Technical University of Munich
TUM School of Medicine and Health

Developing Ensemble Learning and Neural Network Architectures for Technical Variation Removal and Missing View Imputation in Multi-Timepoint Omics Data

Siyu Han

Complete reprint of the dissertation approved by the TUM School of Medicine and Health of the Technical University of Munich for the award of the

Doktor der Naturwissenschaften (Dr. rer. nat.).

Chair:
Prof. Dr. Radu Roland Rad

Examiners:
1. Prof. Dr. Eleftheria Zeggini
2. Prof. Dr. Dmitrij Frishman

The dissertation was submitted to the Technical University of Munich on 8 August 2024 and accepted by the TUM School of Medicine and Health on 4 December 2024.

Abstract

Advancements in omics technologies have provided data spanning multiple timepoints, offering unique insights into individual-level temporal dynamics and unprecedented opportunities for personalized healthcare. However, the complexity and technical challenges inherent in multi-timepoint omics data introduce issues such as technical variations and missing views, which greatly hinder the translation to healthcare applications. This thesis introduces two computational tools, TIGER and LEOPARD, developed to enhance quality control in multi-timepoint omics data, particularly within metabolomics and proteomics.

TIGER: Large multi-timepoint metabolomics datasets inevitably contain unwanted technical variations that can obscure meaningful biological signals and affect how this information is applied to personalized healthcare. Although many methods have been developed to handle these variations, the underlying assumptions of existing tools often apply to only a few specific scenarios and are likely not effective for analysis of multi-timepoint data. To address this gap, TIGER (**T**echnical variation el**I**mination with ensemble learnin**G** archit**E**ctu**R**e)–a robust, non-parametric method is developed and released as a user-friendly R package. TIGER integrates the random forest algorithm into an adaptable ensemble learning architecture. Evaluation shows that TIGER outperforms four popular methods in robustness and reliability across three human datasets covering both targeted and untargeted metabolomics data. Additionally, a case study aimed at identifying age-associated metabolites illustrates how TIGER can be used in longitudinal analyses. A dynamic website is also developed to evaluate the performance of TIGER and the patterns revealed in our longitudinal analysis.

LEOPARD: Large longitudinal studies typically involve high-dimensional multi-view datasets, with each view corresponding to data from distinct biochemical classes or technological platforms. However, various factors, such as measurement dropout, inappropriate sample storage, and platform limitations, can lead to incomplete views. This issue restricts data extrapolation, yet to date, well-established solutions tailored to this critical challenge are still lacking. In this thesis, an innovative neural network

architecture, LEOPARD (missing view comp**L**etion for multi-tim**E**point **O**mics data via re**P**resentation disent**A**nglement and tempo**R**al knowle**D**ge transfer), has been specifically developed to complete missing views in multi-timepoint omics data. By disentangling longitudinal omics data into content and temporal representations, LEOPARD transfers the temporal knowledge to the omics-specific content, thereby completing missing views. The effectiveness of LEOPARD is validated through extensive simulations on three benchmark datasets. Compared to four conventional imputation methods, LEOPARD yields the most robust results across the benchmark datasets. LEOPARD-imputed data also achieve the highest agreement with observed data in our case studies for age-associated metabolites detection, estimated glomerular filtration rate-associated proteins identification, and chronic kidney disease prediction. To the best of our knowledge, this work represents the first step toward a generalized approach to addressing missing views in longitudinal omics data. Additionally, the concept of omics data factorization provides a novel perspective for predictive healthcare that extends beyond data imputation.

Together, TIGER and LEOPARD enhance the reliability and reproducibility of complex multi-timepoint omics datasets, enabling researchers to conduct more robust analyses of biological processes and health outcomes.

Zusammenfassung

Fortschritte in Omics-Technologien haben Daten über mehrere Zeitpunkte hinweg bereitgestellt und bieten einzigartige Einblicke in die zeitlichen Dynamiken auf individueller Ebene und geben so beispiellose Möglichkeiten für die personalisierte Medizin. Die Komplexität und die technischen Herausforderungen, die in *Multi-Timepoint-Omics*-Daten (mit mehreren Zeitpunkten) inhärent sind, führen jedoch zu Problemen wie technischen Variationen und fehlenden Ansichten, die die Umsetzung für medizinische Anwendungen erheblich behindern. Die vorliegende Dissertation stellt zwei computergestützte Werkzeuge vor, TIGER und LEOPARD, die entwickelt wurden, um die Qualitätskontrolle in *Multi-Timepoint-Omics*-Daten, insbesondere in der Metabolomik und Proteomik, zu verbessern.

TIGER: Große *Multi-Timepoint*-Metabolomics-Datensätze enthalten zwangsläufig unerwünschte technische Variationen, die relevante biologische Signale verdecken und die Anwendung dieser Informationen in der personalisierten Gesundheitsversorgung beeinträchtigen können. Obwohl viele Methoden entwickelt wurden, um Variationen zu adressieren, gelten die zugrunde liegenden Annahmen bestehender Algorithmen oft nur für einige spezifische Szenarien und sind wenig effektiv für die Analyse von *Multi-Timepoint*-Daten. Um diese Lücke zu schließen, wurde TIGER (***T****echnical variation el****I****mination with ensemble learnin****G*** *archit****E****ctu****R****e*) entwickelt als eine robuste, nichtparametrische Methode und als benutzerfreundliches R-Paket veröffentlicht. TIGER integriert den *Random-Forest*-Algorithmus in eine anpassungsfähige *Ensemble Learning* Architektur. Die Evaluierung zeigt, dass TIGER in Bezug auf Robustheit und Zuverlässigkeit in drei menschlichen Datensätzen, die sowohl gezielte als auch ungezielte Metabolomics-Daten abdecken, vier populäre Methoden übertrifft. Darüber hinaus zeigt eine Fallstudie, die darauf abzielt, alters-assoziierte Metaboliten zu identifizieren, wie TIGER in longitudinalen Analysen verwendet werden kann. Eine dynamische Website wurde entwickelt, um die Leistung von TIGER zu bewerten und die in unserer longitudinalen Analyse aufgedeckten Muster zu identifizieren.

LEOPARD: Große longitudinale Studien umfassen typischerweise hochdimensionale Datensätze mit vielen Ansichten, wobei jede Ansicht verschiedenen biochemischen Klassen oder technologischen Plattformen entspricht. Verschiedene Faktoren wie fehlende Messungen, unsachgemäße Probenlagerung und Plattformbeschränkungen können jedoch zu unvollständigen Ansichten führen. Dieses Problem schränkt die Datenextrapolation ein, und es gibt derzeit keine etablierten Lösungen, um diese kritischen Herausforderungen zu adressieren. Das innovative neurale Netzwerk LEOPARD (*missing view comp**L**etion for multi-tim**E**point **O**mics data via re**P**resentation disent**A**nglement and tempo**R**al knowle**D**ge transfer*) wurde speziell entwickelt, um fehlende Ansichten in *Multi-Timepoint-Omics*-Daten zu vervollständigen. Durch das Entflechten longitudinaler Omics-Daten in Darstellungen von Inhalt und Zeit extrapoliert LEOPARD das die zeitliche Entwicklung auf den Omics-spezifischen Inhalt und vervollständigt so die fehlenden Ansichten. Die Wirksamkeit von LEOPARD wurde durch umfangreiche Simulationen an drei Benchmark-Datensätzen validiert. Im Vergleich zu vier konventionellen Imputationsmethoden liefert LEOPARD die robustesten Ergebnisse über die Benchmark-Datensätze hinweg. LEOPARD-imputierte Daten erreichen auch die höchste Übereinstimmung mit beobachteten Daten in unseren Fallstudien zur Detektion von alters-assoziierten Metaboliten, zur Identifikation von Proteinen, die mit der geschätzten glomerulären Filtrationsrate assoziiert sind, und zur Vorhersage chronischer Nierenerkrankungen. Die Ergebnisse liefern zum ersten Mal einen Schritt zu einem generalisierten Ansatz zur Behandlung von fehlenden Ansichten in longitudinalen Omics-Daten. Darüber hinaus bietet das Konzept der Omics-Daten-Faktorisierung eine neuartige Perspektive für die prädiktive Gesundheitsversorgung, die über die Datenimputation hinausgeht.

Insgesamt verbessern TIGER und LEOPARD die Zuverlässigkeit und Reproduzierbarkeit komplexer *Multi-Timepoint-Omics*-Datensätze und ermöglichen es Forschern, robustere Analysen biologischer Prozesse und gesundheitlicher Ergebnisse durchzuführen.

Acknowledgments

First and foremost, I am profoundly grateful to my direct supervisor, P.D. Dr. Rui Wang-Sattler, for her invaluable guidance and insight. Her great support has been fundamental in shaping this research, and her constant encouragement provided me with the motivation to overcome numerous challenges in my research.

I would also like to extend my sincere thanks to my thesis committee members, Prof. Dr. Eleftheria Zeggini and Prof. Dr. Dmitrij Frishman for their constructive feedback and suggestions. Their input greatly improved the quality of this work.

I would also like to thank my group members, Dr. Marcela Covic, Dr. Jialing Huang, Dr. Makoto Harada, Shixiang Yu, and Jianhong Ge for their collaboration. A special mention goes to Mengya Shi for her invaluable support during my research.

I am thankful to Prof. Dr. Annette Peters and Dr. Na Cai for their valuable support and the discussions that always provided fresh perspectives on my research.

Furthermore, I am thankful to the Institute of Translational Genomics family: Dr. Iris Fischer, Dr. Anne Weyand, Dr. Bahar Sanli-Bonazzi, Dr. Konstantinos Hatzikotoulas, Dr. Will Rayner, Dr. Nancy Yu, Dr. Grace Png, Peter Kreitmaier, Georgia Katsoula, Ana Luiza Arruda, Yue Huang, Zhengyuan Xue, Barbara Puzek, Shibo Chen, Young-Chan Park, and all the others whose names I may not have mentioned, but whose help have been equally important. Their support and companionship have been invaluable throughout my doctoral journey.

I express my appreciation to my collaborators, Dr. Paolo Casale, Dr. Christian Gieger, Prof. Dr. Barbara Thorand, Prof. Dr. Giuseppe Matullo, Prof. Dr. Flora Sam, Prof. Dr. Karsten Suhre, Dr. Ying Li, Prof. Dr. Barbara Thorand, Dr. Cornelia Prehn, Prof. Dr. Jerzy Adamski, Dr. Agnese Petrera, Dr. Stefanie M. Hauck, Prof. Dr. Christian Herder, Prof. Dr. Michael Roden, Dr. Mark Ibberson, Prof. Dr. Geneviève Derumeaux, Dr. Josefina Lascano Maillard, and Olivier Martin for their valuable feedback on my work.

I also thank my friends and peers, Lianyun Huang, Siyue Jia, Simon Wengert, Jonathan Adam, Jiesheng Lin, and Mariana Ponce-de-Leon for their help and support.

I would like to acknowledge all KORA study group members and participants for their long-term commitment to the KORA study. Without their contributions, this work would not have been possible.

I appreciate the funding and resources provided by the Innovative Medicines Initiative 2 Joint Undertaking (JU) under grant agreement No 821508 (CARDIATEAM), which made this research possible. Additionally, I would like to thank the HELENA Conference Travel Grants and the DZD NEXT Conference Travel Grant for their financial support.

I would like to express my deepest gratitude to my family for their support and understanding. Their love and patience have been a constant source of strength. I am deeply grateful for their belief in me, and their sacrifices have made this achievement attainable.

Thank you all.

Contents

List of Figures

List of Tables

Abbreviations

AdaIN	Adaptive Instance Normalization.
AUPRC	Area Under the Precision-Recall Curve.
AUROC	Area Under the Receiver Operating Characteristic curve.
BCE	Binary Cross-Entropy.
BMI	Body Mass Index.
BRF	Balanced Random Forest.
CART	Classification And Regression Trees.
CC	Correlation Coefficient.
cGAN	conditional Generative Adversarial Networks.
CKD	Chronic Kidney Disease.
CRAN	Comprehensive R Archive Network.
CUDA	Compute Unified Device Architecture.
CV	Cross Validation.
eGFR	estimated Glomerular Filtration Rate.
GAN	Generative Adversarial Networks.
GLMM	Generalized Linear Mixed Model.
GPL	GNU General Public Licence.
GPU	Graphics Processing Unit.
HbA1c	Hemoglobin A1c.
ICA	Independent Component Analysis.
IQR	InterQuartile Range.

k-NN	k-Nearest Neighbors.
KORA	Cooperative Health Research in the Region of Augsburg.
LC-MS	Liquid Chromatography-Mass Spectrometry.
LEOPARD	missing view compLetion for multi-timEpoint Omics data via rePresentation disentAnglement and tempoRal knowleDge transfer.
LM	Linear Model.
LOD	Limits of Detection.
LOESS	LOcally Estimated Scatterplot Smoothing, also known as local polynomial regression.
LOOCV	Leave-One-Out Cross Validation.
MAE	Mean Absolute Error.
MAPE	Mean Absolute Percentage Error.
MAR	Missing At Random.
maskObs	percentage of masked data in Observed views.
MCAR	Missing Completely At Random.
MICE	Multivariate Imputation by Chained Equations.
MNAR	Missing Not At Random.
MSE	Mean Squared Error.
NF	Normalization Factor.
NPX	Normalized Protein eXpression.
NT-Xent	Normalized Temperature-scaled Cross-entropy.
obsNum	Number of observed samples in an incomplete view.
OS	Operating System.
PB	Percent Bias.
PBS	Phosphate-Buffered Saline.
PCA	Principal Component Analysis.
PEA	Proximity Extension Assay.

PMM	Predictive Mean Matching.
PReLU	Parametric Rectified Linear Unit.
QC	Quality Control.
ReLU	Rectified Linear Unit.
RF	Random Forest.
RMSE	Root Mean Square Error.
RSD	Relative Standard Deviation, also known as coefficient of variation.
SERRF	Systematic Error Removal using Random Forest.
SVM	Support Vector Machine.
TIGER	Technical variation elImination with ensemble learninG architEctuRe.
UMAP	Uniform Manifold Approximation and Projection.
WT	Wavelet Transform.
XGB	eXtreme Gradient Boosting.

1 Introduction

The expansion of omics technologies has profoundly impacted biomedical research, facilitating the extensive analysis of biological systems at multiple timepoints. This provides great opportunities for advancing personalized medicine but also introduces complex analytical challenges. Ensuring the accuracy and reliability of multi-timepoint omics data analysis is critical for valid scientific insights.

With a focus on improving data reliability and completeness, we introduce two novel computational tools, TIGER (**T**echnical variation el**I**mination with ensemble learnin**G** archit**E**ctu**R**e) and LEOPARD (missing view comp**L**etion for multi-tim**E**point **O**mics data via re**P**resentation disent**A**nglement and tempo**R**al knowle**D**ge transfer), which enhance the quality control processes within multi-timepoint omics studies. These methodologies not only refine data quality but also pave the way for more robust biological research and healthcare applications.

This chapter provides an overview of the primary multi-timepoint omics data utilized in this study, including the specific omics technologies and study design employed. It also details the metrics used to assess the quality and reliability of omics data. Furthermore, this chapter discusses the mechanisms of missing data within the omics datasets. By laying the foundation and introducing fundamental concepts of multi-timepoint omics data Quality Control (QC), this chapter establishes the connection between omics data and the development and evaluation of advanced computational tools that are discussed in subsequent chapters.

1.1 Data from KORA Study

The KORA study plays a crucial part in this thesis. The metabolomics and proteomics data are used for performance evaluation and case studies during the development of TIGER and LEOPARD. In this thesis, the data from the KORA study [1] are extracted from the baseline survey (KORA S4, examined between 1999 and 2001), the first follow-up study (KORA F4, 2006 - 2008), and the second follow-up (KORA FF4, 2013 - 2014).

1.1.1 Metabolomics Data

We use targeted metabolomics data at three timepoints from the KORA cohort: S4, F4, and FF4. The metabolite profiling of KORA S4 (March - April 2011), F4 (August 2008 - March 2009), and FF4 (February - October 2019) serum samples spans more than a decade, during which analytical procedures have been upgraded several times. The samples of KORA F4 were measured with the analytical kit Absolute*IDQ*® p150 (p150, BIOCRATES Life Sciences AG, Innsbruck, Austria), while the samples of KORA S4 and FF4 were quantified with Absolute*IDQ*® p180 (p180). Kits p150 and p180 allow simultaneous quantification of 163 and 188 metabolites, respectively. To evaluate the technical variation introduced by different kits, samples of 288 individuals from the F4 study were remeasured using the p180 kit (September - October 2019). We use KORA F4 (Original) and F4 (Remeasured) to distinguish the two subsets of the KORA F4 dataset.

During the measurement, the Phosphate-Buffered Saline (PBS) samples were allocated to each 96-well kit plate as negative controls for the subsequent QC procedures. Additionally, three different manufacturer-provided QC samples (QC1, QC2, and QC3) were also allocated to each kit plate. For the p180 kit, each plate additionally quantified five identical pooled EDTA-plasma QC samples (Sera Laboratories International Ltd., Hull, United Kingdom) [2], denoted by QC. Manufacturer-provided QC1, QC2, and QC3 varied due to the platform update, but the pooled EDTA-plasma QC remained the same. Although plasma and serum metabolite concentration profiles are different, they are also highly correlated [3]. Additionally,

1.1.2 Proteomics Data

The proteomics data used in this thesis are from two timepoints, S4 and F4. The data are measured using plasma (S4, February 2020) and serum (F4, December 2016 - January 2017) samples with the Olink® Target 96 Inflammation panel (Olink Proteomics, Uppsala, Sweden) [4]. The panel includes 92 protein biomarkers and uses the Proximity Extension Assay (PEA) technology to measure their relative abundance [5].

1.2 Criteria for Quality Control

In proteomics and metabolomics studies, several QC criteria are widely used to examine the reliability and reproducibility of the data. These criteria help identify potential

errors and variations, thus enabling more robust downstream biological analyses. This section details the definitions and calculations of three key QC metrics: detection rate, Limits of Detection (LOD), and Relative Standard Deviation (RSD).

1. **Limits of Detection (LOD)**

   LOD represents the lowest concentration of a substance that can be reliably distinguished from the background noise. LOD can be calculated for the entire dataset or separately for each plate or batch. In our QC process for the metabolomics data, we calculate LOD for each plate using the following formula:

   $$\text{LOD}_{m,p} = 3 \times \text{median}(\text{PBS}_{m,p}), \tag{1.1}$$

   where p denotes a plate and m denotes a specific metabolite. PBS samples represent the background noise level. To compute the percentage of samples above LOD, we use the following formulas:

   $$\text{above\_LOD\_number}_{m,p} = \sum_{i=1}^{n_p} \mathbf{I}(x_{m,p,i} > \text{LOD}_{m,p}), \tag{1.2}$$

   $$\text{above\_LOD\_ratio}_m = \frac{\sum_p \text{above\_LOD\_number}_{m,p}}{N}, \tag{1.3}$$

   where x represents metabolite concentration, i denotes different subject samples, n_p denotes the total number of subject samples in plate p, and $\mathbf{I}$(condition) is an indicator function that is 1 if the condition is true and 0 otherwise. N is the total subject sample number. Typically, a variable will be kept if 50-75% of its subject samples are above LOD.

2. **Detection Rate**

   The detection rate for each variable is calculated using subject samples as the ratio between the number of detected (non-zero, non-missing) values and the total number of samples.

3. **Relative Standard Deviation (RSD)**

   RSD, also known as the coefficient of variation, is a unitless and standardized measure of dispersion of a data distribution. It is defined as the ratio of the standard deviation to the arithmetic mean:

$$\text{RSD} = \frac{\sigma}{\mu}, \tag{1.4}$$

where σ and μ denotes standard deviation and arithmetic mean, respectively. RSD is computed for each metabolite measured from QC samples. As the metabolomics data from the same type of QC samples should theoretically be identical (but in reality have fluctuations due to technical variations), a low RSD indicates low technical variation. A variable will be kept if its RSD is lower than a cutoff, which generally ranges from 15-30%. In addition to the overall RSD, intraRSD and interRSD can also be calculated to assess the variation within and between plates:

- IntraRSD evaluates the variation within a single plate or batch. For each plate or batch, the QC samples are used to measure the RSD for each plate, and then the RSD values are averaged across plates.
- InterRSD evaluates the variation between plates or batches. The mean of the QC samples is first computed across each plate/batch, and then the interRSD is obtained by computing RSD using these means from different plates.

$$\text{interRSD} = \frac{\sigma_{\text{means}}}{\mu_{\text{means}}}, \tag{1.5}$$

where σ_{means} is the standard deviation of the means and μ_{means} is the mean of the means across different plates or batches.

In this thesis, these QC criteria are used to improve the quality of the benchmark datasets. The results of these QC criteria calculated on the targeted metabolomics data and proteomics data from the KORA study can be found in Tables B.1 and B.2, respectively.

1.3 Missing Mechanism

In multi-omics data analysis, missing data is a common challenge due to the complexity and difficulties of data acquisition. Understanding the mechanisms behind missing data is essential for addressing them effectively. The mechanism of missing data can be broadly categorized into three types:

1. **Missing Completely At Random (MCAR)**

   Definition: The likelihood of missing data is independent of observed and unobserved data. The missingness has no relationship to the values of any data.

   Example: Random technical failures during profiling processing that lead to missing data points across the dataset.

2. **Missing At Random (MAR)**

   Definition: The probability of missing data is related to the observed data but not to the unobserved data.

   Example: For a dataset that includes genomics and proteomics data, the missing protein expression levels may be related to some observed genomic variations that influence protein expression.

3. **Missing Not At Random (MNAR)**

   Definition: The missingness is related to the unobserved data itself, meaning that the missing data depends on the missing values.

   Example: In a health survey, individuals with poor health may skip questions related to their health status to avoid disclosure. The missing data on health status is directly related to their health conditions.

Identifying the underlying missing mechanisms can help select suitable imputation methods and analytical techniques to address missing data. However, determining the missingness mechanism in the multi-omics data context is challenging. Existing methods [6, 7, 8] also adopt the strategy of avoiding "featurizing missingness" to mitigate the risk of making invalid assumptions and resulting in biased results.

2 TIGER for Technical Variation Removal

This chapter introduces TIGER, a robust method developed to address technical variation in metabolomics data. We begin by discussing the challenges of technical variations and the limitations of current solutions (Section 2.1). Next, we detail the TIGER methodology and its ensemble learning architecture (Section 2.2). TIGER is then benchmarked against several state-of-the-art tools (Section 2.3) and evaluated in a multi-timepoint scenario (Section 2.4). The discussion (Section 2.5) explores the performance of different methods and highlights issues of overfitting and underfitting in established methods. Finally, we conclude with the reproducibility and availability section (Section 2.6), which provides information on obtaining the ready-to-use TIGER package, along with the relevant data and scripts.

2.1 Overview

Metabolomics provides a unique perspective to quantitatively characterize small molecule ($<$ 1500 Dalton), metabolites, which can represent the metabolic status of a subject. Liquid Chromatography-Mass Spectrometry (LC-MS) is widely used in metabolomics studies for its high sensitivity and extensive metabolite coverage, making it ideal for detecting and quantifying a broad range of metabolites. Metabolomics analyses facilitate the identification of biomarkers and improve the understanding of biological pathways in health and metabolic diseases such as diabetes [9, 10, 11]. With the rapid growth of metabolomics data, techniques from fields such as data modeling and statistical analysis have enabled the effective identification of metabolites that are related to the phenotype of interest [12]. Despite significant advances in recent years, unwanted technical variation still remains a critical issue in the current metabolomics workflow [13] and prevents the translation of metabolomics analyses in personalized healthcare.

