation
d
Slc31a1
Wwp2
Gch1
Sco1
Cnst
Mdm2
Atp7a
Kif16b
Exoc6b
Kif13a
Pkp2
Ptger3
Kcnj5
Sytl4
Fgf13
Lypla1
Cx3cl1
Bmp8a
Cd38
Sfrp1
Ramp3
Arhgef16
GO Terms
copper ion homeostasis
regulation of blood circulation
negative regulation of insulin secretion
negative regulation of protein transport
post-Golgi vesicle-mediated transport
positive regulation of protein localization to plasma membrane

**Figure B.5: Feature enriched GO terms and the corresponding differentially expressed genes (DEGs) specifically identified in the merged dataset by pooling batch 1 data with batch 2 data after MNN correction.** (a) Feature enriched GO terms and the corresponding DEGs up-regulated in CMP over GMP. (b) Feature enriched GO terms and the corresponding DEGs up-regulated in GMP over CMP. (c) Feature enriched GO terms and the corresponding DEGs up-regulated in GMP over MEP. (d) Feature enriched GO terms and the corresponding DEGs up-regulated in MEP over GMP.

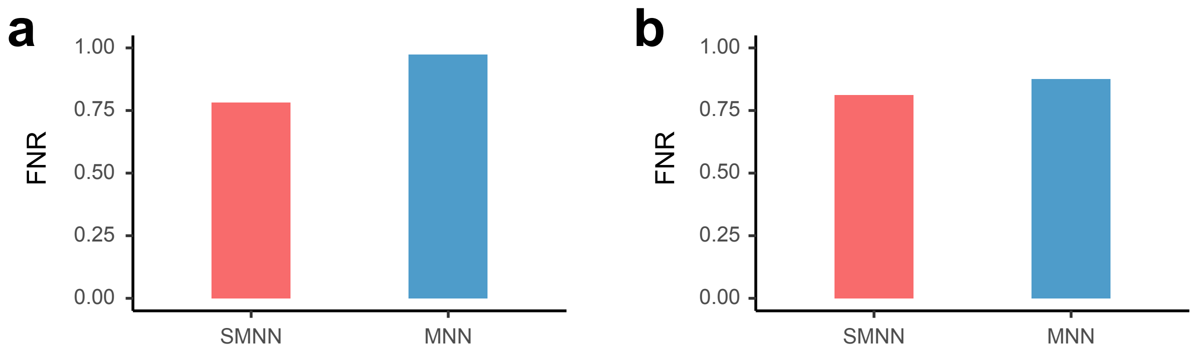


**Figure B.6: False negative rate (FNR) of DEGs (between CMP and GMP), identified in uncorrected batch 1 and in SMNN or MNN-corrected batch 2.** (a) FNR of the DEGs up-regulated in CMP over GEP identified in batch 2 after SMNN and MNN correction. (b) FNR of the DEGs up-regulated in GMP over CMP identified in batch 2 after SMNN and MNN correction.

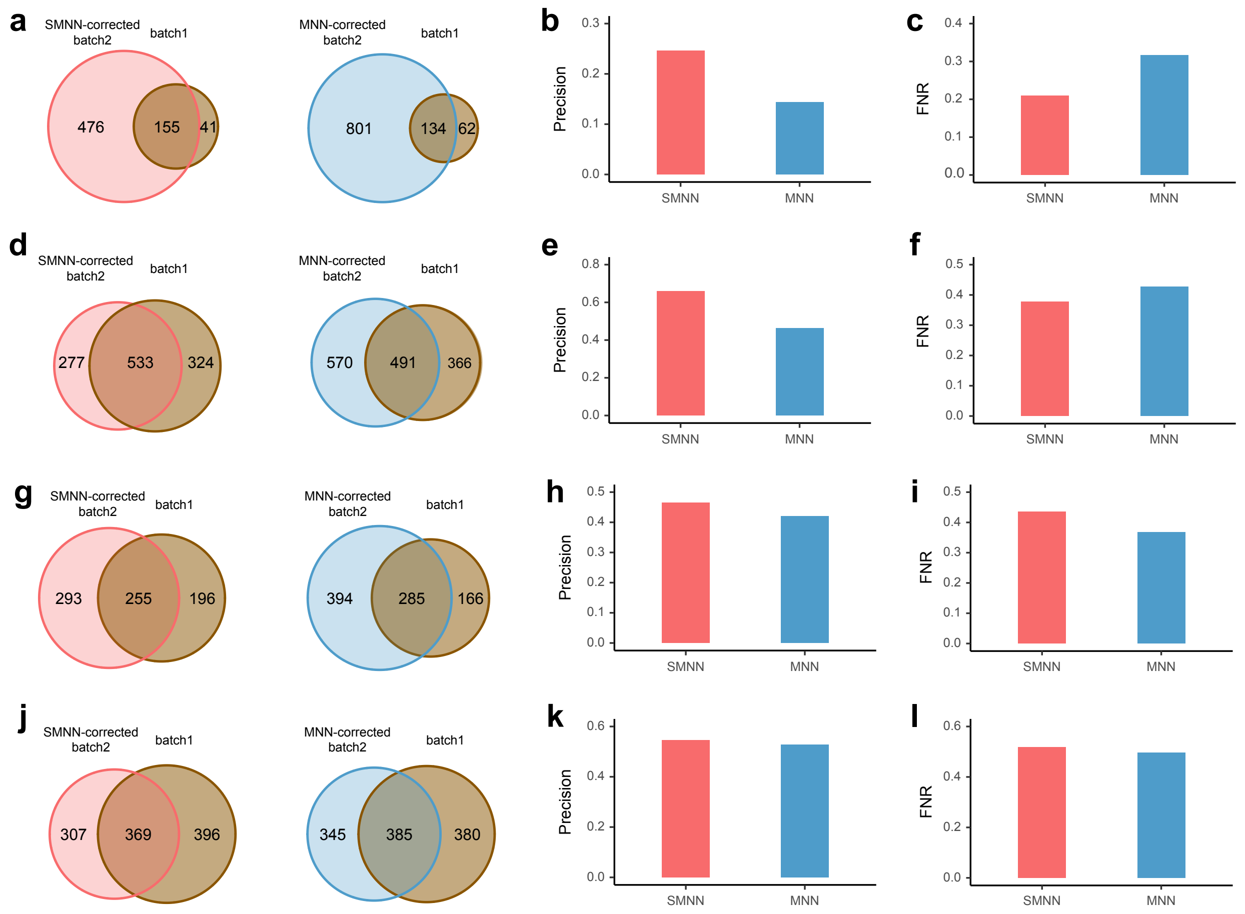


**Figure B.7: Reproducibility of DEGs (between CMP and MEP and between GMP and MEP).** (a) Reproducibility of DEGs up-regulated in CMP over MEP. (b) and (c) Precision and false negative rate (FNR) of the DEGs up-regulated in CMP over MEP. (d) Reproducibility of DEGs up-regulated in MEP over CMP. (e) and (f) Precision and FNR of the DEGs up-regulated in MEP over CMP. (g) Reproducibility of DEGs up-regulated in GMP over MEP. (h) and (i) Precision and FNR of the DEGs up-regulated in GMP over MEP. (j) Reproducibility of DEGs up-regulated in MEP over GMP. (k) and (l) Precision and FNR of the DEGs up-regulated in MEP over GMP.

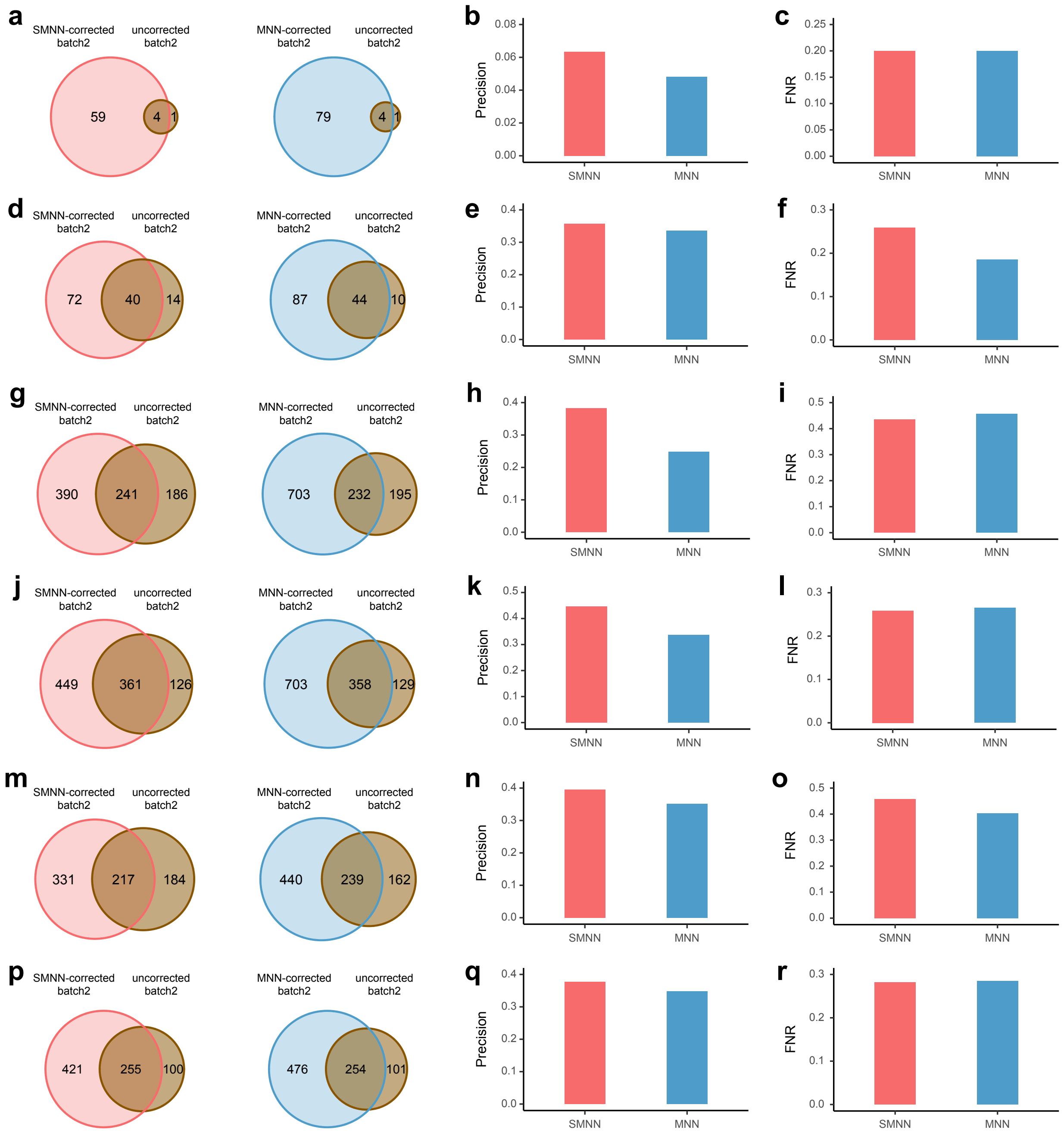


**Figure B.8: Reproducibility of DEGs (between any two cell types, out of the three: CMP, GMP and MEP).** (a) reproducibility for DEGs up-regulated in CMP over GMP. (b) and (c) Precision and false negative rate (FNR) of the DEGs up-regulated in CMP over GMP. (d) Reproducibility of DEGs up-regulated in GMP over CMP. (e) and (f) Precision and FNR of the DEGs up-regulated in GMP over CMP. (g) Reproducibility of DEGs up-regulated in CMP over MEP. (h) and (i) Precision and FNR of the DEGs up-regulated in CMP over MEP. (j) Reproducibility of DEGs up-regulated in MEP over CMP. (k) and (l) Precision and FNR of the DEGs up-regulated in MEP over CMP. (m) Reproducibility of DEGs up-regulated in GMP over MEP. (n) and (o) Precision and FNR of the DEGs up-regulated in GMP over MEP. (p) Reproducibility of DEGs up-regulated in MEP over GMP. (q) and (r) Precision and FNR of the DEGs up-regulated in MEP over GMP.