Unwanted technical variation can cause detectable differences between samples irrespective of biological variation. These variations greatly affect the accuracy of the

measurements and further lead to false discoveries [14, 15]. The main sources of these variations include, but are not limited to:

- plate effects or batch effects: Due to the large sample size, samples might be processed in different plates or batches rather than all together. Those differences in instrument calibration and operator handling can lead to inconsistencies in the measured data.
- temporal drifts: Analytical signals may change over time during long analytical runs, which causes drifts in the detected metabolite levels.
- analytical platforms: The changes or updates of analytical platforms will yield different metabolite profiles and levels.

In metabolomics experiments, biological variation of interest is inevitably confounded with systematic errors or technical variations. These can be categorized as intra- and inter-batch technical variation [16]. Intra-batch variation generally refers to the incremental changes in instrumental response during the measurement of a batch of samples [17]. And inter-batch variation occurs when samples in a large-scale study have to be separated into batches [18]. Broadly speaking, temporal drifts within a single batch can be regarded as intra-batch variation, while changes in analytical platforms can be considered as measurements taken in two separate batches, thus representing inter-batch variation. These unwanted technical variations are challenging, even impossible, to control during measurement [19, 15]. Therefore, it is critical to ”normalize” or ”adjust” raw metabolomics data to remove these variations for a reliable downstream analysis [20].

2.1.1 State of the Art

Many practical methods and tools (Table 2.1) have been proposed to address technical variations within metabolomics data.

One of the most common approaches is to use Normalization Factor (NF). For each metabolite of each batch, the raw metabolite values of the QC samples in this batch act as test values, while the averaged metabolite value of the QC samples from all batches is used as the target value. The NF is the median [21] or mean [22] of the ratios of the test values relative to the target value. Each metabolite from each batch has its own NF to capture the deviation caused by the potential variations. For subject samples, the deviation can be finally offset by dividing the values of subject samples by the

corresponding NF. In this case, the method can be viewed as fitting a linear equation where the calculated NF acts as the slope term.

In addition to linear regression, Locally Estimated Scatterplot Smoothing (LOESS, also known as Local Polynomial Regression), is also being applied to eliminate technical variation [23, 24]. LOESS offers greater model flexibility compared to the linear equation used by NF. Compared to a straight line determined by NF, LOESS fits a polynomial curve by assigning greater weights to points closer to the one being estimated. The rationale is that QC samples temporally close to a given subject sample can better characterize the temporal drifts it experiences compared to those QC samples that are farther away. A LOESS model for removing technical variation is typically trained using injection order information.

More recently, a machine learning-based method, SERRF (Systematic Error Removal using Random Forest) [25], has been developed to normalize large-scale untargeted metabolomics data. The study of SERRF demonstrates that technical variation in the intensity of one compound can be modeled by the intensities of other compounds. By training an RF [27] model to regress unwanted systematic variation, SERRF has surpassed many popular methods, including NOMIS (Normalization using Optimal selection of Multiple Internal Standards) [28], cubic splines normalization [29], and an SVM-based method [30], on several benchmark datasets.

QC-based methods such as NF, LOESS, and SERRF utilize QC samples to train their models for technical variation removal. In scenarios where QC samples are unavailable, WaveICA [26] is a viable alternative. WaveICA utilizes the Wavelet Transform (WT) [31] and the Independent Component Analysis (ICA) [32, 33] to capture and remove technical variation. Assuming that metabolite intensities exhibit temporal trends over injection

Table 2.1: Methods for Technical Variation Removal

Method	**Algorithm**	**Data Type**	**QC-Based**	**Injection Order**	**Multiple QC**	**Reference**
NF	Linear equation	Various	Yes	No	No	[21, 22]
LOESS	LOESS	Various	Yes	Required	No	[23, 24]
SERRF	RF	Untargeted	Yes	Required	No	[25]
WaveICA	WT & ICA	Untargeted	No	Required	No	[26]
TIGER	Ensemble learning	Various	Yes	Support	Support	This study

order, WaveICA first uses WT to decompose the trend into multi-scale data with different frequencies. Subsequently, ICA is used to detect and remove unwanted variation within the multi-scale data. The normalized data are finally reconstructed using the inverse WT. The experiments show that WaveICA outperforms ComBat (combatting batch effects when Combining Batches) [34], a well-known QC-free method based on empirical Bayes, as well as QC-RLSC (Quality Control-based Robust LOESS Signal Correction) [35].

Although these established methods are valuable and help to remove the technical variations within metabolomics data, they are subject to limitations:

- One underlying assumption of NF is that the fluctuation in one metabolite value can be balanced by the values of other metabolites, but it has been argued that this assumption is only valid in limited practical scenarios [36, 28].
- The normalization capability of LOESS is boosted by its more flexible algorithm, but the model of LOESS neglects the potential associations between metabolites and tends to overfit the data. Regarding data structure, LOESS can hardly be applied to datasets that contain more variables (metabolites) than samples [25].
- SERRF is one of the most robust methods, but its model trained on QC samples cannot guarantee to yield satisfactory results on subject samples–this is also the issue of most QC-based methods.
- WaveICA is not afflicted by the common issues of QC-based methods. However, it assumes that biological variation mainly exists in the data with high frequency, while temporal drifts are in the low-frequency part. This assumption may result in the removal of biological variation present in the low-frequency area.

2.1.2 Current Challenges

Recently, the routine use of well-characterized QC samples has been recognized as an effective strategy to improve the external validity of large-scale metabolomics studies. [20, 37, 38]. QC samples can not only help eliminate temporal drifts, batch effects, and other technical variations but also help achieve better inter-laboratory reproducibility, enabling effective comparison of data from different platforms and centers [39].

With the advancement of computational health and machine learning, an increasing number of QC-based methods, such as SERRF, are now employing machine learning algorithms to remove technical variation. In this context, metabolite values of QC and

subject samples are typically used as training and test data, respectively. A traditional machine learning model is generally trained and fine-tuned on a training set that is highly representative of unseen examples. However, in our case, QC samples may not be fully representative of subject samples. Due to underfitting and overfitting, many QC-based methods only yield weak performance on subject samples [40, 26]. Additionally, the same QC samples are often divided into two subsets for model training and validation, which further makes the evaluation result too optimistic–strong evaluation performance obtained on the validation set (i.e. QC samples) does not guarantee a satisfactory result on the test set (i.e. subject samples).

To improve generalization ability, datasets of many cohorts or studies integrated multiple types of QC samples, sometimes corresponding to different metabolite concentration levels, into the measurements. The availability of multiple types of QC samples provides more flexibility and better reproducibility for metabolomics data pre-processing. However, almost none of the existing methods is able to process the datasets with different kinds of QC samples. With the growth of QC-provided metabolomics datasets, the development of highly flexible and readily adaptable QC-based methods become a necessity.

2.1.3 Contribution

Our contribution toward addressing these challenges can be summarized as follows:

- An ensemble learning architecture is developed to enhance model performance on noisy metabolomics datasets. The architecture is more robust to noisy data and mitigates and risk of overfitting on QC samples used for training.

- Based on our ensemble learning architecture, we developed TIGER, a novel algorithm for eliminating technical variation in metabolomics data. Benchmarked against several widely-used methods, TIGER shows the most robust and reliable performance on both targeted and untargeted metabolomics datasets for eliminating intra- and inter-batch technical variation. TIGER can leverage multiple types of QC samples and demonstrates strong performance on cross-kit adjustment. This feature greatly improves data reproducibility and enables the integration of experimental data from different analytical kits.

- A case study is performed to illustrate how TIGER can be applied to longitudinal analysis. The case study suggests that many important patterns only appear after

data adjustment, which emphasizes the importance of technical variation removal for downstream analysis.

- Comprehensive evaluation indicates that multiple metrics should be considered to assess the performance of technical variation removal, to avoid potential under-correction and over-correction issues.

- TIGER is released as a user-friendly R package, available on Comprehensive R Archive Network (CRAN). It can be easily installed in R using the command `install.packages("TIGERr")`. A detailed manual is provided to assist users in applying TIGER to various scenarios. Additionally, a dynamic website (`https://han-siyu.github.io/TIGER_web/`) has been developed, allowing users to interactively compare TIGER with other popular methods and review patterns revealed in our longitudinal analysis.

2.2 Methods

TIGER eliminates the technical variation using an adaptable ensemble learning architecture. This section describes the framework of this architecture and further illustrates how it can be adapted to build TIGER.

2.2.1 Benchmark Datasets Construction

Data from the KORA study [1], the P20 Negative dataset (negative mode, from the Functional Cardio-Metabolomics study), [25] and the Amide dataset of WaveICA [26] are used are benchmark datasets to evaluate the performance of technical variation removal (Table 2.2). The data in the KORA study are the concentrations of targeted metabolites, while the data in the P20 Negative and the Amide measure the compounds or peak intensities of untargeted metabolomics data. Fig. 2.1 illustrates the technical variation of one variation selected from each benchmark dataset. The KORA FF4 dataset contains intra-batch plate effects, while the P20 Negative and Amide datasets exhibit inter-batch technical variation. Additionally, the Amide dataset also shows very strong temporal drifts where the variable intensity decreases as the injection order. Overall, the KORA FF4 has the lowest technical variation, while the Amide dataset shows the highest fluctuation. These three datasets exemplify different types of technical variation and demonstrate the variation at different levels of severity.

Table 2.2: Summary of Benchmark Datasets for Technical Variation Removal

Dataset	Targeted				Untargeted	
	S4	F4 (Original)	F4 (Remeasured)	FF4	P20 Negative	Amide
Batch Count[1]	22	38	4	29	4	4
Variable Count[2]	103	103	103	103	268	6402
Subject Count	1614	3061	288	2218	1174	644
QC Count						
QC	114	NA	22	145	125	85
QC1	22	38	4	29	NA	NA
QC2	22	38	4	29	NA	NA
QC3	22	38	4	29	NA	NA
Median RSD						
QC	0.092	NA	0.088	0.118	0.275	0.514
QC1	0.095	0.123	0.077	0.109	NA	NA
QC2	0.080	0.125	0.068	0.107	NA	NA
QC3	0.080	0.107	0.072	0.109	NA	NA
Median MAPE						
QC	0.074	NA	0.070	0.095	0.226	0.417
QC1	0.073	0.099	0.054	0.087	NA	NA
QC2	0.060	0.098	0.048	0.082	NA	NA
QC3	0.063	0.088	0.055	0.088	NA	NA

[1]Plates for KORA S4, F4, and FF4. Batches for P20 Negative and Amide.
[2]KORA data include 103 metabolites. P20 Negative contains 268 lipids. Amide has 6,402 peaks.

2.2.1.1 Targeted Metabolomics Datasets

Data from KORA FF4 is used for method evaluation where QC are selected as training samples, while QC1, QC2, QC3, and subject samples are used as test data (see Fig. 2.1 for an example variable from KORA FF4). In our case study, datasets of three timepoints, namely KORA S4, F4, and FF4, are used for longitudinal analysis. See the chapter Introduction for a detailed data description of the KORA cohort.

Quality inspection [41, 42] is applied to the metabolomics data. Only the metabolites that are shared by KORA S4, F4, and FF4 and satisfy all the following criteria will be used in this study (Table B.1):

- Metabolites are available for both the p150 and p180 kits
- At least 50% of the measured sample values are above the LOD of its corresponding plates

- The percentage of missing values is lower than 10%
- The median RSD of different QC samples is lower than 25%
- The spearman CCs between the KORA F4 (Remeasured) and F4 (Original) are above 0.5

Finally, 103 targeted metabolites are selected to construct the target metabolomics datasets.

2.2.1.2 Untargeted Metabolomics Datasets

The P20 Negative and Amide are two ready-to-use datasets provided by SERRF and WaveICA. Relevant quality inspection has been conducted in their original studies. P20 Negative dataset is based on plasma samples and acquired using a validated lipidomics assay [25]. In P20 Negative, 1,174 subject samples and 125 identical QC samples were separated into four batches (Fig. 2.1). Each sample has 268 lipids (Table 2.2).

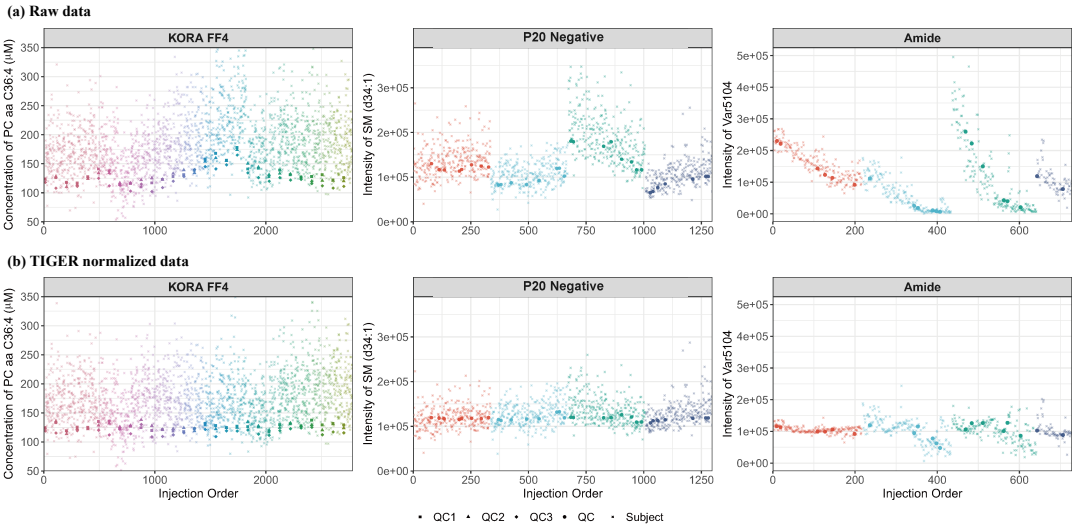

Figure 2.1: Illustration of Technical Variation. (**a**) Raw data of selected metabolites from datasets KORA FF4, P20 Negative, and Amide. Solid dots in each dataset represent QC samples which should be identical. The fluctuations in the QC samples are caused by technical variations. Semi-transparent dots, which are subject samples, generally suffer from similar technical variations as the QC samples. (**b**) Metabolite values after being normalized by TIGER. The x-axis indicates the injection order, and the y-axis represents metabolite concentration (μM) or intensity. Samples from different batches are distinguished by colors.

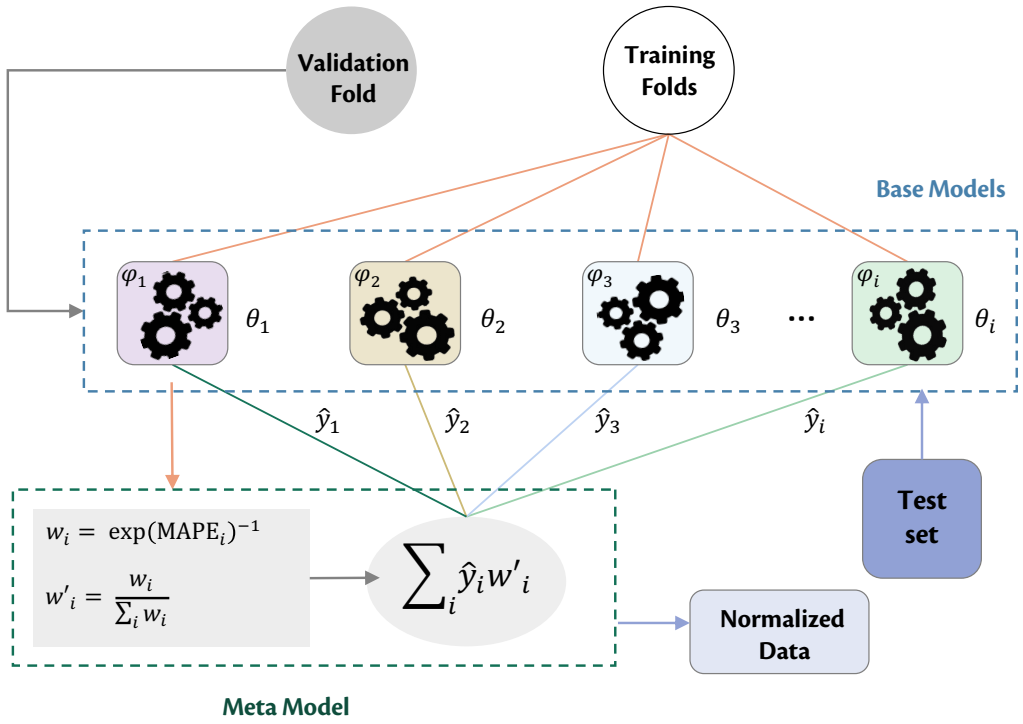

Figure 2.2: Overview of the Ensemble Learning Architecture. The architecture includes several base models and one meta model. The base models are trained with different hyperparameter sets, while the meta model assigns weights to base models such that high-performing base models can obtain great weights.

Dataset Amide is based on plasma samples processed with a UHPLC-QTOF/MS system [26]. 644 subject samples and 85 identical QC samples were collected from four batches, and each sample has 6402 detected peaks (Fig. 2.1 and Table 2.2).

The P20 Negative and Amide datasets only provide one kind of QC sample, 80% of which will be used as training sets, while the remaining QC samples and all subject samples are used as test sets.

2.2.2 Ensemble Learning Architecture

The ensemble learning architecture designed in this study is inspired by the idea of super learner [43] and tailored for the cases where training samples are limited and cannot fully represent test samples. The architecture is comprised of several base models and one meta model. The base models are trained with different hyperparameter sets, which could mitigate the potential overfitting issue. The meta model assigns weights to base

models such that high-performing base models can obtain great weights, but information from underperforming learners can be considered as well (Figure. 2.2).

2.2.2.1 Base Model

The ensemble model has n base models. Each base model is determined by θ_i, a hyperparameter set from pool $\{\theta_1, \theta_2, ..., \theta_n\}$. The base model $\phi_i(\cdot)$ is a machine learning model of the form:

$$\phi_i(y \sim \mathcal{X}) = \varphi(y \sim \mathcal{X}|\theta_i). \tag{2.1}$$

where $\mathcal{X}$ and y represent variable and response; $\varphi(\cdot)$ denotes a function that can be generalized to various approximators, such as SVM, k-Nearest Neighbors (k-NN), or user-defined functions.

When fitting base models, a specific training set, $\mathcal{D} = \{y \sim X|m \in 1 : M, d \in 1 : D\}$ including M instances and D variables, will be shuffled randomly and split into K folds for Cross Validation (CV). For any fold $k \in \{1, 2, ..., K\}$, base models are trained with $K - 1$ training folds $\{y_{-k} \sim X_{-k}\}$ and evaluated on the remaining validation fold X_k:

$$\hat{y}_{i,k} = \phi_i(X_k|y_{-k} \sim X_{-k}), \tag{2.2}$$

where $\hat{y}_{i,k}$ is the predicted result of the validation fold $\{X_k\}$ produced by $\phi_i(\cdot|y_{-k} \sim X_{-k})$, the base model ϕ_i trained with the training folds $\{y_{-k} \sim X_{-k}\}$. The prediction results obtained on the validation folds are collected and concatenated for training the meta model in the next stage:

$$\hat{y}_i = \{\hat{y}_{i,1}, \hat{y}_{i,2}, ..., \hat{y}_{i,k}\}. \tag{2.3}$$

Finally, all base models are re-trained with the whole training set to make full use of all available information.

2.2.2.2 Meta Model

Meta model $\Phi(\cdot)$ corresponds to a model of the form:

$$\Phi(y \sim \mathcal{X}) = \sum_{i}^{n} w_i \phi_i(y \sim \mathcal{X}). \tag{2.4}$$

Unlike many blending models that directly average the predictions of all ensemble members, the architecture here assigns each base model a weight w_i which is transformed from the loss of each validation set via a softmax-like formula:

$$w_i = \frac{\exp(-\ell_i)}{\sum_i^n \exp(-\ell_i)}, \tag{2.5}$$

where ℓ_i is the loss of the corresponding base model $\phi_i(\cdot)$, defined as follows:

$$\ell_i = \mathcal{L}(\hat{y}_i, y), \tag{2.6}$$

$$\mathcal{L}(\hat{y}, y) = \frac{1}{M} \sum_{m=1}^{M} \frac{|\hat{y}^{(m)} - y^{(m)}|}{y^{(m)}}. \tag{2.7}$$

$\mathcal{L}(\cdot)$ is the loss function that quantifies the Mean Absolute Percentage Error (MAPE); $\hat{y}_i^{(m)}$ denotes the m-th values predicted by $\phi_i(\cdot)$ in CV, and $y^{(m)}$ is the corresponding target value.

The numerator of Eq. 2.5 calculates the exponential of the negative loss of each individual base model, and the denominator sums over the exponentials. This ensures base models with small losses have more weights, but learners with large losses can also contribute to the prediction. This alleviates the overfitting problem caused by specific hyperparameter sets, and the resulting ensemble model can outperform the constituent models.

2.2.3 TIGER Algorithm

TIGER iterates many times during the correction (Alg. 1). Only one metabolite will be processed in one iteration. In each iteration, TIGER constructs ensemble models separately for different batches. The model construction can be outlined in three steps: variable selection, model construction, and data correction.

2.2.3.1 Variable Selection

The Correlation Coefficient (CC) values between any two metabolites are calculated separately for training (i.e. QC samples in a general case) and test samples (i.e. subject samples). For a metabolite to be processed, other metabolites with CC values greater than 0.5 in both training and test samples are selected. Then, the intersection of the selected metabolites determines t highly correlated metabolites.

Alg. 1: Overview of TIGER Algorithm

Input: $Q_{D,B}$: QC samples with D metabolites from B batches
Input: $S_{D,B}$: subject samples to be normalized
Output: $S'_{D,B}$: normalized data of subject samples

▷ start iteration of normalization process
for $d \in D$ **do**
  ▷ d: index of the metabolite to be processed
  ▷ step 1: variable selection
  v_Q =highlyCorrelatedMetabolites($Q_{d,B}$)
  v_S =highlyCorrelatedMetabolites($S_{d,B}$)
  $v = v_Q \cap v_S$ ▷ intersection of v_Q and v_S
  ▷ v: selected variables for metabolite d
  for $b \in B$ **do**
    ▷ b: index of batch
    ▷ step 2: model construction
    training set $\mathcal{D}_{d,b} = \{Q_{d,b}, Q_{v,b}\}$
    ▷ $Q_{d,b}$: data of objective metabolite
    ▷ $Q_{v,b}$: data of selected metabolites
    build ensemble model $\Phi_{d,b}(\cdot)$ with $\mathcal{D}_{d,b}$ ▷ to Alg. 2
    ▷ step 3: data correction
    apply $\Phi_{d,b}(\cdot)$ to $S_{d,b}$
  concatenate results
output normalized dataset $S'_{D,B}$

To ensure stable and consistent performance across different iterations, the number of highly correlated metabolites is limited to a specific range $[t_{min}, t_{max}]$. If $t < t_{min}$, the top t_{min} metabolites with the highest CC values in both training and test samples will be selected. If $t > t_{max}$, only the top t_{max} metabolites will be selected.

TIGER supports Pearson product-moment correlation and Spearman's rank correlation to compute CC values. Additionally, partial correlation [44] is supported for variable selection, such that pairwise CC values are conditioned on the correlation with all other metabolites, and indirect associations between distantly related metabolites are ignored [45]. By default, TIGER uses rank-based CC.