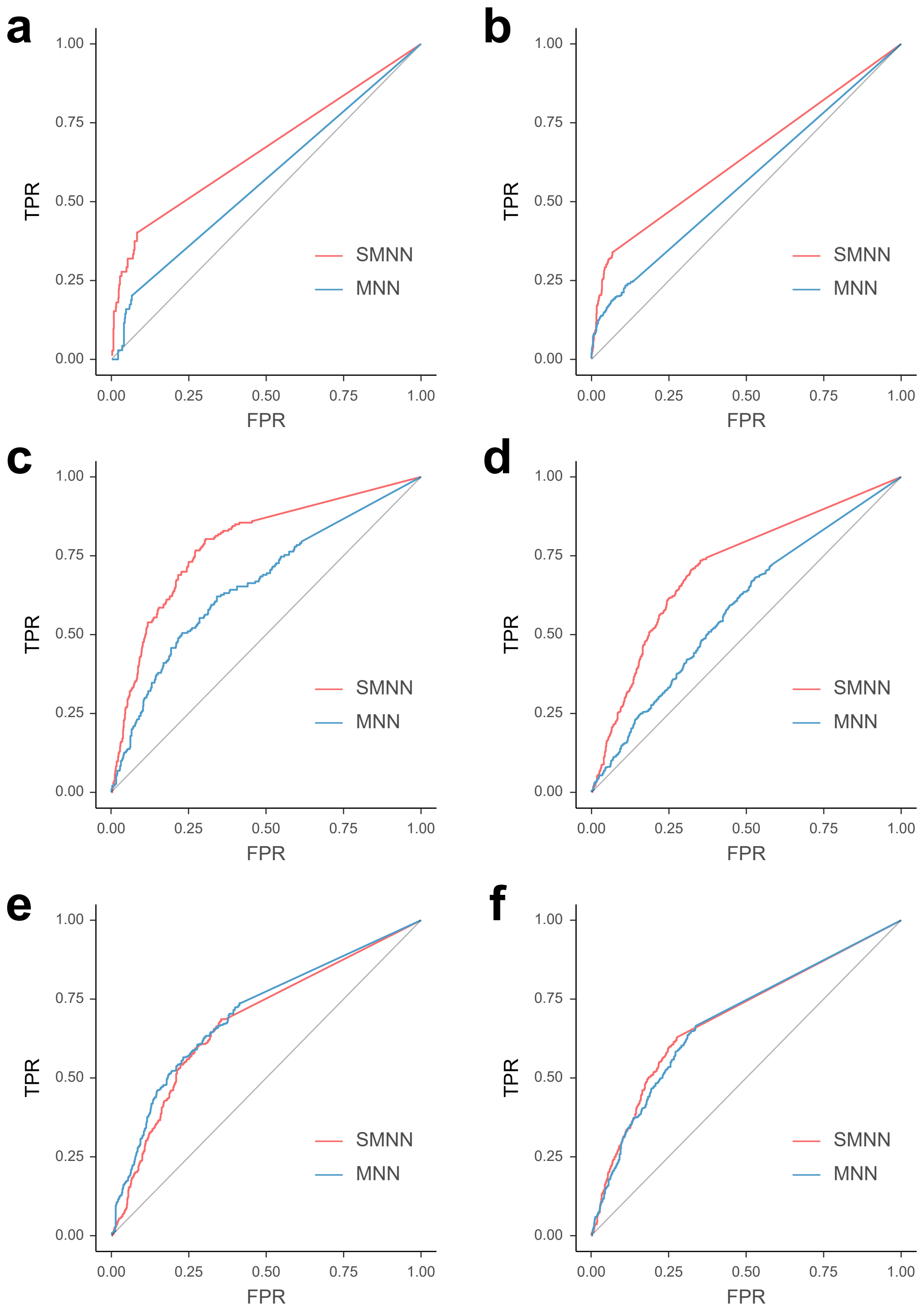


**Figure B.9: ROC curves for DEGs (between any two cell types, out of the three: CMP, GMP and MEP) at various adjusted p-value thresholds.** (a) ROC curve for DEGs up-regulated in CMP over GMP. (b) ROC curve for DEGs up-regulated in GMP over CMP. (c) ROC curve for DEGs up-regulated in CMP over MEP. (d) ROC curve for DEGs up-regulated in MEP over CMP. (e) ROC curve for DEGs up-regulated in GMP over MEP. (f) ROC curve for DEGs up-regulated in MEP over GMP.