The training set is constructed with the data of highly correlated metabolites. This design not only reduces the model complexity but also accounts for potential interactions between metabolite values. Furthermore, the variation introduced by temporal drifts, well position effects, and other unobserved noise are reflected in the correlations. As a result, the model can capture relevant information even if details such as injection order

Alg. 2: Procedures of training ensemble model in TIGER

Input: $\mathcal{D} = \{y, X\}$: training set with response y and variables X
Input: $\{\theta_1, \theta_2, ..., \theta_n\}$: hyperparameter sets for random forest
Input: K: number of folds for cross-validation
Output: normalized metabolomics data

▷ `start to build base models`

compute error ratio y' by Eq. 2.9
$\mathcal{D}_{RF} = \{y', X\}$
shuffle $\mathcal{D}_{RF}$ and split the dataset into K folds
for $i \in \{1, 2, ..., n\}$ **do**
  for $k \in \{1, 2, ..., K\}$ **do**
    train $\psi(y'_{-k} \sim X_{-k}|\theta_i)$ using the training folds
    predict $\hat{y}'_{i,k} = \psi(X_k|y'_{-k} \sim X_{-k})$ ▷ `error ratio`
    transform $\hat{y}_{i,k} = y_k/(\hat{y}'_{i,k} + 1)$ ▷ `concentration`
  get $\hat{y}_i$ by Eq. 2.3
  train $\psi(y' \sim X|\theta_i)$ using the whole training set
  get base model $\phi_i(y \sim X)$ by Eq. 3.7 and Eq. 2.8

▷ `start to fit meta model`

for $i \in \{1, 2, ..., n\}$ **do**
  compute loss $\ell'_i = \mathcal{L}(\hat{y}_i, \bar{y})$ by Eq. 2.7
for $i \in \{1, 2, ..., n\}$ **do**
  compute weight w_i by Eq. 2.5
fit meta model $\Phi(y \sim X) = \sum_i^n w_i \phi_i(y \sim X)$

and well position are not available. Users can still choose to explicitly include injection order and well position in the training data to train TIGER, though it might not be necessary.

2.2.3.2 Model Construction

The ensemble learning architecture designed in this study is a general framework that can be extended to various use cases. In the implementation of TIGER, the architecture is further tailored to suit the task of technical variation removal. Alg. 2 shows how the model is constructed in TIGER using our ensemble learning architecture.

Specifically, the base model $\varphi(\cdot)$ in TIGER is defined as:

$$\varphi(\boldsymbol{y} \sim \boldsymbol{X}|\theta_i) = \frac{\boldsymbol{y}}{\psi(\boldsymbol{y}' \sim \boldsymbol{X}|\theta_i) + 1} \tag{2.8}$$

where $\psi(\cdot)$ is an RF model trained with θ_i, one hyperparameter combination from the hyperparameter pool. By default, the pool contains {`mtry_percent = {0.2, 0.4, 0.6, 0.8}`, `nodesize_percent = {0.2, 0.4, 0.6, 0.8}`}. Users can include more hyperparameters into the pool. Response y in Eq. 2.8 denotes the raw values of a metabolite to be processed, while variable $\mathcal{X}$ denotes the raw values of y's highly correlated metabolites. The RF model $\psi(\cdot)$ predicts y', the error ratio of y, defined as:

$$y' = \frac{y - \bar{y}}{\bar{y}} \tag{2.9}$$

where $\bar{y}$ is the mean or median of y. When fitting the meta model, $\bar{y}$ is used as the target value to compute the loss in Eq. 2.6. If a training set has more than one kind of QC sample, then $\bar{y}$ and y' will be calculated separately for each kind.

In the R package of TIGER, $\bar{y}$ can be configured to be calculated for the whole dataset or each batch. By default, TIGER computes mean-based $\bar{y}$ from the whole dataset ($\bar{y} = \bar{y}_{all}$). In batch-specific case ($\bar{y} = \bar{y}_{batch}$), the obtained metabolite value will be additionally multiplied by a factor, $\bar{y}_{all}/\bar{y}_{batch}$, to offset inter-batch variations. This will make the algorithm more aggressive and competent to process the datasets with strong batch effects.

Overall, for a specific training set $\mathcal{D} = \{y \sim X\}$, TIGER first transforms y, the raw metabolite values of a metabolite to be processed, into the corresponding error ratio y' (Eq. 2.9). Then y' and X are fed into an RF model $\psi(\cdot)$. Accordingly, the predicted result of $\psi(\cdot)$ is also an error ratio denoted by $\hat{y}'$. The predicted error ratio is then converted back to metabolite values by equation $\hat{y}_i = y/(\hat{y}' + 1)$. Subsequently, the output of TIGER's base models $\hat{y}_i$ and $\bar{y}$ are passed to the meta model, and the loss is obtained by Eq. 2.6. Thus, the weight assigned to each base model is based on how close the predicted metabolite value $\hat{y}_i$ is to the target value $\bar{y}$.

2.2.3.3 Data Correction

Given the cross-validated weights, base models can be combined together for data correction. The outcomes are the weighted sums of the predicted results of all base models.

In the implementation of TIGER, test samples are scheduled to be processed on the fly during the model construction, rather than being corrected in a separate step, to avoid redundant computational costs. Parallel computing is supported to accelerate computational speed.

2.2.4 Evaluation Metrics

RSD is one of the most widely used metrics to evaluate data variation. As a low RSD indicates low technical variation, the RSD should decrease after processing the dataset with technical variation removal tools.

In this study, we also include the MAPE as an evaluation metric. MAPE is one of the most common measures in machine learning to evaluate the difference ratio between true values and predicted values (Eq. 2.7). MAPE is also computed for each metabolite measured from QC samples. A lower MAPE indicates the predicted value is closer to its target value, which is defined as the average of the corresponding metabolite values. The MAPE generally decreases after processing the dataset with technical variation removal tools.

Considering that some datasets only provide one kind of QC, while training and testing using the same kind of QC may lead to an over-optimistic evaluation, we further use Principal Component Analysis (PCA) plots to compare the clustering of both QC and subject samples before and after applying technical variation removal tools. Successful removal of technical variations will result in samples, through may be from different batches, clustering together in the PCA plot.

2.3 Performance Evaluation

In this section, we first introduce several established metrics that can be used for evaluating the performance of technical variation removal. Then, we assess how our ensemble learning architecture performs in the task of technical variation removal. Finally, we benchmark TIGER against four popular methods–NF, LOESS, SERRF, and WaveICA–on three benchmark datasets: KORA FF4, P20 Negative, and Amide.

The targeted metabolomics data of the KORA FF4 dataset, from a single batch but spread across 29 plates, are used to assess the removal of intra-batch technical variation. The untargeted metabolomics data of the P20 Negative and Amide datasets, which include data from four batches, are used to evaluate the removal of inter-batch technical variation.

2.3.1 Evaluation of the Ensemble Learning Architecture

Three machine learning algorithms with different complexities, namely k-NN [46], random forest [47] and extreme gradient boosting (XGB) [48, 49], are selected as base

Table 2.3: Hyperparameter Tuning for Non-Ensemble Models

Model	Hyperparameter	Value
k-NN	`k`	[1, 2, 3, **4**]
	`algorithm`	**kd_tree**
RF	`ntree`	[100, 300, **500**]
	`mtry` in ratio	[0.1, 0.2, **0.4**, 0.6, 0.8, 1]
	`samplesize`	[1, 2, 3, **4**]
XGB	`booster`	**gbtree**
	`objective`	**reg:squared error**
	`eval_metric`	**rmse**
	`eta`	**0.01**
	`max_depth`	[**3**, 6, 9]
	`min_child_weight`	[**1**, 2, 3, 4]
	`subsample`	[**0.6**, 0.8, 1]
	`colsample_bytree`	[0.6, 0.8, **1**]
	`early_stopping_rounds`	**50**

Values determined by 5-fold CV and grid search are shown in **bold**.

models to validate the ensemble learning architecture. We compare the performance of each machine learning algorithm under ensemble and non-ensemble settings. The non-ensemble models are fine-tuned using grid search with a 5-fold CV (Table 2.3). Ensemble models are trained using the same hyperparameters as the corresponding non-ensemble models. All models are trained with QC and tested with QC1, QC2, QC3, and subject samples from the KORA FF4 dataset. The KORA FF4 dataset includes four types of QC samples, allowing for training and testing with different QC sample types, which can provide more reliable results.

As shown in Fig. 2.3, all ensemble models show some improvements upon their non-ensemble predecessors, which demonstrates the effectiveness of our ensemble learning architecture. k-NN and XGB give the best and the worst results among non-ensemble models. After adopting the ensemble learning strategy, these two algorithms yielded the least and the greatest improvement. Model complexity may account for this fact: models trained with high complexity algorithms, such as XGB, can capture not only patterns of technical variation but also random noises in the training set, and thus may not generalize well on the unseen examples, which greatly differs from the training data. A model with low complexity, such as k-NN, is less prone to overfit the training set but has limited flexibility to improve accuracy.

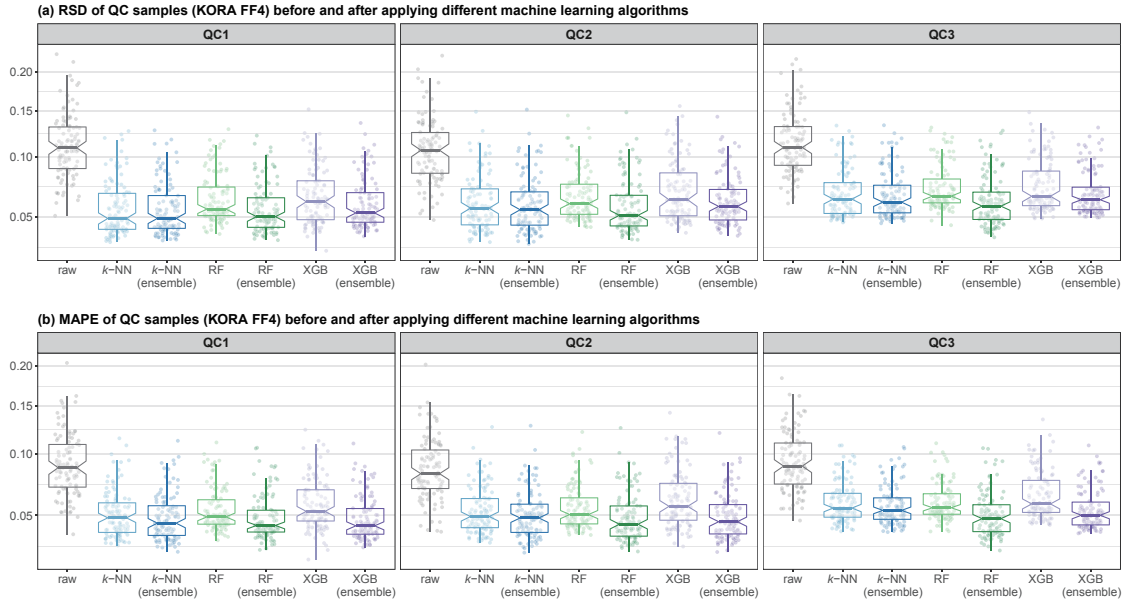

Figure 2.3: Performance Evaluation of the Ensemble Learning Architecture. The performance of k-NN, RF, and XGB are evaluated under both non-ensemble and ensemble scenarios. Each dot represents the RSD (**a**) or MAPE (**b**) value of one metabolite. The results show that ensemble models improve performance over their non-ensemble predecessors. The y-axis has been sqrt-transformed.

Algorithm RF is selected to build the base models in TIGER as it helps to achieve the best performance on both RSD and MAPE with moderate complexity. After incorporating the RF algorithm into the ensemble learning architecture, the medians of RSD and MAPE are 14.43% and 14.79% better than the original model.

To achieve a good balance between model performance and computational complexity, we further evaluate different values for t_{min} and t_{max} during the variable selection phase. The performance is evaluated using parallel computing with 8 cores on a machine equipped with an Intel® Core™ i9-10885 processor, 32 GB of memory, and a 64-bit Windows 10 Operating System (OS). As shown in Table 2.4, in the most extreme scenario, TIGER is able to perform normalization with only one available variable and obtain a median of RSD of 0.0570 (QC2) and a median of MAPE of 0.0581 (QC2). Based on the evaluation result, t_{min} and t_{max} are set as five and ten for TIGER's default setting.

In our evaluation, only one type of QC sample (pooled EDTA-plasma, denoted as QC) was used for training, but we expect the performance can be further boosted if the training set includes additional types of QC samples. This will help the meta model compute more reliable weights to ensure a better generalization ability.

Table 2.4: Evaluation of Different Numbers of Highly Correlated Variables

t_{min}	t_{max}	Time (seconds)	Median of RSD			Median of MAPE		
			QC1	QC2	QC3	QC1	QC2	QC3
1	1	54.81	0.055	0.057	0.062	0.055	0.058	0.062
1	5	114.50	0.052	0.054	0.059	0.048	0.050	0.053
5	5	126.11	0.051	0.053	0.058	0.047	0.050	0.051
5	**10**	148.85	0.050	0.051	0.057	0.044	0.045	0.048
10	10	162.39	0.052	0.052	0.059	0.046	0.045	0.049
15	20	208.05	0.051	0.051	0.056	0.044	0.044	0.047
20	20	237.53	0.051	0.052	0.056	0.043	0.043	0.046
25	30	288.32	0.051	0.051	0.057	0.043	0.042	0.046
30	30	301.83	0.051	0.052	0.056	0.043	0.043	0.046
35	40	336.38	0.051	0.052	0.057	0.042	0.043	0.046
40	40	368.18	0.050	0.052	0.057	0.042	0.043	0.045
45	50	402.18	0.051	0.051	0.057	0.042	0.042	0.045
50	50	430.69	0.050	0.051	0.057	0.043	0.042	0.045
55	60	463.64	0.051	0.051	0.057	0.042	0.042	0.045
60	60	497.64	0.051	0.051	0.056	0.042	0.042	0.045
65	70	531.70	0.051	0.051	0.056	0.042	0.042	0.045
70	70	568.68	0.051	0.052	0.056	0.042	0.042	0.045
75	80	596.33	0.051	0.052	0.056	0.041	0.042	0.045
80	80	632.97	0.051	0.052	0.057	0.041	0.042	0.045
85	90	670.61	0.051	0.052	0.057	0.041	0.042	0.045
90	90	701.89	0.051	0.052	0.057	0.041	0.042	0.045
95	all	745.46	0.051	0.053	0.057	0.041	0.042	0.045
all	all	787.71	0.051	0.053	0.057	0.041	0.042	0.046

The selected hyperparameters are indicated in **bold**.

2.3.2 Evaluation on the Targeted Metabolomics Dataset

We first evaluated different methods on the targeted metabolomics data from the KORA FF4 dataset. SERRF and WaveICA are developed for untargeted metabolomics data. But our evaluation showed that they may be applicable to targeted metabolomics data as well.

Different methods yield similar RSD and MAPE values on the three manufacturer-provided QC samples (Table 2.5). All methods are able to lower the RSD, which means the normalized values are less dispersed than the raw values. The low medians of MAPE of the data processed by TIGER, NF, and SERRF also suggest good performances of these methods. However, the metabolite values seem to further deviate from the target

Table 2.5: Technical Variation Removal: Targeted Metabolomics Data

Method	Median of RSD			Median of MAPE		
	QC1	QC2	QC3	QC1	QC2	QC3
raw	0.109	0.107	0.109	0.087	0.082	0.088
TIGER	**0.050**	**0.051**	**0.057**	**0.044**	**0.045**	**0.048**
NF	0.057	0.060	0.066	0.046	0.048	0.054
LOESS	0.063	0.069	0.069	0.544	0.544	0.543
SERRF	0.094	0.098	0.099	0.074	0.076	0.076
WaveICA	0.078	0.083	0.086	0.253	0.254	0.274

Performance is evaluated on the KORA FF4 dataset.
Bold number indicates the best result.

values after being processed by LOESS and WaveICA–the median of MAPE (QC2) increases from 0.082 (raw) to 0.544 and 0.254, respectively (Fig. 2.4).

The cumulative curves (Fig. 2.5) show how the variable percentage changes with the increase of RSD or MAPE. TIGER has the fastest growing curves with the increase of RSD and MAPE. Around 93% of the metabolites have RSD values lower than 0.1 after being processed by TIGER, compared with 41% for raw data and 85%, 85%, 52%, and 70% for data processed by LOESS, NF, SERRF, and WaveICA, at the same cut-off. This indicates the dataset processed by TIGER has less variation and is of better quality than the other four methods. TIGER also greatly lowers the error ratio–99% of metabolites have MAPE values lower than 10%, better than its two best counterparts (NF and SERRF) by factors of 4% and 8%.

Overall, TIGER achieves the best performance among all investigated methods and strikes a superior balance between RSD and MAPE. The evaluation results show that TIGER outperforms the other four methods and effectively eliminates the intra-batch technical variation. Fig. 2.1 illustrates the normalized results of one metabolite after TIGER's normalization. A detailed metabolite-level comparison between TIGER and its benchmarking partners is available on our dynamic website (`https://han-siyu.github.io/TIGER_web/`).

2.3.3 Evaluation on Untargeted Metabolomics Datasets

Two untargeted metabolomics datasets, P20 Negative and Amide, are further used for performance evaluation. As the Amide dataset contains stronger technical variation (

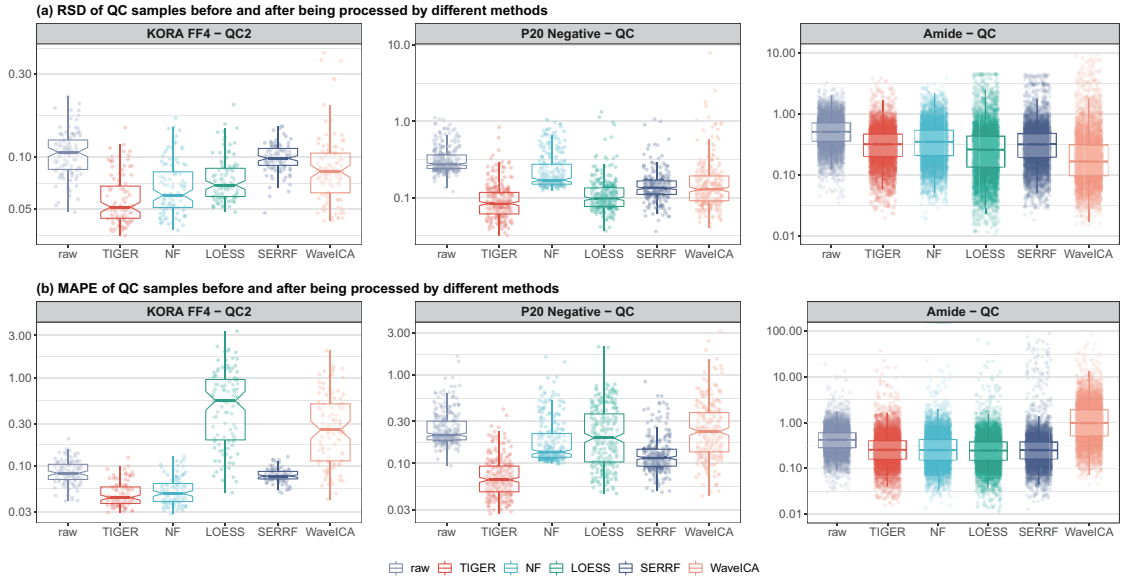

Figure 2.4: Performance of Technical Variation Removal: RSD and MAPE. RSD (**a**) and MAPE (**b**) values are calculated on the QC samples from datasets KORA FF4, P20 Negative, and Amide. Each dot represents the corresponding metric value of one metabolite or variable, while the box shows the overall distribution. The performance in this figure is obtained by TIGER's ready-to-use R package, and the performance here slightly differs from the result in Fig. 2.3 which is obtained from the fine-tuned model. The y-axis has been $\log_{10}$-transformed.

Table 2.6) than the KORA FF4 and P20 Negative datasets, the target value (in Eq. 2.7), an argument of TIGER, is configured to compute based on each batch.

Fig. 2.4 and Table 2.6 show that TIGER effectively lowers the technical variations and improves the data quality of the P20 Negative dataset: the median of RSD is reduced by 19.06% (from 0.274 of raw data to 0.084), while the median of MAPE is reduced by 14.67% (from 0.212 of raw data to 0.065). From the cumulative curves (Fig. 2.5), we notice that only 3.73% of the metabolites in the raw test set have RSD values ranging from 0 to 0.2, while the percentage increases to 91.42% after being processed by TIGER. TIGER also outperforms the other methods with 96.27% of metabolites achieving MAPE lower than 0.2, compared with 51.49%, 70.15%, 86.57%, and 43.66% for LOESS, NF, SERRF, and WaveICA, respectively.

The evaluation on the Amide dataset shows that all methods manage to lower the RSD of QC samples (Fig. 2.4 and Table 2.6), while WaveICA has the lowest median of RSD. In the cumulative curves (Fig. 2.5), WaveICA outperforms its counterparts by achieving the result that 76.27% of the metabolites achieve achieve RSD lower than 0.3.

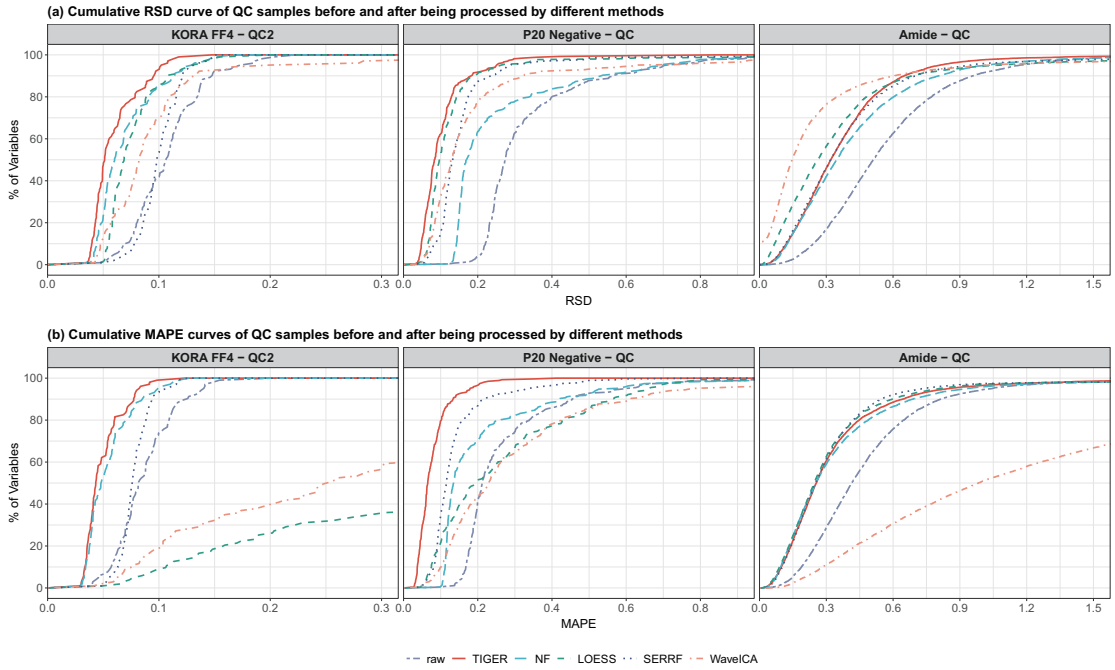

Figure 2.5: Performance of Technical Variation Removal: Cumulative RSD and MAPE. Cumulative RSD (**a**) and MAPE (**b**) curves are calculated on QC samples from the KORA FF4, P20 Negative, and Amide datasets. The x-axis shows different metrics, and the y-axis is the percentage of variables. The curve captures the overall performance of a method, which indicates how effectively the batch effects are eliminated within certain ranges of metric values or the percentage of variables. Some values in Amide unexpectedly become negative after being processed by WaveICA, thus the RSD curve of WaveICA does not start from zero.