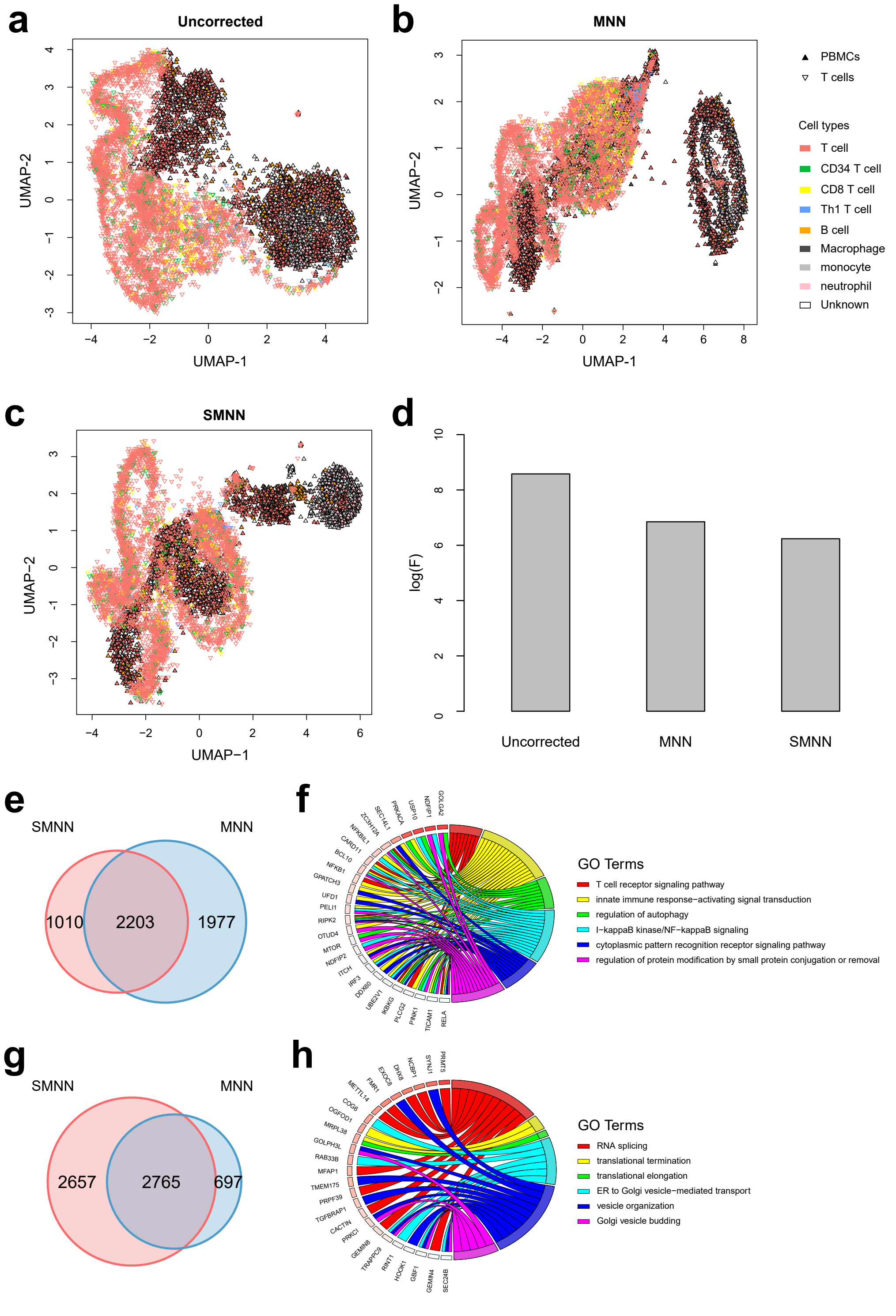


**Figure B.10: Performance comparison between SMNN and MNN in two 10X Genomics datasets for human PBMC and T cells.**(a-c) UMAP plots for PBMC and T cells datasets before and after batch effect correction with MNN and SMNN, respectively. Solid and inverted triangle represent PBMC and T cell datasets, respectively; and different cell types are shown in different colors. (d) Logarithms of F-statistics for merged data of the two batches. (e) Overlap of DEGs up-regulated in T cells over B cells after SMNN and MNN correction. (f) Feature enriched GO terms and the corresponding DEGs up-regulated in T cells over B cells. (g) Overlap of DEGs up-regulated in B cells over T cells after SMNN and MNN correction. (h) Feature enriched GO terms and the corresponding DEGs up-regulated in B cells over T cells.

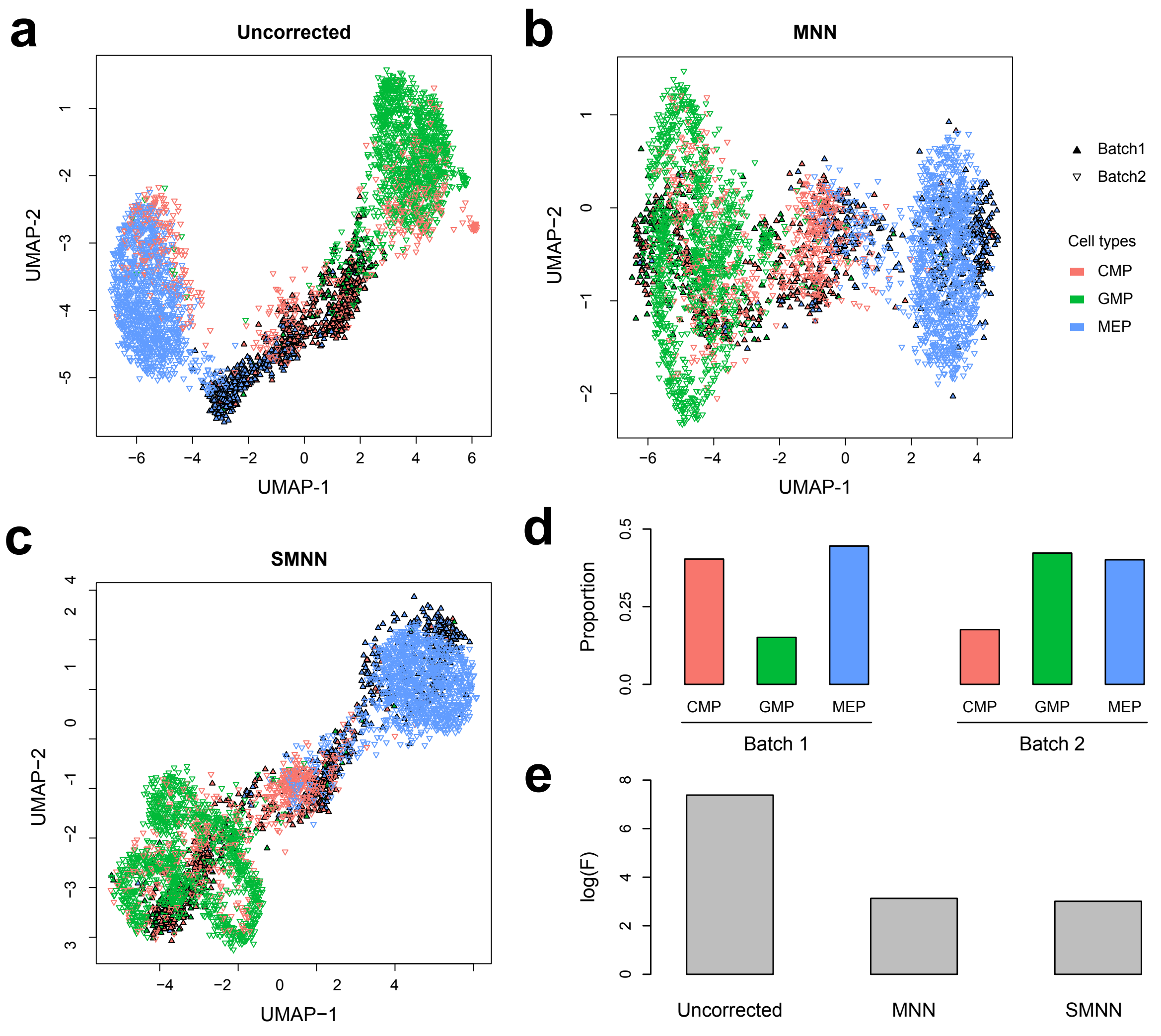


**Figure B.11: Performance of SMNN in two hematopoietic datasets with unbalanced cell population composition.** (a-c) UMAP plots for two hematopoietic datasets before and after correction with MNN and SMNN, respectively. Solid and inverted triangle represent the first and second batch, respectively; and different cell types are shown in different colors. (d) Cell population proportions in the two batches. (e) Logarithms of F-statistics for merged data of the two batches before and after correction with MNN and SMNN.

# APPENDIX C
# ADDITIONAL RESULTS FOR CUE

**Table C.1:** Numbers and ratios of well-imputed sites. Quality control (QC) criterias for PTSD: RMSE $< 0.05$ and Accruracy $> 95\%$; QC for ELGAN: RMSE $< 0.1$ and Accruracy $> 90\%$. The ratios are out of 339,033 HM850-only CpG sites.

| | PTSD (%) | ELGAN (%) |
|---|---|---|
| KNN | 249,425 (73.6%) | 181,304 (53.5%) |
| Logistic | D.N.C. | 93,470 (27.6%) |
| PFR | 269,745 (79.6%) | 213,355 (62.9%) |
| RF | 249,425 (73.6%) | 236,645 (69.8%) |
| XGBoost | 285,330 (84.2%) | 224,317 (66.2%) |
| CUE | **289,604 (85.4%)** | **238,090 (70.2%)** |

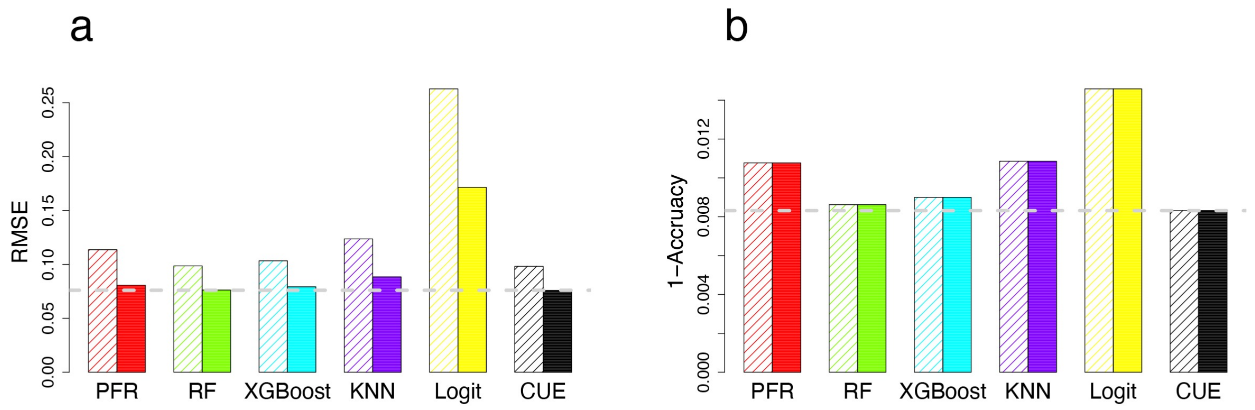


**Figure C.1: Imputed Performances Before (Left Bar) and After QC (Right Bar) for QCed Probes in ELGAN.** (a) RMSE comparisons. (b) 1- Accuracy (classification error) comparisons. Different colors represent different methods. The horizontal dash line is the lowest value corresponding to the best method.