By contrast, around 50.48%, 45.95%, 45.94%, and 41.95% of the metabolites yield RSD below the same threshold, after being processed by LOESS, TIGER, SERRF, and NF, respectively. However, we notice that the WaveICA-normalized data contain counter-intuitive negative values, causing its cumulative RSD curve not to start from the origin (0.0%). In terms of MAPE, the evaluation reveals similar good performance among TIGER, LOESS, NF, and SERRF, while WaveICA gets the highest median of MAPE. This result is also reflected in the corresponding cumulative curves.

Except for WaveICA, all other methods are trained and evaluated using the same QC samples, which could lead to a too-optimistic evaluation. To have a reliable assessment, PCA analysis is further performed to evaluate how each method generalizes to the subject samples. Fig. 2.6 shows four clusters, representing four batches of the raw data of the P20 Negative and Amide datasets, respectively. Similar samples in the PCA plot

Table 2.6: Technical Variation Removal: Untargeted Metabolomics Data

Method	Median of RSD		Median of MAPE	
	P20 Negative	Amide	P20 Negative	Amide
raw	0.274	0.511	0.212	0.415
TIGER	**0.084**	0.321	**0.065**	0.255
NF	0.170	0.347	0.135	0.252
LOESS	0.097	0.261	0.195	**0.246**
SERRF	0.133	0.320	0.116	0.251
WaveICA	0.128	**0.146**	0.230	0.983

Performance is evaluated on the P20 Negative and Amide datasets.
Bold number indicates the best result.

should be clustered together. Therefore, no clear clusters of different batches should be observed after applying normalization methods. After being normalized by TIGER, SERRF, and WaveIC, the QC and subject samples in the P20 Negative and Amide datasets are clustered together without distinct batch differences, which demonstrates that these methods effectively eliminate the technical variations within the data. By contrast, after normalized by NF, QC and subject samples still contain strong batch effects, which means NF underfits the data.

From the RSD and MAPE results in Fig. 2.4 and Table 2.6, it seems that LOESS achieves the best performance on the Amide dataset. However, the PCA plots shows that the batch effect can still be detected in its subject samples, while QC samples cluster tightly. This suggests that LOESS suffers from overfitting with a favorable result on QC samples but a substandard performance on subject samples. Based on our evaluation, TIGER achieves the most compact clusters for both QC and subject samples, proving that TIGER's capability for technical variation removal. Fig. 2.1 shows the normalized results of certain metabolites after applying TIGER.

2.3.4 Computational Speed Analysis

As one dataset generally has only a small number of QC samples for model training, the increase of samples may not make a big impact on the complexity of TIGER. Additionally, by default, TIGER selects between 5 and 10 highly correlated variables to train each model; therefore, an increase of variables does not greatly raise the complexity of each ensemble model. However, the running time will increase with the number of variables and batches, as TIGER builds different ensemble models for different metabolites

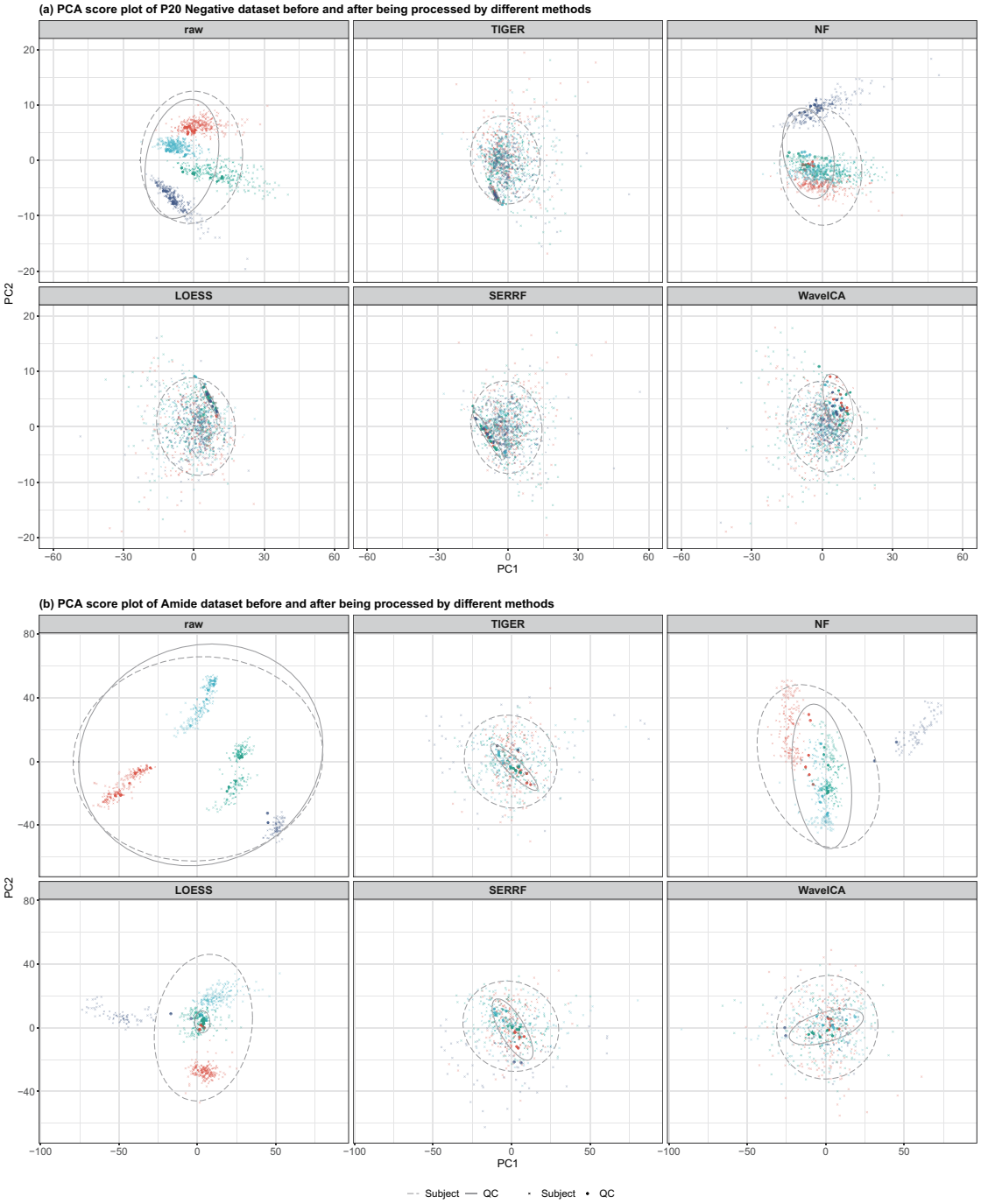

Figure 2.6: Performance of Technical Variation Removal: PCA. QC samples and subject samples are represented by solid points and semi-transparent dots. Samples from different batches, marked with different colors, are expected to mix together after normalization. (**a**) For P20 Negative dataset, the dataset processed by NF has evident batch effects in both QC and subject samples, which suggests that NF underfits the data. (**b**) For the Amide dataset, LOESS overfits the data. The QC samples cluster in one group after LOESS's normalization, but subject samples from different batches are still separated.

and batches. For a dataset with D variables and B batches or plates, TIGER needs to train $D \times B$ models to normalize the whole dataset. TIGER takes 2 minutes 28.85 seconds, 2 minutes 21.75 seconds, and 57 minutes 30.46 seconds to process the KORA FF4 (103 variables, 22 plates), P20 Negative (268 variables, 4 batches), and Amide (6402 variables, 4 batches) datasets in parallel with 8 cores. In this case, TIGER takes around 140 seconds to build 1,000 models when $20 \sim 30$ QC samples per batch are available for model training. The evaluation of different numbers of highly correlated variables (Table 2.4) shows that the running time of TIGER increases approximately linearly with the number of variables. TIGER trades model complexity for a robust prediction, but the increased cost is still acceptable.

2.4 Case Study

The case study is conducted to illustrate how TIGER can be integrated into longitudinal analysis. This case study, which includes missing data imputation, intra- and inter-batch technical variation removal, cross-kit adjustment, and mixed effects modeling, may serve as a practical analysis pipeline for longitudinal studies.

2.4.1 Data Imputation

KORA F4 (Original) has a few missing values that need to be imputed. To ensure an accurate result, a masked dataset with simulated missing values is created using KORA F4 (Original). Out of 3,061 individuals of 103 metabolites (315,283 data points), six missing values are scattered in the data of four metabolites (PC aa C32:2, PC ae C42:4, SM (OH) C14:1 and SM C20:2). A subset without missing values is first extracted from F4 (Original). For each of these four metabolites, we randomly masked 10 values with the assumption that data of one specific metabolite are MCAR [50]. The resulting masked dataset is then imputed by four popular algorithms [51, 52] namely Predictive Mean Matching (PMM) [53], Classification And Regression Trees (CART) [54], k-NN imputation [55], and Linear Model (LM) [56]. The results are evaluated with MAPE,

Table 2.7: Case Study: Result of Missing Value Imputation

	PMM	k-NN	LM	CART
MAPE	**0.122**	0.130	0.132	0.159

Bold number indicates the best result.

measuring the difference between the original and imputed values. Based on the result (Table 2.7), method PMM achieves the best performance and is selected to impute the missing values in KORA F4 (Original).

2.4.2 Cross-Kit Adjustment

In this case study, we aim to correct datasets measured at three timepoints, namely the KORA S4, F4 (Original), and FF4. The three datasets exhibit obvious inter-batch technical variation (see the raw data distributions in Fig. 2.7). Effective data correction and adjustment will help ensure a reliable downstream analysis. Considering that all three datasets provided the same kind of pooled EDTA-plasma QC sample (QC, five per plate), we use QC to correct the datasets of three timepoints. And the KORA F4 (Original) is adjusted using the 288 repeated measurements of the F4 (Remeasured).

The metabolite concentration levels of the KORA FF4 are used as a reference to align the other datasets for inter-batch correction. After the data correction, the metabolite concentration levels of the KORA S4 and F4 will match the levels of the KORA FF4. Intra-batch technical variation within the data from 29 plates of the KORA FF4 is first eliminated using TIGER. When normalizing the S4, the target values $\bar{y}$ (Eq. 2.9) are calculated from the FF4 so that TIGER attempts to remove the inter-batch technical variation by minimizing the discrepancy between the S4 and FF4. TIGER by default eliminates the technical variation within the input dataset, in which the target values are automatically computed using the input dataset itself. Here, TIGER is configured to calculate the target values from another dataset.

The adjustment of the KORA F4 consists of two steps: (1) Data of the KORA F4 (Remeasured) were generated from four plates of the p180 kit. Data correction is first performed through the method we used for the S4 to combat intra- and inter-batches, such that the values in the F4 (Remeasured) are aligned to the FF4. (2) Then the samples with repeated measurements in the F4 (Original) are used as training samples for cross-kit adjustment. In a broad sense, the noises introduced by different kits can be categorized as inter-batch technical variation, but the noises are further amplified due to the change of analytical kit. In this case, each of these 288 subject samples can be considered as one kind of QC sample. The remeasured data are used as target values to minimize the difference between the KORA F4 (Remeasured) and F4 (Original). After the cross-kit adjustment, the KORA F4 (Original) is comparable to FF4.

To evaluate the quality of the adjusted data, the RSD values of QC1, QC2, and QC3 are calculated on the raw F4 (Original) and adjusted F4 (Original). After TIGER's

Table 2.8: Case Study: Data Characteristics

Characteristics	Samples from S4	Samples from F4	Samples from FF4
Age (years)	61.75 ± 4.90	68.75 ± 4.90	75.75 ± 4.90
Sex (female, %)	49.73	49.73	49.73
BMI (kg/m^2)	26.18 ± 3.10	27.05 ± 3.18	27.00 ± 3.39
Systolic blood pressure (mmHg)	127.37 ± 15.93	124.00 ± 15.73	119.55 ± 16.72
Fasting glucose in serum (mg/dl)	96.95 ± 8.90	96.01 ± 9.30	101.01 ± 13.77
HbA1c (%)	5.57 ± 0.33	5.54 ± 0.30	5.58 ± 0.38

$N = 374$. Values are percentage or mean ± standard deviation.

adjustment, the median of RSD calculated on QC1, QC2, and QC3 reduce from 0.1225, 0.1251, and 0.1070 to 0.0967, 0.0953, and 0.0914, respectively.

While RSD indicates data dispersion, our primary concern is how closely the adjusted remeasured data aligns with the normalized remeasured data. Because QC1, QC2, and QC3 are different in KORA F4 (Original) and F4 (Remeasured), we cannot directly compute MAPE on these manufacturer-provided QC samples. To investigate how well TIGER performs on cross-kit adjustment, we further perform 4-fold stratified CV analysis using the 288 samples with remeasurements from the normalized KORA F4 (Remeasured) and the F4 (Original). MAPE is computed on each validation fold to quantify the difference between the adjusted original data and the normalized remeasured data. In the KORA F4 (Original), the 288 original measured samples were spread across 38 plates. For each plate, we randomly and equally split the samples into four groups. Groups of each plate are combined together to construct 4 folds for stratified CV. After TIGER's adjustment, the median and mean of MAPE were reduced by 16.81% (from 0.250 to 0.081) and 19.29% (from 0.296 to 0.103) respectively, which indicates that TIGER is effective for cross-kit adjustment.

2.4.3 Analysis for Aging Trends

We further use raw and TIGER normalized datasets of KORA S4, F4, and FF4 to investigate age-associated metabolites. To weaken the influence of diseases and medical treatments on metabolite concentration, subject samples from KORA are screened according to their corresponding phenotype data. For each timepoint, we exclude individuals with obesity, defined as the Body Mass Index (BMI) $>$ 35 kg/m^2, or with

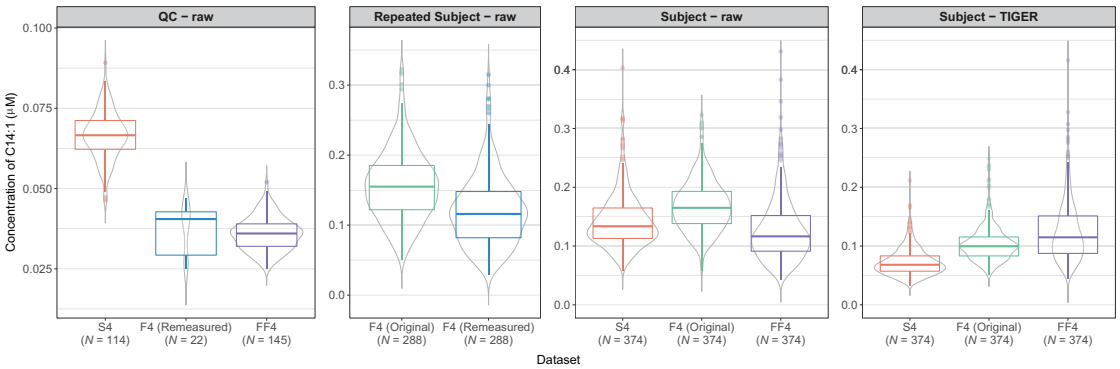

Figure 2.7: Concentration of Metabolite C14:1 from the KORA Datasets. The box of the S4 in the first panel is higher than that of both the F4 (Remeasured) and FF4 for the identical QC samples. The second plot shows that the values of the F4 (Original) are higher than the values in F4 (Remeasured). The last two plots show the concentration of C14:1 from the S4, F4, and FF4 with and without TIGER's normalization. The raw values of subject samples theoretically suffer from similar technical variations to the raw values of QC and remeasured samples. N denotes the number of samples.

hypertension, defined as the systolic blood pressure $>$ 160 mmHg, or with type 1 or type 2 diabetes. Non-fasting samples are also removed. The remaining data available at all three timepoints consist of 374 individuals. The average age of these participants from KORA S4, F4, and are 61.75, 68.75, and 75.75 years old. The mean values of BMI, fasting glucose, and Hemoglobin A1c (HbA1c) were relatively stable during the 14 years investigation, whereas the mean of systolic blood pressure decreased (Table 2.8).

We use a linear mixed-effects model [57] with a random intercept to investigate the relationships between each of the 103 metabolites and age. Overall, 73 metabolites in the normalized data are significantly associated with age with Bonferroni correction (P-value $< 0.05/103$). By contrast, 89 metabolites in the raw data are detected as significant. We notice 38 metabolites show different significance of age associations in raw and TIGER normalized data. Fig. 2.7 shows the concentration distribution of metabolite C14:1 (Tetradecenoylcarnitine). Estimated from the raw data, the regression coefficient of `C14:1` $\sim$ `age` is -5.68e-04 (P-value = 0.0038). When fitting the model using the data processed by TIGER, the regression coefficient goes to 3.16e-03 (P-value = 3.04e-91). The positive correlation found in C14:1 is consistent with recent longitudinal studies showing that C14:1 and C18:1 (Octadecenoylcarnitine) are among the most age-dependent metabolites [58], and many fatty acids, including C14:1 and C18:1, are significantly increased in midlife [59]. The positive correlation between C14:1

and age can only be revealed in the corrected data, which confirms TIGER's effectiveness in technical variation removal. We also notice the positive correlation between age and C18:1. Fitting the models with raw and TIGER corrected data, the regression coefficients of C18:1 $\sim$ age are 2.45e-3 (P-value = 8.13e-53) and 3.14e-3 (P-value = 5.81e-90), respectively. This result implies TIGER does not impair true biological variations within the data.

The detailed association results are available on TIGER's dynamic website, where readers can interactively examine the aging trends associated with different metabolites.

2.5 Discussion

In this study, we developed TIGER, a reliable method for technical variation removal in metabolomics data powered by an adaptable ensemble learning architecture. Evaluated with targeted and untargeted metabolomics datasets, TIGER outperforms four widely-used methods (NF, LOESS, SERRF, and WaveICA) on both intra- and inter-batch technical variation removal tasks. A case study was performed to illustrate how TIGER can be used for the detection of true aging-associated metabolites in longitudinal datasets.

Now more and more advanced machine learning models have been developed to capture the complex but subtle structure hidden within the data. However, a highly sophisticated model may not be robust enough for a dataset with a limited sample size or a large quantity of noise, which is often the case in the omics data. To tackle these issues, an ensemble learning architecture was devised. Instead of turning a weak learner into a strong one or searching for a specific hyperparameter combination that achieves the lowest training error, the architecture improves a model's robustness to data variation and mitigates the risk of overfitting by considering the output from multiple learners. The fundamental concept behind the proposed ensemble architecture is that even base models with weaker performance can contribute to lowering the technical variation and provide information to the ensemble, potentially outperforming a fine-tuned individual model. This advantage broadens the application scope of our ensemble learning architecture, extending its use case beyond technical variation removal in metabolomics data.

In this study, we used three metrics–RSD, MAPE, and PCA–to evaluate different methods. Relying solely on RSD could lead to overly optimistic results and the problem of over-correction, where biological variations are also removed during normalization.

In our case study, MAPE can provide additional insight into TIGER's performance by quantifying how much the adjusted KORA F4 (Original) data deviate from the F4 (Remeasured) data. Additionally, as the repeated subject samples exhibit both technical and biological variations, RSD in this scenario becomes less applicable, making MAPE a more practical metric for reliable evaluation.

We further observed that LOESS and WaveICA displayed different patterns with the increase of technical variation. LOESS achieved the most balanced result among the five methods on the QC samples (Figs. 2.4). However, its normalized subject data still showed strong batch effects (Fig. 2.6). This problem may be explained by the fact that the LOESS-based model captures data patterns that only exist in QC samples and do not fully reflect the entire datasets, thus yielding unfavorable results for subject samples. Although QC samples can be split into training (80%) and test sets (20%), the evaluation still entails the risk of producing over-optimistic results. The method may yield worse results if the training samples are more distinct from the subject samples. To avoid potential bias, we highly recommend using different kinds of evaluation metrics and methods, including, but not limited to, RSD, MAPE, and PCA, to perform data evaluation. Furthermore, it is also beneficial to introduce several kinds of QC samples into the acquisition process of metabolomics data. In the evaluation of the KORA FF4, identical pooled EDTA-plasma QC samples were used to train QC-based methods, while manufacturer-provided QC plasma samples (QC1, QC2, and QC3) and subject serum samples were used to construct the test set and evaluate performance. The overall data distribution of the QC can greatly differ from the distributions of QC1, QC2, and QC3. Therefore, a method with good results on QC1, QC2, and QC3 will theoretically generalize robustly on subject samples. The strong performance of TIGER in the evaluation of the KORA dataset demonstrates that TIGER is effective in reducing technical variation, even if the training data are not fully representative of the test data.

Another observation is that as the technical variation increases, the performance of NF becomes weaker. This linear equation-based method was only inferior to TIGER in the evaluation of the KORA FF4 dataset, but it was surpassed by LOESS, SERRF, and WaveICA on the P20 Negative and Amide datasets. We speculated that this might result from the limited capability of a linear equation in capturing complex data patterns. The raw data of the KORA FF4 used in this study is already of quite high quality. When applied to a dataset with a large quantity of variation, such as the Amide dataset, NF underfits the data and fails to yield satisfactory results. By contrast, SERRF gradually outperformed LOESS and WaveICA with the increase of technical variation. The

favorable performances achieved by TIGER and SERRF may be partly owing to the robustness of the RF algorithm that underlies the methods. The ensemble learning architecture further improves TIGER's effectiveness and robustness, thus achieving better overall performances than SERRF across the three datasets.

We also observed intriguing phenomena from WaveICA's performance. In terms of RSD, WaveICA gradually outperformed other QC-based tools with the increase in data variability. WaveICA achieved the best RSD on Amide dataset, proving WaveICA's effectiveness for batch effect removal, especially when QC samples are not available. However, its high MAPE should not be ignored–this may involve a risk of over-correction. The negative values generated by WaveICA may also hinder the subsequent biological analysis and make the data less interpretable.

In our case study, TIGER was applied to a longitudinal analysis involving data collected at three timepoints and measured with two different kits. We demonstrated that TIGER considerably reduced the technical variation introduced by different kits and improved the homogeneity and comparability of data. Previous studies have shown that the concentrations of many metabolites, including C14:1 and C18:1, are positively correlated with age [41, 60, 59, 58]. We found these associations often become more apparent only after the technical variations are mitigated. Our findings also indicate that many other metabolites are associated with age, as detailed on our interactive website. It would be of great significance to perform in-depth investigations to validate these findings.

TIGER is designed to provide practical and customizable functions for eliminating technical variations, including systematic errors introduced by different analytical kits. To the best of our knowledge, TIGER is the first ready-to-use tool that facilitates cross-kit adjustment. In this study, the adjustment was evaluated using kits from two different versions. The performance of TIGER with two completely different kits has not been tested, as adjusting data from entirely distinct sources may lead to misleading results. Although TIGER does not make any assumptions about the data source and is theoretically applicable to any numeric data measured using various techniques, in this study, it was only evaluated using data from LC-MS analysis.

2.6 Reproducibility and Availability

In order to ensure the reproducibility and accessibility of our computational method, all relevant data, code, and documentation have been made available, and a dynamic website has been developed to provide variable-level performance evaluation.

2.6.1 Data

KORA FF4's QC data are included in TIGER's R package. The cohort data that support the findings of this study are operated by Helmholtz Munich and available via the KORA platform `https://www.helmholtz-munich.de/en/epi/cohort/kora` upon reasonable request.

P20 Negative dataset, provided by SERRF as a demo dataset, is available at `https://slfan2013.github.io/SERRF-online/`. The dataset is downloaded and used with the consent of the authors.

Amide dataset is included in WaveICA's R package and available at `https://github.com/dengkuistat/WaveICA`. The dataset was downloaded and used in compliance with the copyright policy of the publisher.

2.6.2 R Package

The R implementation of TIGER, `TIGERr`, is a free R package under the GNU General Public Licence (GPL). The package is developed with the help of dependencies ppcor [44], randomForest [47], caret [61], and pbapply [62]. The manual, included in Appendix C, is generated with roxygen2 [63].

TIGER is compatible with almost all widely-used OS, including Windows, UNIX/Linux, and macOS. Users can install TIGER in R via command `install.packages("TIGERr")`, and an appropriate version, as well as dependencies, will be installed automatically. The package has been included in CRAN. The package is also available at CRAN (`https://CRAN.R-project.org/package=TIGERr`) and GitHub (`https://github.com/HAN-Siyu/TIGER`).

2.6.3 Dynamic Website

The dynamic website (accessible at `https://han-siyu.github.io/TIGER_web/`) is a shiny-based application [64] which supports interactive figures for the detailed results of KORA-derived datasets. Packages including shinydashboard [65], flexdashboard [66], ggplot2 [67], ggsci [68], and plotly [69] are employed in the background to control layouts, output figures, and enable interactive features.

The dynamic website features two functional modules: one for method evaluation results and another for longitudinal analysis. Users can input the name of a metabolite of interest to compare the results from different methods and to examine its concentration distribution. Relevant statistics are displayed when the cursor hovers over the plots.

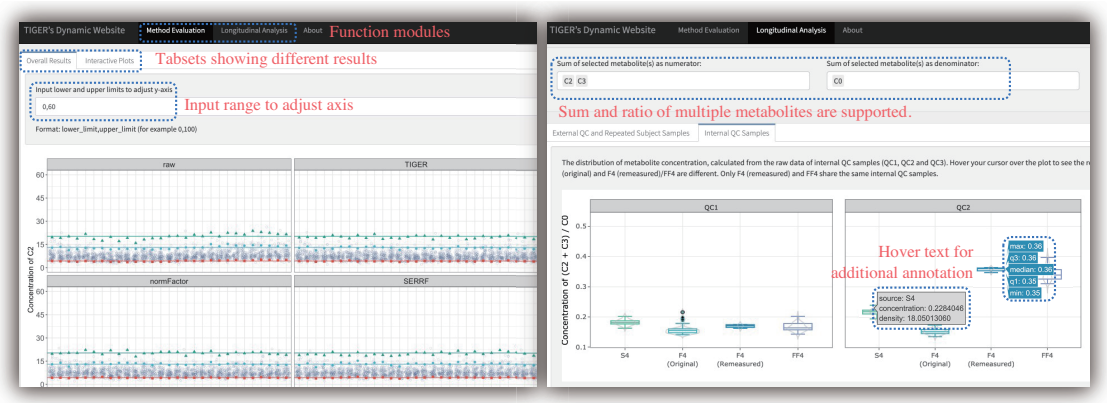

Figure 2.8: Overview of the Dynamic Website.

3 LEOPARD for Missing View Imputation

In this chapter, we present LEOPARD, a pioneering method for imputing missing views in multi-timepoint omics data. We begin with an overview (Section 3.1) of existing methods and their limitations, followed by a detailed explanation of the contributions made by LEOPARD. The methods section (Section 3.2) elaborates on the construction of benchmark datasets and the architecture of LEOPARD, including hyperparameter optimization and ablation tests. LEOPARD is evaluated through comprehensive simulations (Section 3.3), case studies (Section 3.4), and applicability analysis (Section 3.5). The discussion (Section 3.6) provides insights into the broader application of LEOPARD, its limitations, and potential caveats. In the reproducibility and availability section (Section 3.7), we provide details on how to access the reproducible scripts so that LEOPARD can be deployed and tested by other researchers.

3.1 Overview

The rapid advancement of omics technologies has enabled researchers to obtain high-dimensional datasets across multiple views, enabling unprecedented explorations into the biology behind complex diseases [70]. In biomedical analysis, multi-view datasets integrate multiple types of biological information or different perspectives of the same biological phenomenon. Each "view" represents a different type of data or a different aspect of the same biological system, capturing different biological processes or molecular characteristics. For example, different views can correspond to different types of omics data, or data acquired through different platforms, each contributing a partial or entirely independent perspective on complex biological systems [71].

While advancements in multi-omics measurements have increased throughput and enabled the acquisition of multiple views in a single assay [72], data preprocessing, analysis, and interpretation remain significant and important challenges. One of the most pressing challenges is the presence of missing data [73], which at its best (when missingness occurs at random) reduces statistical power, and at its worst (when it is not random)

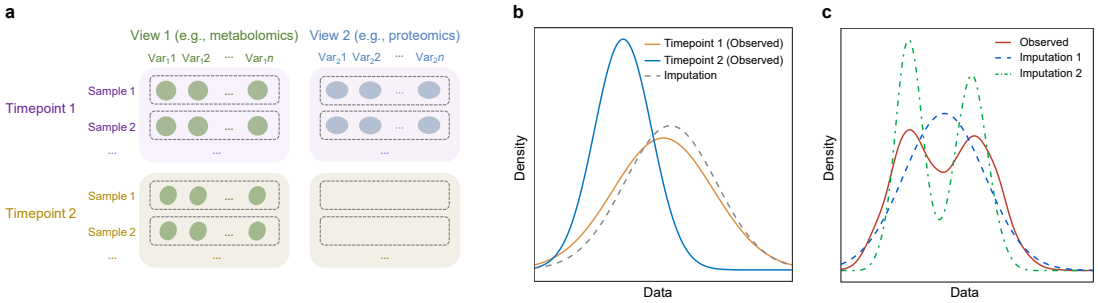

Figure 3.1: Description of Missing View Problem. **a**, An example of a missing view in a longitudinal multi-omics dataset. In this instance, View 1 and View 2 correspond to metabolomics and proteomics data, respectively. Data of View 1 at Timepoint 2 are absent. **b**, An example of data density calculated from a variable in observed data (Timepoint 1 and Timepoint 2) and imputed data. The data density indicates a distribution shift across the two timepoints. Imputation methods developed for cross-sectional data cannot account for the temporal changes within the data, and their imputation models built with data from one timepoint, such as Timepoint 1, might not be appropriate for inferring data from another timepoint, such as Timepoint 2. **c**, Compared to Raw data, data of Imputation 1 may exhibit lower MSE than data of Imputation 2, but Imputation 1 potentially lose biological variations present in the data.

can lead to biased discoveries. Unlike missing data points that may be scattered across the entire dataset, a missing view refers to the complete absence of all features from a certain view, as shown in Fig. 3.1a. In longitudinal studies that can span decades, the problem of missing views in multi-timepoint omics data becomes increasingly common due to factors such as dropout in omics measurements, experimental errors, or unavailability of specific omics profiling platforms at certain timepoints. The missing view is at least MAR, or even MNAR if the missingness is related to the unobserved data themselves. The incompleteness of these datasets hinders multi-omics integration [74] and investigations into predisposing factors (such as age and genetics), enabling factors (such as healthcare service and physical activity), and biomarkers for diseases. However, there is a lack of tailored methods for this critical issue.

3.1.1 Existing Methods and Their Limitations

Missing view imputation or completion refers to the estimation of missing omics data in a multi-view context. The task of missing view imputation in the multi-timepoint scenario is more complex than cross-sectional missing value imputation, as it needs to harmonize data distributions across views [75] and capture temporal patterns [76] (Fig. 3.1a).

Table 3.1: Methods for Data Imputation

Method	Imputation Type	Missing Data	Multi-Timepoint	Environment	Reference
PMM	Multiple	Values	No	R	[21, 22]
GLMM	Multiple	Values	Yes	R	[23, 24]
missForest	Single	Values	No	R	[25]
cGAN	Single	Views	No	Python	[26]
LEOPARD	Single	Views	Yes	Python	This study

Generic methods (Table 3.1), such as PMM [77], missForest [78], and KNNimpute [79], learn direct mappings between views from observed data without missingness. However, this strategy is inadequate or suboptimal [80] for longitudinal data, as it precludes investigations into temporal variation, which can be of great interest: the learned mappings may overfit the training timepoints, making them unsuitable for inferring data at other timepoints, especially when biological variations cause distribution shifts over time (Fig. 3.1b). To address the complexities of longitudinal data, numerous effective imputation methods have been developed based on the Generalized Linear Mixed Model (GLMM) [81]. Existing studies have also explored the use of spline interpolation and Gaussian processes to extrapolate or interpolate missing timepoints [82, 83]. However, the typically limited number of timepoints in current human cohorts can restrict the effectiveness of these longitudinal methods. Given these challenges, there is a growing need for view completion methods that are specifically designed for multi-timepoint omics data. While metrics like the Mean Squared Error (MSE) and Percent Bias (PB) are commonly used to evaluate imputation results [52], these quantitative metrics alone may not fully capture data quality in the context of omics data. As depicted in Fig. 3.1c, data imputed by method 1 may have a lower MSE than that imputed by method 2, but at a loss of biologically meaningful variations. Further case studies would be helpful to evaluate imputation methods.

3.1.2 Contribution

In this paper, we introduce LEOPARD (missing view completion for multi-timepoint omics data via representation disentanglement and temporal knowledge transfer), a neural network-based approach that offers a novel and effective solution to this challenge. LEOPARD extends representation disentanglement [6] and style transfer [84] techniques, which have been widely applied in various contexts such as image classifi-

cation [85], image synthesis [86], and voice conversion [87], to missing view completion in longitudinal omics data. LEOPARD factorizes omics data from different timepoints into omics-specific content and timepoint-specific knowledge via contrastive learning. Missing views are completed by transferring temporal knowledge to the corresponding omics-specific content.

We demonstrate the effectiveness of LEOPARD through extensive simulations using human proteomics and metabolomics data from the MGH COVID study [88] and the KORA cohort [1]. Additionally, we perform two case studies using real omics data to assess whether biological information is preserved in the imputed data, providing a comprehensive assessment of LEOPARD's performance in both regression and classification tasks.

In summary, the key contributions of this study are:

- We propose LEOPARD, a novel method tailored for missing view completion in multi-timepoint omics data that applies representation disentanglement and style transfer in an innovative manner.
- Our study shows that generic imputation methods designed for cross-sectional data are not suitable for longitudinal data, emphasizing the need for tailored approaches. Additionally, we highlight that canonical evaluation metrics do not adequately reflect the quality of imputed biomedical data. Further investigations, including regression and classification analyses, can augment these metrics in the assessment of imputation quality and preservation of biological variations.
- Our research reveals that omics data across timepoints can be factorized into content and temporal knowledge, providing a foundation for further explorations into biological temporal dynamics. This insight offers a novel perspective for predictive healthcare that extends beyond the problem of data imputation.
- LEOPARD is a python-based tool available on GitHub (`https://han-siyu.github.io/TIGER_web/`). The manual is provided in the jupyter-notebook format to help users train their own models for their datasets.

3.2 Methods

In our missing view completion problem, we are given a generalized dataset $\mathcal{X}_{V,T}^{N}$ that includes omics data from N individuals across multiple views (V) and timepoints (T).

Table 3.2: Summary of Benchmark Datasets for Missing View Completion

	MGH COVID	KORA metabolomics	KORA multi-omics
Omics type	proteomics	metabolomics	metabolomics, proteomics
View[1]			
$v1$	panel: cardiometabolic	class: GPL	omics: metabolomics
$v2$	panel: inflammation	class: non-GPL	omics: proteomics
Variable number			
$v1$	322	70	104
$v2$	295	36	66
Timepoint			
$t1$	D0	F4	S4
$t2$	D3	FF4	F4
Time span	3 days	7 years	7 years
Sample number[2]			
Total	218	2,085	1,062
Training	140	1,335	680
Validation	35	333	170
Test	43	417	212

[1]The views in the MGH COVID dataset correspond to two Olink® panels: Explore 384 cardiometabolic ($v1$) and Explore 384 inflammation ($v2$). The 106 metabolites in the KORA metabolomics dataset are classified into five analyte classes: 70 glycerophospholipids (GPL), 15 acylcarnitines, 11 sphingolipids, 9 amino acids, and 1 monosaccharide. One view ($v1$) comprises 70 metabolites from the GPL class, while the other view ($v2$) comprises 36 metabolites from the other classes. The views in the KORA multi-omics dataset correspond to the metabolomics data from the Biocrates AbsoluteIDQ® p180 kit and the proteomics from the Olink® Target 96 Inflammation panel.

[2]The samples from each dataset are split into training, validation, and test sets in a 64%, 16%, and 20% ratio, respectively. The test data in $v2$ at $t2$ are masked for performance evaluation.

For simplification, we consider data from two views $v1$,$v2$ and two timepoints $t1$,$t2$. Data in $\mathcal{X}^N_{V=v2,T=t2}$ are set to be incomplete. The samples from each dataset are split into training, validation, and test sets in a 64%, 16%, and 20% ratio, respectively. We use $\mathcal{D}^{split}_{v,t}$ to denote data split from different views and timepoints. The test data in $v2$ at $t2$ (i.e. $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$) are masked for performance evaluation. We attempt to develop a model that captures the mappings between $v1$ and $v2$, while simultaneously extracting temporal knowledge from $t1$ and $t2$ using all observed data in the training set.

3.2.1 Benchmark Datasets Construction

We evaluate LEOPARD using three real longitudinal omics datasets. These distinct datasets are designed based on data variations, time spans, and sample sizes. The first two are mono-omics datasets, constructed with the proteomics data from the MGH COVID study and the metabolomics from the KORA cohort, respectively. Views in both datasets correspond to panels or biochemical classes. Missingness in these datasets exemplifies a common issue encountered in longitudinal studies where data from certain panels or biochemical classes are incomplete in some but not all timepoints. The third dataset is a multi-omics dataset consisting of both metabolomics and proteomics data from the KORA cohort. In this dataset, views correspond to different omics. This dataset exemplifies the situation where data of a type of omics is incomplete.

3.2.1.1 MGH COVID Dataset

The MGH COVID study includes plasma proteins measured for patients at three timepoints: on day 0 (D0) for all patients on the day they were hospitalized for COVID, and on days 3 (D3) and 7 (D7) for patients still hospitalized then. We utilize proteomics data from D0 ($t1$) and D3 ($t2$), which have the largest sample sizes ($N = 218$, one duplicated sample removed), to construct the first mono-omics dataset.

The proteomics data from the MGH COVID study were obtained using plasma samples and the Olink® Explore 1536 platform (Olink Proteomics, Watertown, MA, USA), which consists of 1,472 proteins across four Olink® Explore 384 panels: inflammation, oncology, cardiometabolic, and neurology [88]. The platform enables relative quantification of analyte concentrations in the form of $\log_2$-scaled Normalized Protein eXpression (NPX) values, where higher values correspond to higher protein levels. We selected proteins listed in the Cardiometabolic and Inflammation panels to construct the two views for the MGH COVID proteomics dataset, and the inflammation view was assumed to be incomplete. The proteins are processed based on the following QC criteria:

- The protein should have no missing values.
- For a specific protein, at least 50% of measured sample values are equal to or above LOD.

After quality inspection, a total of 322 and 295 proteins from the Cardiometabolic ($v1$) and Inflammation ($v2$) panels are selected.

3.2.1.2 KORA Datasets

Data from the KORA study are extracted from S4, F4, and FF4 [89, 90]. We use metabolomics data (N = 2,085) from F4 ($t1$) and FF4 ($t2$) from the KORA cohort to construct the second mono-omics dataset. We additionally use metabolomics and proteomics data (N = 1,062) from the KORA S4 ($t1$) and F4 ($t2$) to construct the multi-omics dataset.

As the benchmark datasets are used for evaluating imputation results, some QC criteria are different from the ones we used in the previous chapter to evaluate TIGER. Here, only metabolites meeting the following QC criteria [42, 91] are selected (Table B.1):

- The metabolite should be available in both p150 and p180 measured data.
- The metabolite should have no missing values.
- For a specific metabolite, at least 50% of measured sample values are equal to or above LOD of corresponding plates.
- The median RSD of QC samples should be $< 25\%$.
- The Spearman CC of the metabolite values measured by the KORA F4 (Remeasured) and F4 (Original) should be > 0.5.

After QC procedures, the metabolite data are further normalized using TIGER [92] with its default setting to remove the plate effects. For the multi-omics dataset, TIGER was also used to remove the technical variation introduced by different kits following our previous protocol [92, 93].

As to proteomics data, only proteins that pass the following QC criteria are selected (Table B.2):

- The protein should have no missing values.
- For a specific protein, at least 75% of the measured sample values are equal to or above LOD.

For the KORA metabolomics dataset, 106 targeted metabolites satisfy all criteria and are categorized into five analyte classes: acylcarnitine, amino acid, glycerophospholipid, sphingolipid, and monosaccharide. Two view groups are constructed by 70 glycerophospholipids ($v1$) and 36 metabolites from the other four classes ($v2$). For the KORA

multi-omics dataset, 104 metabolites ($v1$) and 66 proteins ($v2$) satisfy all QC criteria and are selected to construct two views (3.2).

To evaluate LEOPARD's capability for arbitrary temporal knowledge transfer, we further expanded the KORA metabolomics dataset to include data from S4 (as $t1$) and FF4 (as $t3$), spanning approximately 14 years. We divide the metabolites into two views, following the same strategy as the original KORA metabolomics dataset. Due to different QC results across the two analytical kits, two metabolites, specifically PC aa C38:1 in $v1$ and C16:2 in $v2$, are excluded. The final dataset comprises 102 metabolites with 614 individuals who have data at both timepoints. These samples are also divided into training, validation, and test sets with a ratio of 64%, 16%, and 20% respectively, corresponding to 393, 98, and 123 samples. The data in $\mathcal{D}^{\text{test}}_{v=v2,t=t3}$ are masked for performance evaluation.

3.2.2 CGAN Architecture as a Reference Method

Existing neural network-based missing view completion methods [94, 95, 96] have shown remarkable performance in the field of computer vision. Given their inapplicability to omics data, we designed a conditional Generative Adversarial Networks (cGAN)[97] model specifically tailored for omics data, as a reference method. The cGAN extends the original GAN [98] model by introducing additional information into the generation process, thereby providing more control over the generated output. Our adapted architecture is inspired by VIGAN (View Imputation via Generative Adversarial Networks) [99] and a method proposed by Cai et al. [7], both initially designed for multi-modality image completion. In our context, the completion of missing views is conditioned on the data from observed views. Moreover, we enhanced the baseline cGAN model with an auxiliary classifier28 to ensure that the imputed view can be paired with the corresponding observed view.

3.2.2.1 Architecture Design and Implementation

In the training phase, the generator of the cGAN is trained on observed data from the training set to capture the mappings between two views. The discriminator guides the generator to produce data with a distribution similar to that of actual data. The discriminator also has an auxiliary classifier [100] to ensure the generated data can be paired with input data. In the inference phase, the generator utilizes the mappings it has learned from the observed data to impute the missing view in the test set. Compared to

methods PMM and missForest, our cGAN model has the potential to learn more complex mappings between views. However, these three methods are not able to capture temporal changes within longitudinal data and can only learn from samples where both views are observed.

Specifically, the generator G_{cGAN} learns the complex mappings between $\mathcal{D}^{\text{train}}_{v=v1,t}$ and $\mathcal{D}^{\text{train}}_{v=v2,t}$, with $t = t1$ in our case. The reconstruction loss $\mathcal{L}_{rec\_cGAN}$ quantifies the differences between the actual and reconstructed data. The discriminator D_{cGAN} computes the adversarial loss $\mathcal{L}_{adv\_cGAN}$ by distinguishing if data are real ($\mathcal{D}^{\text{train}}_{v=v2,t=t1}$) or generated ($\hat{\mathcal{D}}^{\text{train}}_{v=v2,t=t1}$). D_{cGAN} also has an auxiliary classifier that computes the auxiliary loss $\mathcal{L}_{aux\_cGAN}$ by predicting whether the pairs of views are real, i.e.,$\mathcal{D}^{\text{train}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$, or fake, i.e., $\mathcal{D}^{\text{train}}_{v=v1,t=t1}$, $\hat{\mathcal{D}}^{\text{train}}_{v=v2,t=t1}$. The final loss is defined as:

$$\mathcal{L}_{cGAN} = w_{rec\_cGAN} \times \mathcal{L}_{rec\_cGAN} + w_{adv\_cGAN} \times \mathcal{L}_{adv\_cGAN} + w_{aux\_cGAN} \times \mathcal{L}_{aux\_cGAN}, \tag{3.1}$$

where w_{rec_cGAN}, w_{adv_cGAN}, and w_{aux_cGAN} are weights for the corresponding losses. After training, the generator G_{cGAN} is applied to $\mathcal{D}^{\text{test}}_{v=v1,t=t2}$ to generate $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$.

The generator of the cGAN model consists of several residual blocks [101] and uses the Parametric Rectified Linear Unit (PReLU) [102] as its activation function. Each residual block includes batch normalization [103] as necessary. MSE loss serves as $\mathcal{L}_{rec\_cGAN}$. Both the MSE loss and the Binary Cross-Entropy (BCE) loss are considered for $\mathcal{L}_{adv\_cGAN}$ and $\mathcal{L}_{aux\_cGAN}$, determined by the hyperparameter tuning experiments. MSE and BCE losses are defined as:

$$\mathcal{L}_{MSE} = \frac{1}{N} \|x^i - \hat{x}^i\|^2, \tag{3.2}$$

$$\mathcal{L}_{BCE} = -\frac{1}{N} \sum_{i=1}^{N} \left[y^i \log\left(\hat{y}^i\right) + \left(1 - y^i\right) \log\left(1 - \hat{y}^i\right) \right], \tag{3.3}$$

where N is the number of samples, x^i and $\hat{x}^i$ represents the true and estimated values for the i-th sample, while y^i and $\hat{y}^i$ represents the true and predicted labels for the i-the sample.

The cGAN model is trained with the Adam optimizer [104], with a mini-batch size of 16 for the MGH COVID dataset and 32 for the two KORA-derived datasets. The model is implemented using PyTorch [105] (v1.11.0), PyTorch Lightning [106] (v1.6.4), and

tensorboard [107] (v2.10.0), and run on a Graphics Processing Unit (GPU) operating the Compute Unified Device Architecture (CUDA, v11.3.1).

3.2.2.2 Hyperparameter Optimization

We utilize the training (64%) and validation (16%) sets from the KORA metabolomics dataset, which has the largest sample size among the three benchmark datasets, to optimize the hyperparameters. We employ a grid search over various combinations of hidden layer size, numbers of residual blocks, batch normalization, and weights for different losses. We determine the number of training epochs based on early stopping triggered by the MSE reconstruction accuracy calculated on the observed data from the validation sets. Our experiments include variations in the number of hidden neurons for both the generator and discriminator, with options including 32, 64, 128, and 256. The number of residual blocks spanned from 2 to 6.

The final hyperparameters comprise five residual blocks of 64 neurons each for the generator and three hidden layers of 128 neurons each for the discriminator. Batch normalization is incorporated into the first four residual blocks of the generator and the last two layers of the discriminator to stabilize the learning process and accelerate convergence. A weight of 0.5 to $\mathcal{L}_{rec\_cGAN}$ and a weight of 0.25 to $\mathcal{L}_{adv\_cGAN}$ and $\mathcal{L}_{aux\_cGAN}$ are determined. The MSE loss is selected for both $\mathcal{L}_{adv\_cGAN}$ and $\mathcal{L}_{aux\_cGAN}$. The determined hyperparameters are fixed and used in all evaluations.