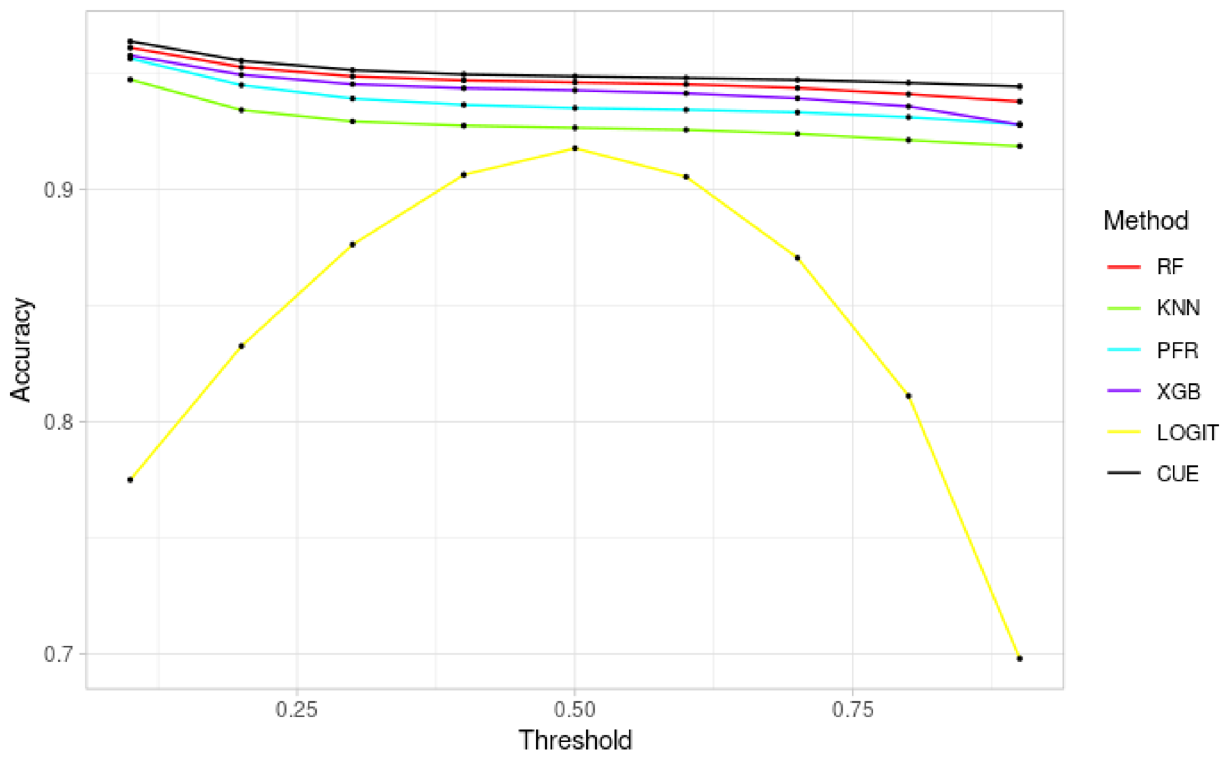


**Figure C.2: Imputation accuracy across different thresholds in the ELGAN dataset.** Our CUE method (black line) achieves the highest accuracy across all thresholds. Logistic regression (yellow line) seems sensitive to the threshold probably because it is trained based on labels defined at the cutoff of 0.5.

# BIBLIOGRAPHY

Angermueller, C., Lee, H. J., Reik, W., and Stegle, O. (2017). Deepcpg: accurate prediction of single-cell dna methylation states using deep learning. *Genome biology*, 18(1):67.

Aryee, M. J., Jaffe, A. E., Corrada-Bravo, H., Ladd-Acosta, C., Feinberg, A. P., Hansen, K. D., and Irizarry, R. A. (2014). Minfi: a flexible and comprehensive bioconductor package for the analysis of infinium dna methylation microarrays. *Bioinformatics*, 30(10):1363–1369.

Bakusic, J., Schaufeli, W., Claes, S., and Godderis, L. (2017). Stress, burnout and depression: A systematic review on dna methylation mechanisms. *Journal of psychosomatic research*, 92:34–44.

Bardsley, J. M., Solonen, A., Haario, H., and Laine, M. (2014). Randomize-then-optimize: A method for sampling from posterior distributions in nonlinear inverse problems. *SIAM Journal on Scientific Computing*, 36(4).

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers Inc.

Bhasin, M., Zhang, H., Reinherz, E. L., and Reche, P. A. (2005). Prediction of methylated cpgs in dna sequences using a support vector machine. *FEBS letters*, 579(20):4302–4308.

Bibikova, M., Barnes, B., Tsan, C., Ho, V., Klotzle, B., Le, J. M., Delano, D., Zhang, L., Schroth, G. P., and Gunderson, K. L. (2011a). High density dna methylation array with single cpg site resolution. *Genomics*, 98(4):288–295.

Bibikova, M., Barnes, B., Tsan, C., Ho, V., Klotzle, B., Le, J. M., Delano, D., Zhang, L., Schroth, G. P., Gunderson, K. L., et al. (2011b). High density dna methylation array with single cpg site resolution. *Genomics*, 98(4):288–295.

Bibikova, M., Le, J., Barnes, B., Saedinia-Melnyk, S., Zhou, L., Shen, R., and Gunderson, K. L. (2009). Genome-wide dna methylation profiling using infinium® assay. *Epigenomics*, 1(1):177–200.

Bird, A. (2002). Dna methylation patterns and epigenetic memory. *Genes & development*, 16(1):6–21.

Birnbaum, A. (1961). On the foundations of statistical inference: binary experiments. *The Annals of Mathematical Statistics*, pages 414–435.

Blum, M. G., Nunes, M. A., Prangle, D., Sisson, S. A., et al. (2013). A comparative review of dimension reduction methods in approximate bayesian computation. *Statistical Science*, 28(2):189–208.

Bock, C., Paulsen, M., Tierling, S., Mikeska, T., Lengauer, T., and Walter, J. (2006). Cpg island methylation in human lymphocytes is highly correlated with dna sequence, repeats, and predicted dna structure. *PLoS Genet*, 2(3):e26.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Bryk, M., Briggs, S. D., Strahl, B. D., Curcio, M. J., Allis, C. D., and Winston, F. (2002). Evidence that set1, a factor required for methylation of histone h3, regulates rdna silencing in s. cerevisiae by a sir2-independent mechanism. *Current Biology*, 12(2):165–170.

Butler, A., Hoffman, P., Smibert, P., Papalexi, E., and Satija, R. (2018). Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nature biotechnology*, 36(5):411.

Bø, T. H., Dysvik, B., and Jonassen, I. (2004). Lsimpute: accurate estimation of missing values in microarray data with least squares methods. *Nucleic acids research*, 32(3):e34–e34.

Chen, M. and Zhou, X. (2017). Controlling for confounding effects in single cell rna sequencing studies using both control and target genes. *Scientific reports*, 7(1):13587.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Das, R., Dimitrova, N., Xuan, Z., Rollins, R. A., Haghighi, F., Edwards, J. R., Ju, J., Bestor, T. H., and Zhang, M. Q. (2006). Computational prediction of methylation status in human genomic sequences. *Proceedings of the National Academy of Sciences*, 103(28):10713–10716.

Dedeurwaerder, S., Defrance, M., Bizet, M., Calonne, E., Bontempi, G., and Fuks, F. (2014). A comprehensive overview of infinium humanmethylation450 data processing. *Briefings in bioinformatics*, 15(6):929–941.

Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

Down, T. A., Rakyan, V. K., Turner, D. J., Flicek, P., Li, H., Kulesha, E., Graef, S., Johnson, N., Herrero, J., Tomazou, E. M., et al. (2008). A bayesian deconvolution strategy for immunoprecipitation-based dna methylome analysis. *Nature biotechnology*, 26(7):779–785.

Duò, A., Robinson, M. D., and Soneson, C. (2018). A systematic performance evaluation of clustering methods for single-cell rna-seq data. *F1000Research*, 7.

Fang, F., Fan, S., Zhang, X., and Zhang, M. Q. (2006). Predicting methylation status of cpg islands in the human brain. *Bioinformatics*, 22(18):2204–2209.

Fisher, R. A. (1930). Inverse probability. *Mathematical Proceedings of the Cambridge Philosophical Society*, 26(4):528–535.

Goldsmith, J., Bobb, J., Crainiceanu, C. M., Caffo, B., and Reich, D. (2011). Penalized functional regression. *Journal of Computational and Graphical Statistics*, 20(4):830–851.