3.2.3 LEOPARD Architecture

As illustrated in Fig. 3.2d, the LEOPARD architecture comprises several hierarchical components. First, data of each view are transformed into vectors of equal dimensions using corresponding pre-layers. Subsequently, omics data of all views are decomposed into content and temporal representations. The content encoder captures the intrinsic content of the views, while the temporal encoder extracts knowledge specific to different timepoints. A generator then reconstructs observed views or completes missing views by transferring the temporal knowledge to the view-specific content using Adaptive Instance Normalization (AdaIN) [84]. Lastly, we use a multi-task discriminator to discriminate between real and generated data.

The LEOPARD model is trained by minimizing four types of losses: contrastive loss, representation loss, reconstruction loss, and adversarial loss. An ablation test is performed to evaluate the contribution of each loss. Minimizing Normalized Temperature-

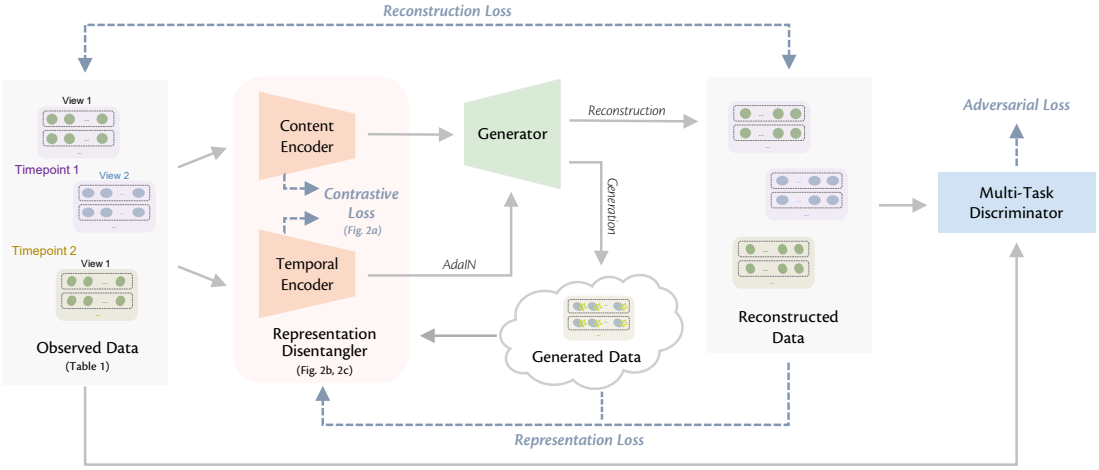

Figure 3.2: Overview of the Architecture of LEOPARD. The architecture of LEOPARD. Omics data from multiple timepoints are disentangled into omics-specific content representation and timepoint-specific temporal knowledge by the content and temporal encoders. The generator learns mappings between two views, while temporal knowledge is injected into content representation via the AdaIN operation. The multi-task discriminator encourages the distributions of reconstructed data to align more closely with the actual distribution. Contrastive loss enhances the representation learning process. Reconstruction loss measures the MSE between the input and reconstructed data. Representation loss stabilizes the training process by minimizing the MSE between the representations factorized from the reconstructed and actual data. Adversarial loss is incorporated to alleviate the element-wise averaging issue of the MSE loss.

scaled Cross-entropy (NT-Xent)-based contrastive loss [108] optimizes the factorization of data into content and temporal representation. For both representations, minimizing the contrastive loss brings together the data pairs from the same view or timepoint and pushes apart the data pairs from different ones, so that the encoders learn similar intrinsic content (or temporal knowledge) across timepoints (or views). The representation loss, also computed on content and temporal representations, measures the MSE of the representations factorized from the actual and reconstructed data. LEOPARD minimizes this loss based on the intuition that the representations of the actual and reconstructed data should be alike. The reconstruction loss measures the MSE between imputed and observed values. Previous studies [109, 110, 111] have demonstrated that the optimization of MSE loss often results in averaged outputs, leading to blurring effects when generating images. In our context, this might diminish biological variations present in omics data. To alleviate this issue, we use adversarial loss to encourage the predicted distribution to align more closely with the actual distribution.

LEOPARD has three unique features compared to conventional architectures for multi-view data completion:

- Instead of focusing on direct mappings between views, which can only be learned from paired data where both views are present, LEOPARD formulates this imputation task in terms of representation learning and style transfer. This allows LEOPARD to utilize all available data, including observations present in only one view or timepoint.
- It incorporates contrastive loss to disentangle the representations unique to views and timepoints, which enables the model to learn more generalized and structured representations. Our experiments show that this is of importance to improve data quality.
- It uses the multi-task discriminator to solve multiple adversarial classification tasks simultaneously by yielding multiple binary prediction results, which has been proven to be more efficient and effective than a discriminator for a multi-class classification problem [112].

3.2.3.1 Architecture Design and Implementation

View-specific pre-layers E^v_{pre} are used to embed input data $x^i_{v,t} \in \mathcal{D}^{\text{train}}_{v,t}$ of different views into dimensionally uniform embeddings $z\_pre^i_{v,t}$. Here, i represents the data from the i-th individual. The representation disentangler of LEOPARD comprises a content encoder E_c and a temporal encoder E_t, both shared by input data across different views and timepoints. This module learns a timepoint-invariant content representation $z\_content^i_{v,t}$ and temporal feature $z\_temporal^i_{v,t}$ from $z\_pre^i_{v,t}$. Following the encoding process, the generator G employs the AdaIN technique to re-entangle content representation and temporal knowledge:

$$\begin{aligned}\text{AdaIN}(z\_content, z\_temporal) = \\ \sigma(z\_temporal) \times \frac{z\_content - \mu(z\_content)}{\sigma(z\_content)} + \mu(z\_temporal)\end{aligned} \tag{3.4}$$

where μ and σ denote the mean and standard deviation operations respectively. View-specific post-layers E^v_{post} convert the re-entangled embeddings back to omics data $\hat{x}^i_{v,t}$. The discriminator D is trained to classify whether an input is a real sample or a generated

output coming from G and E^v_{post}. D produces the same number of outputs as the source classes of the observed data, each corresponding to one view at one timepoint. For a sample belonging to source class $c_{v,t}$, we penalize D during the update cycle of D if its output incorrectly classifies a real data instance as false or a generated data instance as true for $c_{v,t}$; when updating G, we only penalize G if D correctly identifies the generated data instance as false for $c_{v,t}$.

In our study, we define the contrastive loss $\mathcal{L}_{con}$ as the mean of the NT-Xent losses calculated separately for content and temporal representations. The NT-Xent loss is formulated as follows:

$$\mathcal{L}_{\text{NT-Xent}}\left(z^i, z^j\right) = -\log \frac{\exp\left(\text{sim}\left(z^i, z^j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp\left(\text{sim}\left(z^i, z^k\right)/\tau\right)}, \tag{3.5}$$

where z^i and z^j are embeddings of a positive pair (i, j); τ is a temperature factor that scales the similarities; $\mathbb{1}_{[k\neq i]}$ is an indicator function that equals 1 when $k \neq i$ and 0 otherwise. Function $\text{sim}(\cdot)$ denotes cosine similarity, defined as:

$$\text{sim}(a, b) = \frac{a \cdot b}{\|a\|\|b\|}. \tag{3.6}$$

In each training iteration, we generate data for the missing view before calculating the loss. This allows the generated data to be factorized into corresponding content and temporal representations, further facilitating loss minimization.

For the $\mathcal{L}_{\text{NT-Xent}}$ calculated on content representations, positive pairs are defined as $\langle z\_content^i_{v=v1,t=t1},\ z\_content^i_{v=v1,t=t2}\rangle$ and $\langle z\_content^i_{v=v2,t=t1}, z\_content^i_{v=v2,t=t2}\rangle$, which are the same kind of content embeddings from the same individuals across different timepoints. Similarly, the positive pairs of temporal representations are from the same timepoint but different views including $\langle z\_temporal^i_{v=v1,t=t1},\ z\_temporal^i_{v=v2,t=t1}\rangle$ and $\langle z\_temporal^i_{v=v1,t=t2}, z\_temporal^i_{v=v2,t=t2}\rangle$. The representation loss $\mathcal{L}_{rep}$ is the mean of the MSE losses calculated for content and temporal representations. For each type of representation, LEOPARD measures the MSE between the representation factorized from the actual data and reconstructed data. The reconstruction loss $\mathcal{L}_{rec}$ quantifies the discrepancies between the actual and reconstructed data. Any missing values in the observed view are encoded as the mean values across each specific variable, and these mean-encoded values are excluded from the computation of $\mathcal{L}_{rec}$ during back-propagation. This strategy enhances the robustness of LEOPARD in scenarios where

input data contain missing values. The generator can arbitrarily produce data for any source classes given the content and temporal representations.

To ensure the representation disentangler can capture the highly structured data pattern, we only compute the $\mathcal{L}_{rec}$ on the data generated from content and temporal representations derived from different source classes. For instance, $\langle z\_content^{i}_{v=v1,t=t1}, z\_temporal^{i}_{v=v2,t=t1}\rangle$ or $\langle z\_content^{i}_{v=v2,t=t2}, z\_temporal^{i}_{v=v1,t=t1}\rangle$. Data generated from representation pairs of the same views and timepoints, such as $\langle z\_content^{i}_{v=v1,t=t2}, z\_temporal^{i}_{v=v1,t=t2}\rangle$ are not used for optimization. This design imposes additional restraints, and LEOPARD is tamed to learn more generalized representations, which helps prevent overfitting.

Similar to the cGAN model described in the previous section, the adversarial loss $\mathcal{L}_{adv}$ is also computed based on the MSE. The final loss of LEOPARD is defined as:

$$\mathcal{L}_{\text{LEOPARD}} = w_{con} \times \mathcal{L}_{con} \; + \; w_{rep} \times \mathcal{L}_{rep} \; + \; w_{rec} \times \mathcal{L}_{rec} + \; w_{adv} \times \mathcal{L}_{adv}, \tag{3.7}$$

where w_{con}, w_{rep}, w_{rec}, and w_{adv} are the weights of the losses. The encoders, generator, and discriminator of LEOPARD are built from blocks of layers without skip connections. Each block starts with a dense layer. An instance normalization layer [113] is added after the dense layer for the content encoder. The encoders and generator use the PReLU as their activation functions, while the discriminator uses the sigmoid function. A dropout layer [114] is incorporated after the activation layer, where necessary.

The LEOPARD model is trained with the Adam optimizer, with a mini-batch size of 64. The model is implemented under the same computational environment as the cGAN model.

3.2.3.2 Ablation Test

We conduct a comprehensive ablation study to assess the individual contributions of the four distinct losses incorporated into our LEOPARD architecture. By excluding each loss, we benchmark the performance against a baseline setting that only utilizes reconstruction loss. The ablation test is performed with the training and validation sets from the KORA metabolomics dataset. Grid search is used to determine the optimal weights for the losses, and the median of PB computed from the validation set is used to quantify the performance. The network layer numbers and sizes are consistent during the evaluation. We use three hidden layers for the generator and encoders, with each

layer containing 64 neurons. The weight for reconstruction loss is fixed at 1, while the weights for the other three losses vary across 0.01, 0.05, 0.1, 0.5, and 1. The number of training epochs is determined by the model's saturation point in learning, which is when the median of PB computed on the validation set ceases to decrease significantly. Our experiments show that all four losses contribute to lowering PB in the imputed data.

Figure 3.3: Ablation Test on Losses. The ablation test evaluates the contribution of four losses: reconstructive loss (rec), representation loss (rep), contrastive loss (con), and adversarial loss (adv). Grid search strategy is used to determine the optimal weights for different losses, and the median PB computed from the validation set is used to quantify the performance. The network layers are consistent during the evaluation. The plot shows the performance of reconstruction loss combined with (**a**) a single loss and (**b**) multiple losses, using the optimal weight of each loss. The weight for rec is fixed at 1, and the weights for the other losses vary across 0.01, 0.05, 0.1, 0.5, and 1. The optimal weights for different losses in each combination are: rec + rep ($w_{rep} = 0.01$), rec + con ($w_{con} = 0.01$), rec + adv ($w_{adv} = 0.5$), rec + rep ($w_{rep} = 0.01$) + adv ($w_{adv} = 1$), rec + con ($w_{con} = 0.01$) + adv ($w_{adv} = 1$), rec + rep ($w_{rep} = 0.5$) + con ($w_{con} = 0.01$), rec + rep ($w_{rep} = 0.1$) + con ($w_{con} = 0.1$) + adv ($w_{adv} = 1$).

The optimal weights include $w_{rec} = 1$, $w_{con} = 0.1$, $w_{rep} = 0.1$, and $w_{adv} = 1$. The performance of each loss combination at their optimal weights is summarized in Fig. 3.3.

3.2.3.3 Further Hyperparameter Optimization

The training and validation sets from the KORA metabolomics dataset are further used to optimize LEOPARD's hyperparameters, aiming to effectively capture data structure for existing data reconstruction and achieve robust generalization for missing data imputation. The weights for different losses have been determined in the ablation test. We then conduct a grid search across various combinations of hidden layer size, hidden layer number, dropout rate, projection head, and temperature for contrastive loss.

The number of hidden neurons within the encoders, generator, and discriminator vary across 32, 64, 128, and 256, with the number of layers ranging from 2 to 4, and dropout

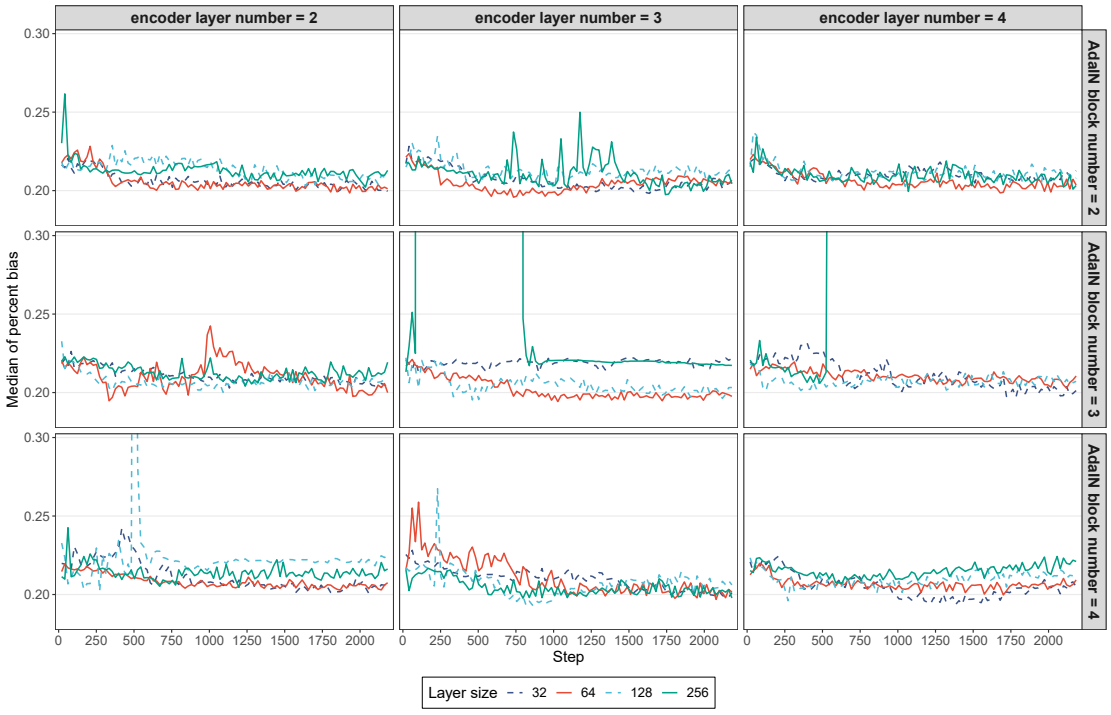

Figure 3.4: Hyperparameter Tuning: Network Width and Depth. Grid search strategy is used to determine the optimal layer number for the content and temporal encoders, AdaIN block number for the generator, and size for each layer. The median PB computed from the validation set is used to quantify the performance. The model achieves the lowest median PB with a setting of three layers for the encoders, each containing 64 neurons, and three AdaIN blocks for the generators, each including a 64-neuron layer.

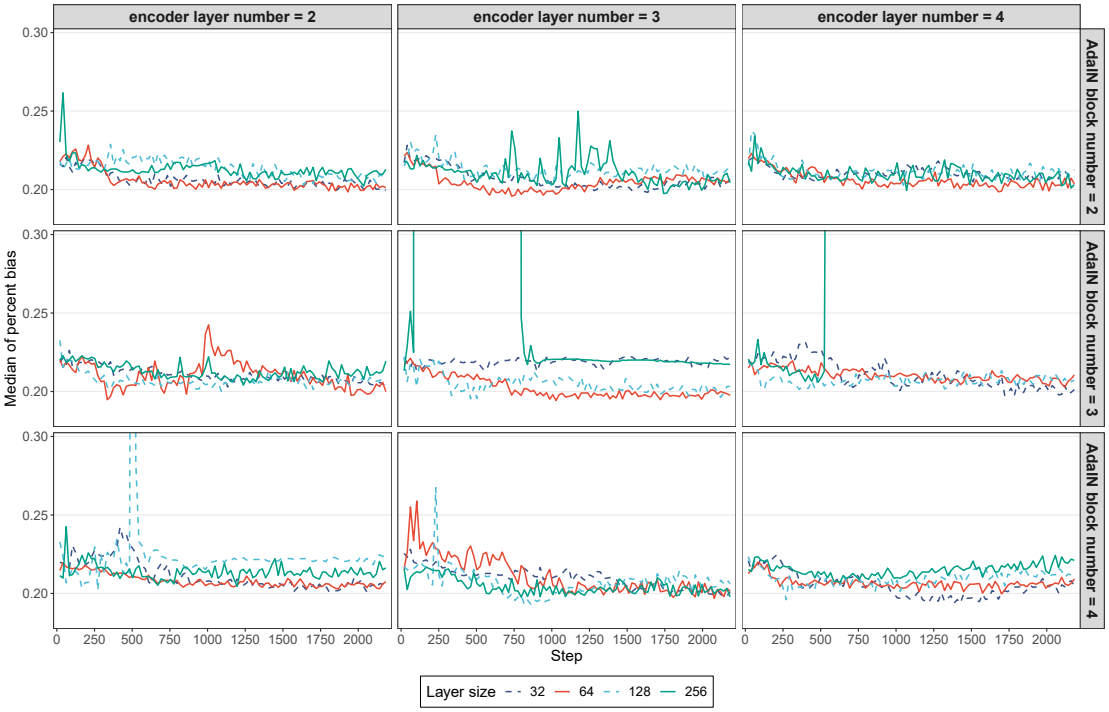

Figure 3.5: Hyperparameter Tuning: Contrastive Loss. Based on the hyperparameters determined in the previous experiments, we further fine-tuned the projection head and temperature used in our contrastive loss. We use grid search to evaluate the performance of LEOPARD without a projection head and with a projection head of different output sizes. The temperature varies across 0.05, 0.1, 0.5, 1, 5, 10, and 30. Based on our experiments, LEOPARD is trained with a temperature of 0.05 without using a projection head.

rates of 0%, 30%, and 50%. Our findings show that higher numbers of hidden neurons and layers tend to yield worse performance in terms of median PB (Fig. 3.4). LEOPARD is configured with three 64-neuron layers incorporated into both content and temporal encoders and the generator. The discriminator includes two hidden layers, each having 128 neurons. Dropout is not used.

The projection head and temperature are two important hyperparameters that control the performance of contrastive learning. The projection head is a compact network consisting of one full connected hidden layer with the same layer size as the input dimension, a Rectified Linear Unit (ReLU) [115] and one output layer. The temperature is a scalar that scales the similarities before the softmax operation. Some previous experiments performed on image datasets emphasized the importance of the projection head and reported different output sizes yielded similar results [108]. We evaluate the performance

of LEOPARD both without a projection head and with a projection head of the output size varying across 16, 32, 64, 128, 256, and 512. The temperature is fined-tuned across 0.05, 0.1, 0.5, 1, 5, 10, and 30. Based on our experiments, LEOPARD is trained with a temperature of 0.05 and without using a projection head (Fig. 3.5).

The determined hyperparameters, including loss weights, remain unchanged in all our performance evaluations.

3.2.4 Strategy for Performance Evaluation

Due to the lack of established methods specifically designed for missing view completion in multi-timepoint omics datasets, we benchmark LEOPARD against three widely recognized generic imputation methods: missForest, PMM, and GLMM, as well as a cGAN model designed for this study. The cGAN serves as a reference model to demonstrate how existing neural network approaches, typically suited for cross-sectional data, perform in longitudinal scenarios. MissForest, as a representative non-parametric method, was chosen for its robustness and ability to handle complex, non-linear relationships among variables. PMM and GLMM, both implemented within the Multivariate Imputation by Chained Equations (MICE) [51] framework, represent established multiple imputation methods that not only address missing values but also allow for the assessment of imputation uncertainty. GLMM, with its ability to capture temporal patterns inherent in longitudinal data, is particularly advantageous for data imputation in longitudinal scenarios.

We assess the performance of LEOPARD, cGAN, missForest, PMM, and GLMM on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ of each benchmark dataset. During training, the methods build models using data from $\mathcal{D}^{\text{train}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$, and $\mathcal{D}^{\text{train}}_{v=v1,t=t2}$, along with different numbers of training observations ($obsNum$) from $\mathcal{D}^{\text{train}}_{v=v2,t=t2}$. By varying $obsNum$, we assess how additional observed data from $t2$ affects their imputation performances. When $obsNum$ is zero, data of $v2$ at $t2$ are assumed as completely missing, and GLMM cannot be trained due to limited longitudinal information. In this scenario, we train a [56] to complete the missing view. This additionally allows us to evaluate how the performance of GLMM compares to that of the simpler LM method.

Apart from missing views, the presence of missing data points in observed views is also very common in omics analysis. Therefore, LEOPARD is designed to tolerate a small number of missing data points in the observed views. We further use the KORA metabolomics dataset, which has the largest sample size, to evaluate the performance of different methods when observed views contain missing values. We simulate missing

values by randomly masking 1%, 3%, 5%, 10%, and 20% of the data in the observed views (*maskObs*) under the assumption that data points are MCAR. The experiment is repeated 10 times for each specified proportion.

3.2.4.1 Methods Configuration

The LEOPARD and cGAN models are trained using the hyperparameters previously described. For missForest, the imputation is performed using a 100-tree RF [27] model, with the maximum number of iterations (`maxit`) set to 10. Multiple imputations by PMM, LM, and GLMM are performed using the R packages mice [51] and micemd [116]. Each method's imputations are performed five times (`m = 5`) with a `maxit` value set to five. The PMM model is built using argument `method = "pmm"`. When $obsNum = 0$, data of $v2$ at $t2$ are assumed to be completely missing. In this scenario, the LM method is trained using `method = "norm"`. When $obsNum$ is a non-zero value, the GLMM model is as built using `method = "2l.glm.norm"`.

All methods used only the data in the training sets to build imputation models. Their performance is evaluated on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$. Different imputation methods may require specific data structures for input: cGAN and LEOPARD first build imputation models using training data, then apply the built models to test set to complete missing views. In contrast, the input data for other methods can be an incomplete matrix with missing values coded as `NA`. We adapt the input data accordingly to accommodate these specific requirements:

- Method cGAN only learns from samples where both views are present. Therefore, its training data only include training data from the first timepoint ($\mathcal{D}^{\text{train}}_{v=v1,t=t1}$ and $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$) and data of different $obsNum$ from the second timepoint ($\mathcal{D}^{\text{train}}_{v=v1,t=t2}$ and $\mathcal{D}^{\text{train}}_{v=v2,t=t2}$).

- LEOPARD can additionally learn from data where only one view is available. In addition to $\mathcal{D}^{\text{train}}_{v=v1,t=t1}$ and $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$, the entire $\mathcal{D}^{\text{train}}_{v=v1,t=t2}$ is included in its training. The variation of $obsNum$ only affects the number of observed samples from $\mathcal{D}^{\text{train}}_{v=v2,t=t2}$.

- The input data for missForest combine training data (including $\mathcal{D}^{\text{train}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$, $\mathcal{D}^{\text{train}}_{v=v1,t=t2}$, and data of different $obsNum$ from $\mathcal{D}^{\text{train}}_{v=v2,t=t2}$) and test data ($\mathcal{D}^{\text{test}}_{v=v1,t=t2}$), with `NA` filling the masked data in the matrix.

- For the multiple imputation methods in MICE family, the input data are constructed with training data (identical to that used for missForest) and test data (including $\mathcal{D}^{\text{test}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{test}}_{v=v2,t=t1}$ and $\mathcal{D}^{\text{test}}_{v=v1,t=t2}$). To ensure test data remain unused for model training, a logical vector with `TRUE` assigned to test samples is passed to the ignore argument. Masked values are filled with `NA` in the matrix.

- When building the GLMM model, the input data additionally contain sample IDs and timepoint labels. A constant residual error variance is assumed for all individuals. Building a GLMM model for a large dataset is extremely time-consuming; thus, for the MGH COVID dataset, we select the top 100 highly Spearman-correlated proteins for each protein requiring imputation. The selected proteins are incorporated into the imputation process by passing to the argument `predictorMatrix`.

3.2.4.2 Evaluation Metrics

Two evaluation metrics, PB and Uniform Manifold Approximation and Projection (UMAP) [117], are used for performance evaluation. PB and UMAP are also calculated for each individual imputation of the multiple imputation methods. When evaluating *maskObs*, the PB values are averaged across 10 repetitions, and the UMAP plots visualize the repetition that exhibits the lowest median of PB.

PB is selected as a performance metric as it quantifies the relative deviation of imputed values from actual observations, offering a more straightforward interpretation compared to metrics like Root Mean Square Error (RMSE), MSE, and Mean Absolute Error (MAE). PB is calculated for each variable using the formula:

$$\text{PB}_i = \frac{1}{m} \sum_{imp\ =1}^{m} \text{median} \left(\frac{\left| \hat{x}^i_{(imp)} - x^i \right|}{x^i} \right), \tag{3.8}$$

where $\hat{x}^i_{(imp)}$ is the imputed value for the i-th variable from the imp-th imputation, while m is the number of imputations determined by the argument `m` in multiple imputation methods. For single imputation methods, LEOPARD, cGAN, and missForest, $m = 1$. PB results for each imputation method are visualized using dot and box plots, with each dot representing a variable in the specific dataset.

PB offers a variable-level assessment of imputation performance, while UMAP visualization illustrates overall similarities between observed and imputed datasets, providing a dataset-level evaluation. For the evaluation using UMAP, we first fit a UMAP model using the data of $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$ and $\mathcal{D}^{\text{train}}_{v=v2,t=t2}$. We then use the fitted model to embed the

data of $\mathcal{D}^{\text{test}}_{v=v2,t=t1}$, $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$, and $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t1}$, where $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ represents the imputed data for $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ produced by different imputation methods. For PMM, LM, and GLMM, $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ is the average of all estimates from their multiple imputations. An imputation method is considered effective if the distribution of $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ embeddings is highly similar to that of the $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ embeddings. The UMAP models are fitted with the identical configurations described in the prior section.

3.3 Performance Evaluation

In this section, we provide a comprehensive evaluation of LEOPARD's performance. Initially, we employ UMAP to demonstrate LEOPARD's effectiveness in representation disentanglement. Following this, we benchmark LEOPARD against standard imputation tools using both the mono-omics and multi-omics datasets to validate its effectiveness and robustness across different data types. Additionally, we conduct a analysis of the variables that result in high PB to reveal potential areas for improvement and understand the limitations of the method. To evaluate the adaptability of LEOPARD, we lastly examine LEOPARD's performance in scenarios where the observed views contain missing values, which are essential for real-world applications where incomplete data is common.

3.3.1 Representation Disentanglement of LEOPARD

We use the KORA multi-omics dataset to examine if LEOPARD can effectively disentangle content and temporal representations from omics data. The disentanglement of content and temporal representations is evaluated using the KORA multi-omics dataset. In this analysis, the model is trained for 600 epochs to ensure that the contrastive loss stabilizes and reaches full saturation (Fig. 3.6a). We use the UMAP for visualizing the content and temporal representations of the validation set across different views and timepoints. As the training progresses, we expect similar representations to gradually cluster together in UMAP, while dissimilar ones form distinct clusters.

The disentanglement progress is visualized with the following steps: First, content and temporal representations are factorized from the metabolomics ($t1$ and $t2$) and proteomics data ($t1$). Then the generator imputes the proteomics data ($t2$) by incorporating the temporal information from the metabolomics data ($t2$) into the content representation from the proteomics data ($t1$). The generated proteomics data ($t2$) are then fed to the content and temporal encoders to extract the corresponding representations. Subsequently, these content or temporal representations of both the observed and imputed

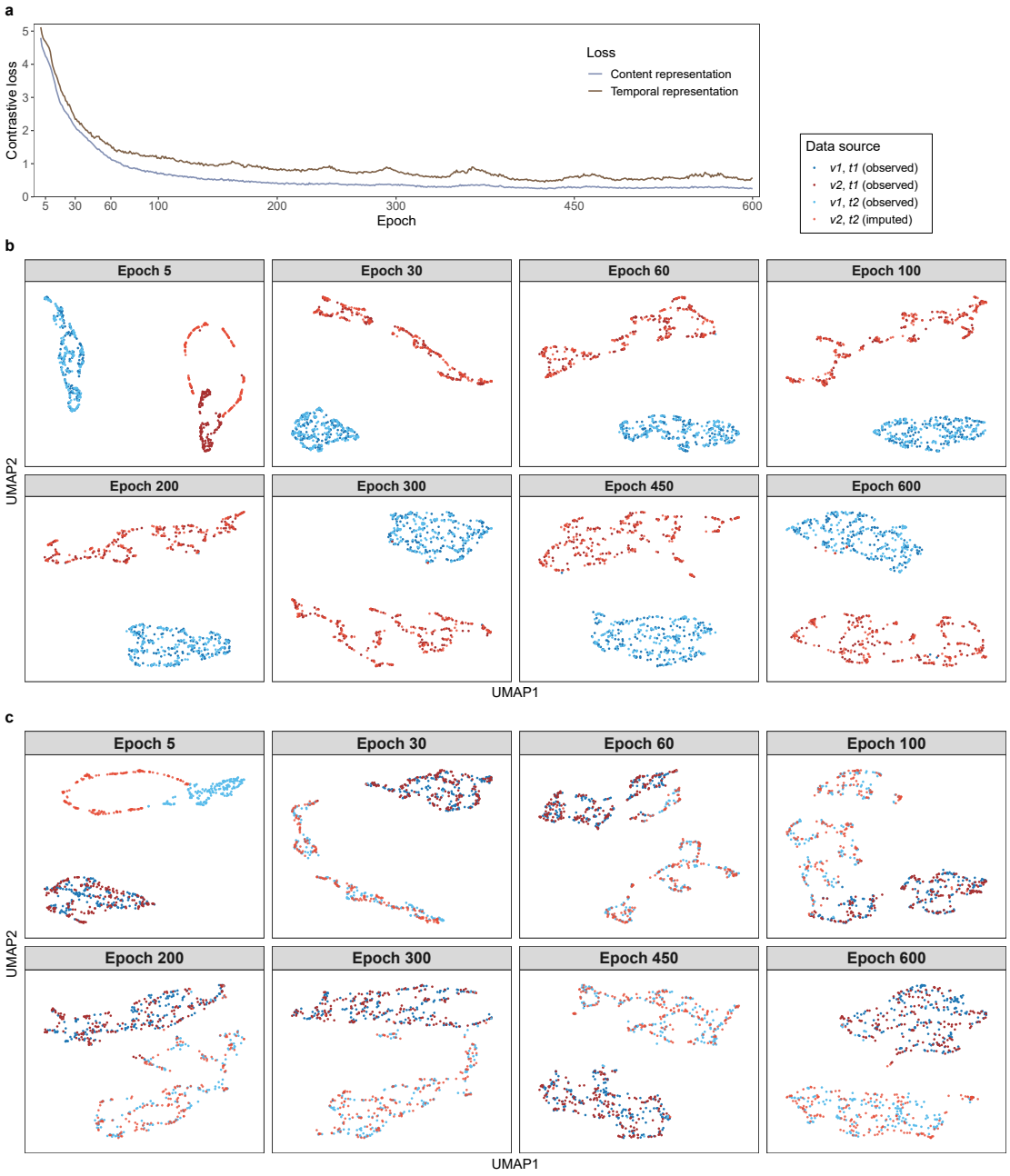

Figure 3.6: Representation Disentanglement Process of LEOPARD. a shows the contrastive loss computed for content and temporal representations. **b-c** show the UMAP embeddings of content (**b**) and temporal (**c**) representations at various training epochs. Representations encoded from data of $v1$ and $v2$ (metabolomics and proteomics, depicted by blue and red dots) at timepoint $t1$ and $t2$ (S4 and F4, depicted by dark- and light-colored dots) are plotted. The data of $v2$ at $t2$ are imputed data produced after each training epoch, while the other data are from the observed samples in the validation set.

data are standardized to ensure all latent variables have a mean of zero and a standard deviation of one. Afterward, two separate UMAP models are built using the R package umap [118], each fitted to the content and temporal representations, with a configuration of `n_neighbors = 15` and `min_dist = 0.1`. Lastly, scatter plots are generated using the R packages ggplot2 [67] and ggsci [68]. Each point in the plot represents an individual sample, and the color indicates the data sources. The visualization epochs are selected experimentally to illustrate the progress of representation disentanglement during the training process.

As the model trains, the contrastive loss decreases (Fig. 3.6a), indicating that LEOPARD is increasingly able to encode the representations for different views and timepoints. The content representation embeddings (Fig. 3.6b) of observed $v1$ (in blue) and $v2$ (in red) separate rapidly during training (epoch 5), but those of the imputed $v2$ for $t2$ (in light red) do not mix with those of the observed $v2$ at $t1$ (in dark red). This suggests that while LEOPARD can distinguish between $v1$ and $v2$ after only a few training epochs, it is not yet capable of producing high-quality $v2$ for $t2$ that have similar content information as the observed $v2$ at $t1$. After 30 epochs of training, the content representation of $v2$ at $t2$ is better encoded, with its embeddings mixing with those of $v2$ at $t1$. Similar trends are observed in the temporal representations (Fig. 3.6c), where embeddings of each timepoint ($t1$ and $t2$ in dark and light colors, respectively) gradually form their corresponding clusters as training progresses. We notice that the temporal representations take more epochs than the content representations to form distinct clusters. Even after 100 epochs, some temporal representation embeddings of $t1$ are still mixed with those of $t2$. However, after around 450 epochs, LEOPARD is demonstrably able to encode temporal information that is unique to $t1$ and $t2$.

3.3.2 Evaluation of Missing View Completion: Mono-Omics Datasets

For the MGH COVID proteomics dataset, missForest overall exhibits the lowest PB, whereas LEOPARD performs slightly worse than missForest and its neural network-based counterpart, cGAN (Fig 3.7, upper row). When compared to LM, GLMM does not show improved performance. As $obsNum$ increases, the PB values of all methods tend to decrease, and the performance gap between LEOPARD and missForest diminishes. Specifically, when $obsNum$ is 100, the UMAP representation (Fig. 3.7, upper row) reveals that the clusters of the imputed data generated by all five methods (green dots) closely approximated the actual data (blue dots), indicating high similarity between the imputed and original datasets.

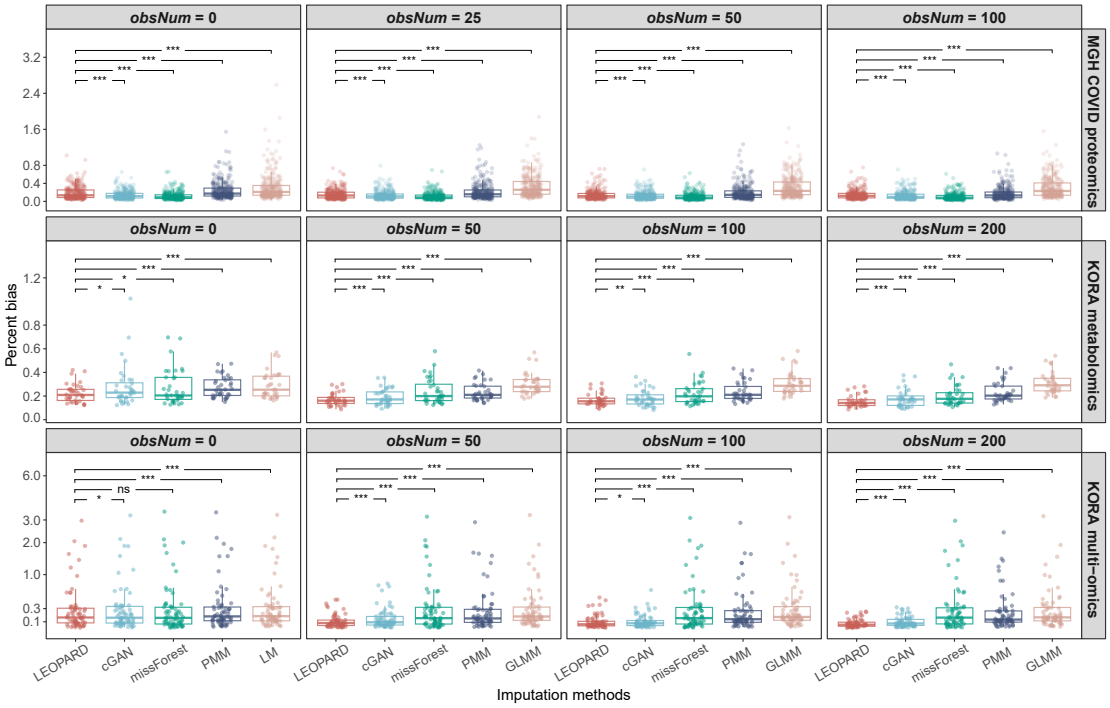

Figure 3.7: Evaluation of Missing View Completion: PB. PB evaluated on $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ of the MGH COVID proteomics dataset (upper row), KORA metabolomics dataset (middle row), and KORA multi-omics dataset (lower row). Each dot represents a PB value for a variable. Please note that LM is used for imputation instead of GLMM when $obsNum = 0$. Significance level: not significant (ns), $P < 0.05$ (*), $P < 0.01$ (**), and $P < 0.001$ (***).

Interestingly, we observe that missForest, despite yielding the best performance for the MGH COVID dataset, produces the most unstable result for the KORA metabolomics dataset, showing the largest InterQuartile Range (IQR) of 0.186 when $obsNum$ is 0 (Fig. 3.7, middle row). In comparison, LEOPARD achieves the smallest IQR of 0.094 under the same condition, while cGAN, PMM, and LM obtain IQR values of 0.125, 0.132, and 0.166, respectively. As $obsNum$ increases to 200, LEOPARD, cGAN, missForest, and PMM lower their median PB values to 0.142, 0.172, 0.177, and 0.204, respectively. However, GLMM obtains a median PB of 0.291 and does not outperform LM. From the UMAP plots generated from the data imputed under $obsNum = 200$, we notice a large amount of the embeddings from $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ generated by missForest and PMM are mixed with those of $\mathcal{D}^{\text{test}}_{v=v2,t=t1}$, instead of $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$, implying that they overfit to $t1$ and do not generalize well to the second timepoint (Fig. 3.8, middle row). Moreover, the UMAP embeddings of $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ from missForest, PMM, and GLMM only partly overlap with

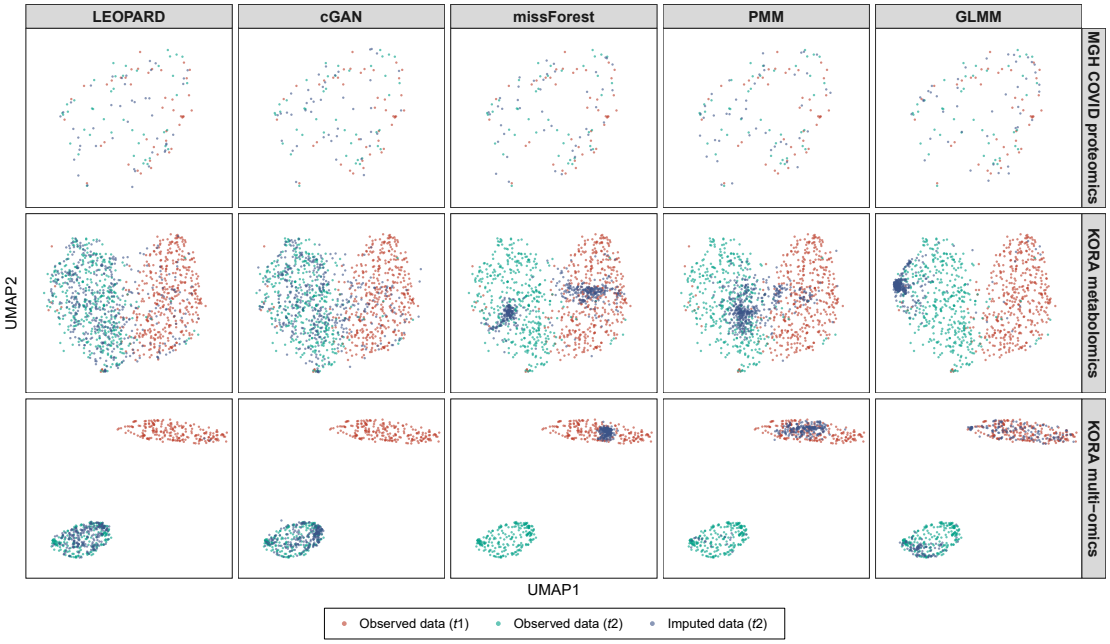

Figure 3.8: Evaluation of Missing View Completion: UMAP. UMAP models are initially fitted with the training data from the MGH COVID proteomics dataset (upper row, $t1$: D0, $t2$: D3), KORA metabolomics dataset (middle row, $t1$: F4, $t2$: FF4), and KORA multi-omics dataset (lower row, $t1$: S4, $t2$: F4). Subsequently, the trained models are applied to the corresponding observed data (represented by red and blue dots for $t1$ and $t2$) and the data imputed by different methods (represented by green dots) under the setting of $obsNum = 100$ for the MGH COVID dataset and $obsNum = 200$ for the two KORA-derived datasets. The distributions of red and blue dots illustrate the variation across the two timepoints, while the similarity between the distributions of blue and green dots indicates the quality of the imputed data. A high degree of similarity suggests a strong resemblance between the imputed and observed data.

those of $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$, suggesting that some variations in the observed data have not been captured. In contrast, the embeddings of the data imputed by LEOPARD widely spread within the embedding space of the observed data, which demonstrates that LEOPARD has effectively learned and approximated the observed data's distribution.

3.3.3 Evaluation of Missing View Completion: Multi-Omics Datasets

In contrast to mono-omics datasets, where both views are from the same omics type, multi-omics datasets require imputation methods to capture more intricate relationships between omics data to ensure accurate results. In our evaluation of the KORA multi-omics data, all methods show some extremely high PB values when $obsNum$ is 0 (Fig. 3.7,

lower row). As $obsNum$ increases to 200, LEOPARD greatly reduces its median PB from 0.152 to 0.061, outperforming its closest competitor, cGAN, which reduces its median PB from 0.158 to 0.076. In contrast, the performances of missForest (from 0.156 to 0.159) and PMM (from 0.177 to 0.131) do not show similar improvements. GLMM reduces its median PB from 0.176 to 0.163 as $obsNum$ increases from 50 to 200. The UMAP visualization further reveals a limited ability of missForest, PMM, and GLMM to capture signals from the $t2$ timepoint, as their embeddings of $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ cluster with $\mathcal{D}^{\text{test}}_{v=v2,t=t1}$, not $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ (Fig. 3.8, lower row). LEOPARD's performance is further validated by a high similarity between the distributions of the imputed and observed data embeddings in the UMAP space.

We also calculated the PB and UMAP of each individual imputation of the multiple imputation methods (PMM, LM, and GLMM). The results can be found in Appendix Figs. A.1 and A.2.

3.3.4 Analysis on Extremely High PB Values

We then investigate the extremely high PB values (> 0.8) observed in the KORA multi-omics dataset. Under $obsNum = 0$, we notice that proteins with low abundance (< 4.0) tend to exhibit extremely high PB in the imputed values (Fig. 3.9). For instance, SCF (stem cell factor), with a median abundance of 9.950, has a PB of 0.090 calculated from the LEOPARD-imputed data. In contrast, NT3 (Neurotrophin-3), with a much lower median abundance of 0.982, shows a PB of 1.187 calculated from the same imputed data. Increasing $obsNum$ can substantially lower these extremely high PB values for LEOPARD and cGAN, but makes no similar contributions for missForest, PMM, and GLMM.

3.3.5 Evaluation on Missing Values in Observed Views

We further evaluate how different methods perform when observed views contain missing values. Our findings indicate that all the methods experienced an increase in PB when the observed views contain missing values (Fig. 3.10). However, LEOPARD and missForest are robust to the missing data points in terms of PB. In contrast, cGAN and GLMM exhibit high sensitivity to those missing values. Method cGAN does not show similar improvement with the increase of $obsNum$ as it performs in Fig. 3.7 (middle row) and is gradually surpassed by missForest. GLMM overall exhibits higher PB than the other methods.