Gonzalo, S. (2010). Epigenetic alterations in aging. *Journal of applied physiology*, 109(2):586–597.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Grün, D., Muraro, M. J., Boisset, J.-C., Wiebrands, K., Lyubimova, A., Dharmadhikari, G., van den Born, M., van Es, J., Jansen, E., and Clevers, H. (2016). De novo prediction of stem cell identity using single-cell transcriptome data. *Cell stem cell*, 19(2):266–277.

Haghverdi, L., Lun, A. T., Morgan, M. D., and Marioni, J. C. (2018). Batch effects in single-cell rna-sequencing data are corrected by matching mutual nearest neighbors. *Nature biotechnology*, 36(5):421.

Hannig, J., Iyer, H., Lai, R. C., and Lee, T. C. (2016). Generalized fiducial inference: A review and new results. *Journal of the American Statistical Association*, 111(515):1346–1361.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Horvath, S. and Raj, K. (2018). Dna methylation-based biomarkers and the epigenetic clock theory of ageing. *Nature Reviews Genetics*, 19(6):371–384.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of classification*, 2(1):193–218.

Huh, R., Yang, Y., Jiang, Y., Shen, Y., and Li, Y. (2019). Same-clustering: Single-cell aggregated clustering via mixture model ensemble. *bioRxiv*, page 645820.

Iurlaro, M., von Meyenn, F., and Reik, W. (2017). Dna methylation homeostasis in human and mouse development. *Current opinion in genetics & development*, 43:101–109.

Johnson, W. E., Li, C., and Rabinovic, A. (2007). Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127.

Joubert, B. R., Felix, J. F., Yousefi, P., Bakulski, K. M., Just, A. C., Breton, C., Reese, S. E., Markunas, C. A., Richmond, R. C., Xu, C.-J., et al. (2016). Dna methylation in newborns and maternal smoking in pregnancy: genome-wide consortium meta-analysis. *The American Journal of Human Genetics*, 98(4):680–696.

Kim, H., Golub, G. H., and Park, H. (2006). Missing value estimation for dna microarray gene expression data: local least squares imputation. *Bioinformatics*, 22(11):1410–1411.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kiselev, V. Y., Andrews, T. S., and Hemberg, M. (2019). Challenges in unsupervised clustering of single-cell rna-seq data. *Nature Reviews Genetics*, page 1.

Klutstein, M., Nejman, D., Greenfield, R., and Cedar, H. (2016). Dna methylation in cancer and aging. *Cancer research*, 76(12):3446–3450.

Laird, P. W. (2010). Principles and challenges of genome-wide dna methylation analysis. *Nature Reviews Genetics*, 11(3):191.

Laurent, L., Wong, E., Li, G., Huynh, T., Tsirigos, A., Ong, C. T., Low, H. M., Sung, K. W. K., Rigoutsos, I., and Loring, J. (2010). Dynamic changes in the human methylome during differentiation. *Genome research*, 20(3):320–331.

Leek, J. T. (2014). Svaseq: removing batch effects and other unwanted noise from sequencing data. *Nucleic acids research*, 42(21):e161–e161.

Liew, A. W.-C., Law, N.-F., and Yan, H. (2010). Missing value imputation for gene expression data: computational techniques to recover missing data from available information. *Briefings in bioinformatics*, 12(5):498–513.

Lister, R., Pelizzola, M., Kida, Y. S., Hawkins, R. D., Nery, J. R., Hon, G., Antosiewicz-Bourget, J., O'malley, R., Castanon, R., and Klugman, S. (2011). Hotspots of aberrant epigenomic reprogramming in human induced pluripotent stem cells. *Nature*, 471(7336):68.

Liu, Z., Xiao, X., Qiu, W.-R., and Chou, K.-C. (2015). idna-methyl: Identifying dna methylation sites via pseudo trinucleotide composition. *Analytical biochemistry*, 474:69–77.

Logue, M. W., Smith, A. K., Wolf, E. J., Maniates, H., Stone, A., Schichman, S. A., McGlinchey, R. E., Milberg, W., and Miller, M. W. (2017). The correlation of methylation levels measured using illumina 450k and epic beadchips in blood samples. *Epigenomics*, 9(11):1363–1371.

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Meissner, A., Gnirke, A., Bell, G. W., Ramsahoye, B., Lander, E. S., and Jaenisch, R. (2005). Reduced representation bisulfite sequencing for comparative high-resolution dna methylation analysis. *Nucleic acids research*, 33(18):5868–5877.

Moran, S., Arribas, C., and Esteller, M. (2016). Validation of a dna methylation microarray for 850,000 cpg sites of the human genome enriched in enhancer sequences. *Epigenomics*, 8(3):389–399.

Muraro, M. J., Dharmadhikari, G., Grün, D., Groen, N., Dielen, T., Jansen, E., van Gurp, L., Engelse, M. A., Carlotti, F., and de Koning, E. J. (2016). A single-cell transcriptome atlas of the human pancreas. *Cell systems*, 3(4):385–394. e3.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Nestorowa, S., Hamey, F. K., Sala, B. P., Diamanti, E., Shepherd, M., Laurenti, E., Wilson, N. K., Kent, D. G., and Göttgens, B. (2016). A single-cell resolution map of mouse hematopoietic stem and progenitor cell differentiation. *Blood*, 128(8):e20–e31.

Network, C. G. A. (2012). Comprehensive molecular portraits of human breast tumours. *Nature*, 490(7418):61.

Network, C. G. A. (2015). Comprehensive genomic characterization of head and neck squamous cell carcinomas. *Nature*, 517(7536):576.

Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Winter, D., Lara-Astiaso, D., Gury, M., and Weiner, A. (2015). Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell*, 163(7):1663–1677.

Rozenblatt-Rosen, O., Stubbington, M. J., Regev, A., and Teichmann, S. A. (2017). The human cell atlas: from vision to reality. *Nature News*, 550(7677):451.

Ruike, Y., Imanaka, Y., Sato, F., Shimizu, K., and Tsujimoto, G. (2010). Genome-wide analysis of aberrant methylation in human breast cancer cells using methyl-dna immunoprecipitation combined with high-throughput sequencing. *BMC genomics*, 11(1):1–11.

Sandoval, J., Heyn, H., Moran, S., Serra-Musach, J., Pujana, M. A., Bibikova, M., and Esteller, M. (2011). Validation of a dna methylation microarray for 450,000 cpg sites in the human genome. *Epigenetics*, 6(6):692–702.

Santos, H. P., J., Bhattacharya, A., Martin, E. M., Addo, K., Psioda, M., Smeester, L., Joseph, R. M., Hooper, S. R., Frazier, J. A., Kuban, K. C., O'Shea, T. M., and Fry, R. C. (2019). Epigenome-wide dna methylation in placentas from preterm infants: association with maternal socioeconomic status. *Epigenetics*, 14(8):751–765. Santos, Hudson P Jr Bhattacharya, Arjun Martin, Elizabeth M Addo, Kezia Psioda, Matt Smeester, Lisa Joseph, Robert M Hooper, Stephen R Frazier, Jean A Kuban, Karl C O'Shea, T Michael Fry, Rebecca C K23 NR017898/NR/NINR NIH HHS/United States U01 NS040069/NS/NINDS NIH HHS/United States UG3 OD023348/OD/NIH HHS/United States United States Epigenetics Epigenetics. 2019 Aug;14(8):751-765. doi: 10.1080/15592294.2019.1614743. Epub 2019 May 21.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Smyth, G. K. (2005). Limma: linear models for microarray data. In *Bioinformatics and computational biology solutions using R and Bioconductor*, pages 397–420. Springer.

Spitzer, M. H., Gherardini, P. F., Fragiadakis, G. K., Bhattacharya, N., Yuan, R. T., Hotson, A. N., Finck, R., Carmi, Y., Zunder, E. R., and Fantl, W. J. (2015). An interactive reference framework for modeling a dynamic immune system. *Science*, 349(6244):1259425.

Stan Development Team (2020). RStan: the R interface to Stan. R package version 2.19.3.

Stegle, O., Teichmann, S. A., and Marioni, J. C. (2015). Computational and analytical challenges in single-cell transcriptomics. *Nature Reviews Genetics*, 16(3):133.

Stuart, T. and Satija, R. (2019a). Integrative single-cell analysis. *Nat Rev Genet*, 20(5):257–272. Stuart, Tim Satija, Rahul Research Support, N.I.H., Extramural Research Support, Non-U.S. Gov't Review England Nature reviews. Genetics Nat Rev Genet. 2019 May;20(5):257-272. doi: 10.1038/s41576-019-0093-7.

Stuart, T. and Satija, R. (2019b). Integrative single-cell analysis. *Nature Reviews Genetics*, page 1.

Sun, Z., Chen, L., Xin, H., Jiang, Y., Huang, Q., Cillo, A. R., Tabib, T., Kolls, J. K., Bruno, T. C., and Lafyatis, R. (2019). A bayesian mixture model for clustering droplet-based single-cell transcriptomic data from population studies. *Nature communications*, 10(1):1649.

Team, R. D. C. (2008). R: A language and environment for statistical computing.

Tibshirani, R. J. and Efron, B. (1993). An introduction to the bootstrap. *Monographs on statistics and applied probability*, 57:1–436.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. (2001). Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525.

Turecki, G. and Meaney, M. J. (2016). Effects of the social environment and stress on glucocorticoid receptor gene methylation: a systematic review. *Biological psychiatry*, 79(2):87–96.

Van Der Maaten, L. (2014). Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245.

Van Loan, C. F. and Golub, G. H. (1983). *Matrix computations*. Johns Hopkins University Press.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408.

Welch, J. D., Kozareva, V., Ferreira, A., Vanderburg, C., Martin, C., and Macosko, E. Z. (2019). Single-cell multi-omic integration compares and contrasts features of brain cell identity. *Cell*, 177(7):1873–1887 e17.

Yang, Y., Huh, R., Culpepper, H. W., Lin, Y., Love, M. I., and Li, Y. (2019). Safe-clustering: Single-cell aggregated (from ensemble) clustering for single-cell rna-seq data. *Bioinformatics*, 35(8):1269–1277.

Yu, G., Wang, L.-G., Han, Y., and He, Q.-Y. (2012). clusterprofiler: an r package for comparing biological themes among gene clusters. *Omics: a journal of integrative biology*, 16(5):284–287.

Zhang, G., Huang, K., Xu, Z., Tzeng, J., Conneely, K. N., Guan, W., Kang, J., and Li, Y. (2016). Across-platform imputation of dna methylation levels incorporating nonlocal information using penalized functional regression. *Genetic epidemiology*, 40(4):333–340.

Zhang, W., Spector, T. D., Deloukas, P., Bell, J. T., and Engelhardt, B. E. (2015). Predicting genome-wide dna methylation using methylation marks, genomic position, and dna regulatory elements. *Genome biology*, 16(1):1–20.

Zheng, G. X., Terry, J. M., Belgrader, P., Ryvkin, P., Bent, Z. W., Wilson, R., Ziraldo, S. B., Wheeler, T. D., McDermott, G. P., and Zhu, J. (2017). Massively parallel digital transcriptional profiling of single cells. *Nature communications*, 8:14049.

Zhou, X., Li, Z., Dai, Z., and Zou, X. (2012). Prediction of methylation cpgs and their methylation degrees in human dna sequences. *Computers in biology and medicine*, 42(4):408–413.

Zhou, X. and Tuck, D. P. (2007). Msvm-rfe: extensions of svm-rfe for multiclass gene selection on dna microarray data. *Bioinformatics*, 23(9):1106–1114.

Zhu, L., Lei, J., Klei, L., Devlin, B., and Roeder, K. (2019). Semisoft clustering of single-cell data. *Proceedings of the National Academy of Sciences*, 116(2):466–471.

Zou, L. S., Erdos, M. R., Taylor, D. L., Chines, P. S., Varshney, A., Parker, S. C., Collins, F. S., and Didion, J. P. (2018). Boostme accurately predicts dna methylation values in whole-genome bisulfite sequencing of multiple human tissues. *BMC genomics*, 19(1):1–15.