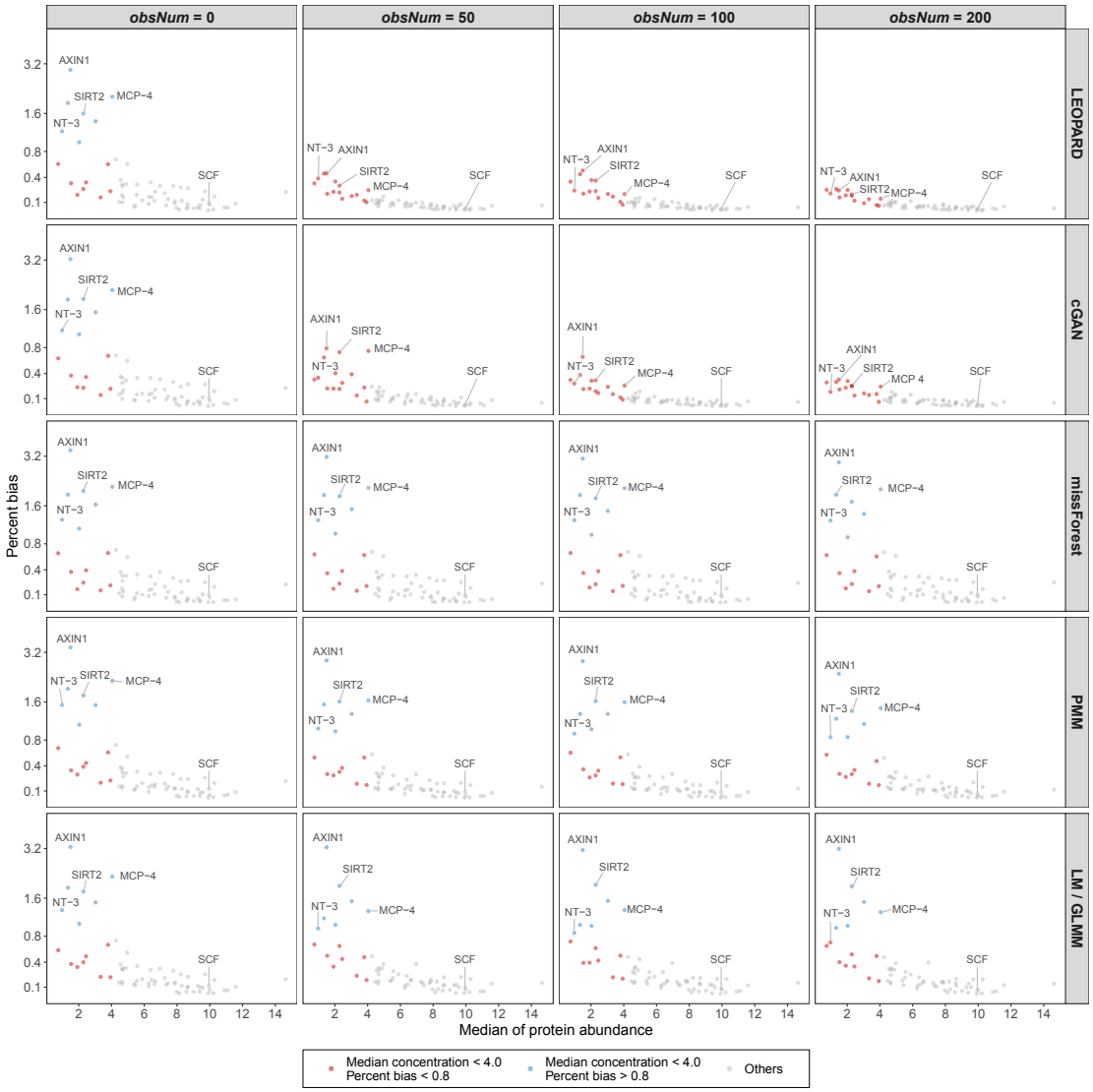

Figure 3.9: Analysis on the Variables with High PB. The proteins with low abundance (median concentration < 4.0) tend to exhibit extremely high PB (> 0.8) in the imputed values obtained under $obsNum = 0$. The extremely high PB values of LEOPARD can be lowered by increasing $obsNum$. Please note that LM is used for imputation instead of GLMM when $obsNum = 0$.

The UMAP plots (Fig. 3.11) further demonstrate that LEOPARD's performance remains comparable to scenarios with no missing data in the observed views (Fig. 3.8, middle row), unlike the other methods which display overfitting or a great loss of data variation. Although LEOPARD outperforms other methods, we observed a change in the distribution of the imputed data (blue dots): as *maskObs* increases, these blue dots begin to shrink toward their center and become more concentrated. This leads to a reduced coverage of the outer areas of the ground truth embeddings (green dots) and suggests that the imputed data might not capture the full variability of the data when the proportion of missing data is high.

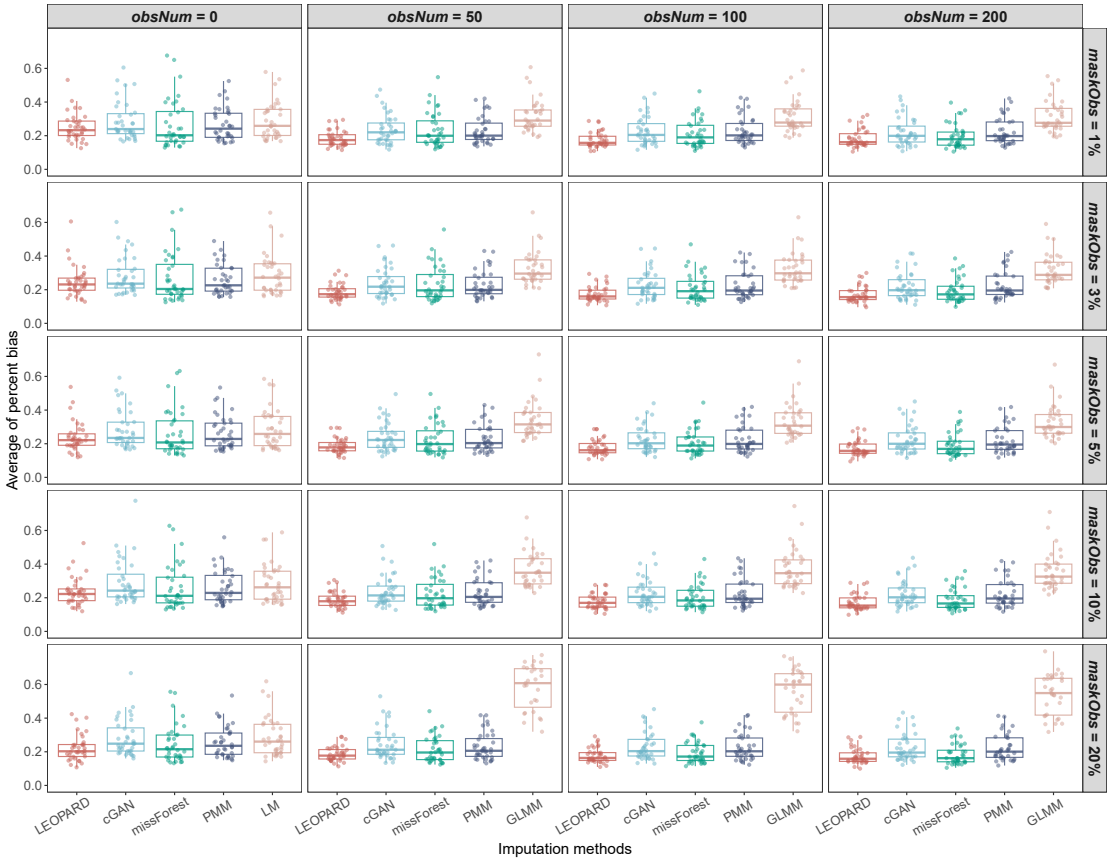

Figure 3.10: Evaluation on Missing View Completion with Missing Values: PB. The average PB are computed for $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ of the KORA metabolomics dataset across 10 repeated completions, under each proportion of masked data points in the observed views (*maskObs*). In each repetition, the data points are masked randomly. Each dot represents a PB value for a variable. Please note that LM is used instead of GLMM when $obsNum = 0$.

Figure 3.11: Evaluation on Missing View Completion with Missing Values: UMAP. UMAP models are initially fitted with the training data from the KORA multi-omics dataset ($t1$: S4, $t2$: F4). Subsequently, the trained models are applied to the corresponding observed data (represented by red and blue dots for $t1$ and $t2$) and the data imputed by different methods (represented by green dots) under $obsNum$ = 200 and varying $maskObs$. Please note that, for each $maskObs$, only the repetition that exhibits the lowest median of PB are visualized. The distributions of red and blue dots illustrate the variation across the two timepoints, while the similarity between the distributions of blue and green dots indicates the quality of the imputed data. A high degree of similarity suggests a strong resemblance between the imputed and observed data.

3.4 Case Studies

Since evaluation metrics can sometimes oversimplify the assessment of imputed data quality, we further perform several case studies, covering both regression and classification tasks, to investigate whether biological signals are preserved in the imputed data obtained at $obsNum = 0$.

3.4.1 Regression analysis

The regression models are fitted using the observed data and different imputed data corresponding to $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$. We employ multivariate linear regression models for each of the observed or imputed data. The imputed data of LEOPARD, cGAN, missForest, PMM, and LM are obtained under the setting of $obsNum = 0$. The performance of LEOPARD, cGAN, and missForest are evaluated on their imputed data directly, while two multiple imputation methods, PMM and LM, are evaluated by pooling their multiple

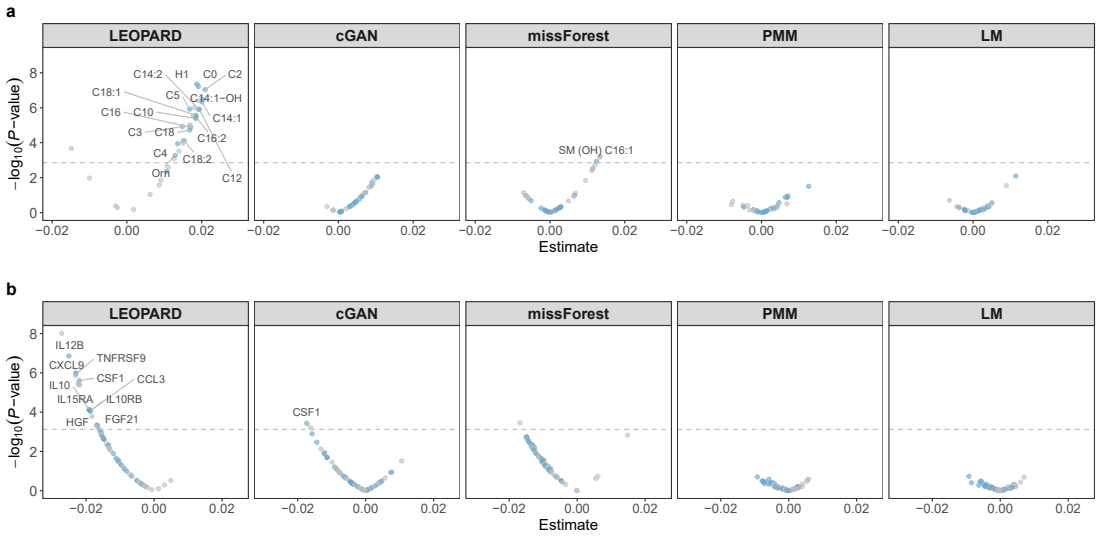

Figure 3.12: Volcano Plots for Regression Analyses. **a**, Volcano plots display age-associated metabolites detected in the $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ and $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ ($obsNum = 0$) of the KORA metabolomics dataset ($N = 417$). 18 significant metabolites (P-value $< 0.05/36$) identified in the observed data are shown in blue. Replicated metabolites from the data imputed by different methods are marked with labels. **a**, Volcano plots display eGFR-associated proteins detected in the $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ and $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ ($obsNum = 0$) of the KORA multi-omics dataset ($N = 212$). 28 significant metabolites (P-value $< 0.05/66$) identified in the observed data are shown in blue. Replicated metabolites from the data imputed by different methods are marked with labels.

estimates using Rubin's rules [56]. In both analyses, we apply a Bonferroni correction to adjust the P-value significance threshold to mitigate the risk of false positives in multiple testing.

For the KORA metabolomics dataset ($N = 417$), we use the concentration of each metabolite as the response variable and age as the predictor variable, while controlling for sex, to detect age-associated metabolites. Of the 18 metabolites significantly associated with age after a Bonferroni correction for multiple testing ($P < 0.05/36$) in the observed data, 17 are also significant in the data imputed by LEOPARD (see Fig. 3.12a). Among these 17 metabolites, several, including C14:1 (Tetradecenoylcarnitine), C18 (Octadecanoylcarnitine), C18:1 (Octadecenoylcarnitine), and Orn (Ornithine), have been validated by previous research [119, 58, 120, 59] showing that they might be particularly relevant in aging and age-related metabolic conditions. In contrast, only one metabolite is significantly associated with age in the data imputed by missForest. No metabolite is identified as significant in the data imputed by cGAN, PMM, and LM. The results on each imputation of PMM and LM are shown in Appendix Fig. A.3.

We then use the KORA multi-omics dataset ($N = 212$) to identify proteins associated with the estimated Glomerular Filtration Rate (eGFR), controlling for age and sex. The eGFR values are computed from serum creatinine, sex, race, and age, using the Chronic Kidney Disease Epidemiology Collaboration equation [121]. The NPX values of each protein are used as the response, while the eGFR values as the predictor. In the observed data, 28 proteins are significantly associated with eGFR after a Bonferroni correction ($P < 0.05/66$). Of these 28 proteins, 10 proteins remain significant in the data imputed by LEOPARD (see Fig 3.12b), while one is significant in the data from cGAN, and none is identified as significant in the data from missForest, PMM, and LM. Among the 10 proteins detected from the LEOPARD-imputed data, eight (TNFRSF9, IL10RB, CSF1, FGF21, HGF, IL10, CXCL9, and IL12B) have been validated by prior research [4]. The results on each imputation of PMM and LM are shown in Appendix Fig. A.4.

3.4.2 Classification analysis

We further train Balanced Random Forest (BRF) [122] models using the Python library imbalanced-learn [123] to predict Chronic Kidney Disease (CKD). The BRF is specifically selected to address the dataset imbalance and reduce the risk of overfitting to the majority class. The BRF models are individually fitted using the observed and different imputed data of $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ obtained under the setting of $obsNum = 0$, corresponding to

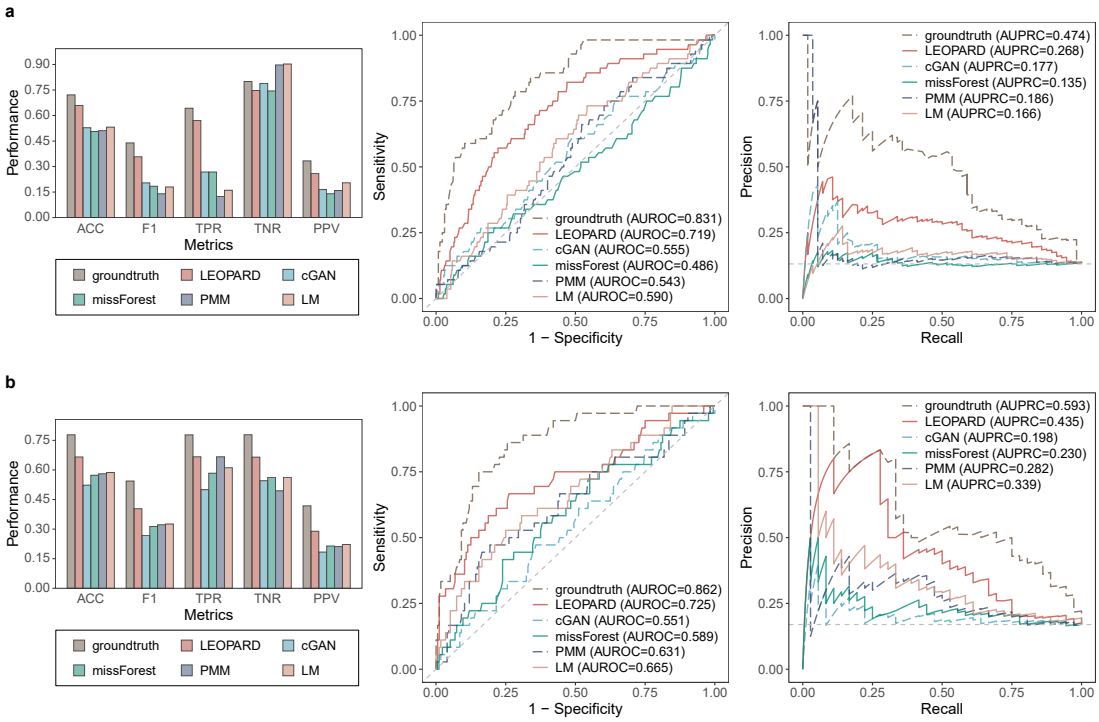

Figure 3.13: Multi-Metric Evaluation for Classification Analyses. CKD classification evaluated using $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ and $\hat{\mathcal{D}}^{\text{test}}_{v=v2,t=t2}$ ($obsNum = 0$) from (**a**) the KORA metabolomics dataset ($N = 416$, $N_{positive} = 56$, $N_{negative} = 360$) and (**b**) the KORA multi-omics dataset ($N = 212$, $N_{positive} = 36$, $N_{negative} = 176$). Models are trained using the BRF algorithm with identical hyperparameters and evaluated using LOOCV. Evaluation metrics in the bar plot include true positive rate (TPR, also known as sensitivity), true negative rate (TNR, also known as specificity), positive predictive value (PPV, also known as precision), accuracy (ACC), and F1 score. The dashed lines in the ROC and PR curves represent the performance of a hypothetical model with no predictive capability.

36 metabolites from the KORA metabolomics dataset ($N = 416$, one sample removed due to a missing CKD label) and 66 proteins from the KORA multi-omics dataset ($N = 212$). CKD cases are defined as having an eGFR < 60 mL/min/1.73m^2 [124]. In the two datasets, 56 and 36 individuals are identified as CKD cases, respectively.

Due to the limited sample size, we validate the performance using the LOOCV strategy, allowing maximal use of data for both training and validation. The models for LEOPARD, cGAN, and missForest are trained using their respective imputed data, while the models for PMM and LM are trained on the average estimates across their multiple imputations. All models are trained with default hyperparameters (`criterion = "gini"`, `min_samples_split = 2`, `min_samples_leaf = 1`, `max_features = "sqrt"`,

Table 3.3: Case Study: CKD Prediction on KORA Metabolomics Dataset

Method	TPR	TNR	PPV	ACC	F1 Score	AUROC	PRRPC
observation	0.643	0.800	0.333	0.721	0.439	0.831	0.474
PMM	0.125	**0.897**	0.159	0.511	0.140	0.543	0.186
LM	0.161	0.903	0.205	0.532	0.180	0.590	0.166
missForest	0.268	0.744	0.140	0.506	0.184	0.486	0.135
cGAN	0.268	0.789	0.165	0.528	0.204	0.555	0.177
LEOPARD	**0.571**	0.747	**0.260**	**0.659**	**0.358**	**0.719**	**0.268**

Bold indicates the highest performance among imputation methods.

`bootstrap = True`), except that `n_estimators` is set to `1000` and `class_weight` is set to `"balanced_subsample"`.

Performance metrics were calculated using the R package caret [61]. These metrics provided a comprehensive understanding of the predictive power of the observed and imputed data. The ROC curves are plotted to illustrate the trade-off between sensitivity and 1-specificity at varying decision thresholds. Considering the imbalance in our dataset, and with our primary interest in the positive class, which is also the minority, we further plot PR curves to depict the trade-off between precision and recall at different thresholds for the classifiers trained with the different data. For PR curves, the baseline performance of a non-discriminative model is determined by the proportion of positive cases ($56/416 = 0.135$ for the KORA metabolomics dataset and $36/212 = 0.170$ for the KORA multi-omics dataset). Both the ROC and PR curves are plotted using the R package precrec [125].

For the KORA metabolomics dataset, the observed data obtain an F1 Score of 0.439, and the data imputed by LEOPARD achieves the closest performance with an F1 Score of 0.358 (Fig. 3.13a, Table 3.3). LEOPARD also outperforms its competitors in terms of accuracy, sensitivity, precision, Area Under the Receiver Operating Characteristic curve (AUROC), and Area Under the Precision-Recall Curve (AUPRC). The proteins from the KORA multi-omics dataset perform better than the metabolites from the metabolomics dataset for this task. The F1 Score increases to 0.544 for the observed data of the KORA multi-omics dataset. LEOPARD outperforms its competitors with an F1 Score of 0.403, an AUROC of 0.725, and an AUPRC of 0.435 (Fig. 3.13b, Table 3.4). The prediction results on each individual imputation of PMM and LMM are displayed in Appendix Figs. A.5 and A.6.

Table 3.4: Case Study: CKD Prediction on KORA Multi-Omics Dataset

Method	TPR	TNR	PPV	ACC	F1 Score	AUROC	PRRPC
observation	0.778	0.778	0.418	0.778	0.544	0.862	0.593
PMM	**0.667**	0.494	0.212	0.580	0.322	0.631	0.282
LM	0.611	0.562	0.222	0.587	0.326	0.665	0.339
missForest	0.583	0.562	0.214	0.573	0.313	0.589	0.230
cGAN	0.500	0.545	0.184	0.523	0.269	0.551	0.198
LEOPARD	**0.667**	**0.665**	**0.289**	**0.666**	**0.403**	**0.725**	**0.435**

Bold indicates the highest performance among imputation methods.

3.5 Applicability Analysis

We next explore how many training samples are required for LEOPARD to have robust view completion and assess the performance of LEOPARD in completing different missing views at different timepoints. These analyses reveal LEOPARD's utility and adaptability in different analytical scenarios.

3.5.1 Minimum Training Samples Required for Robust Results

The evaluation is performed on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ of our three benchmark datasets by varying the number of training samples from 20 to 160 and $obsNum$ from 0 to 50. Each condition is tested 10 times with different samples randomly selected from the training sets. The performance is evaluated by PB averaged across these repetitions. Fig. 3.14 simplifies the boxplot and shows the median and the IQR of the averaged PB values calculated for the variables in the imputed data.

Across all datasets, average PB generally decreases with more training samples, indicating an improvement in view completion. Consistent with our previous evaluation, PB values decrease as $obsNum$ increases. Additionally, we notice that the average PB steadily decreases for the MGH COVID proteomics dataset, which exhibits the smallest variation between the two timepoints in our UMAP plots (Fig. 3.8). In contrast, the average PB for the other two datasets shows some fluctuations, particularly for the KORA multi-omics dataset, which shows the most obvious variation between two timepoints. When $obsNum = 0$, the MGH COVID proteomics and the KORA metabolomics datasets require about 120 training samples to obtain stable results; the KORA multi-omics dataset, however, exhibits a wide range of PB under this condition. When we increase $obsNum$ to 20, the performance stabilizes with approximately 60 to 80 samples

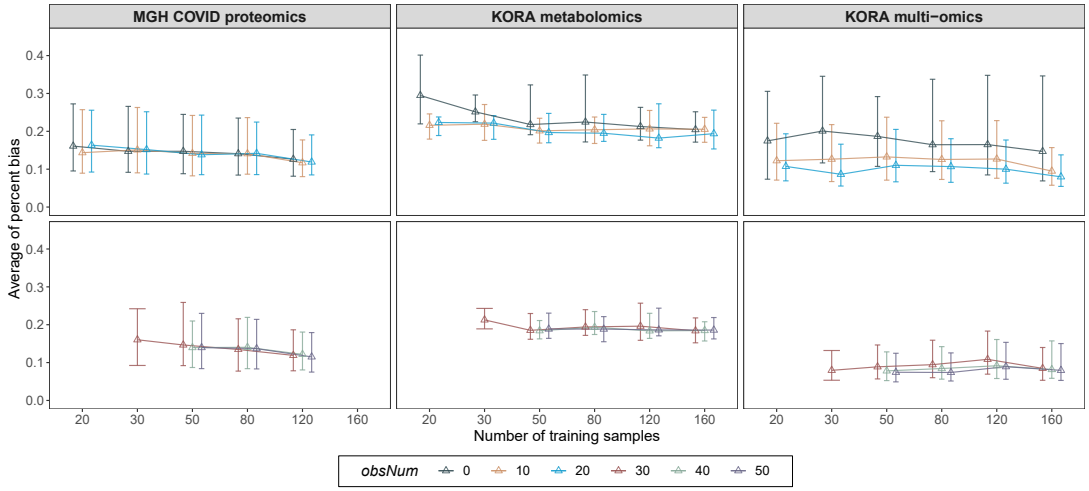

Figure 3.14: Analysis on the Minimum Number of Training Samples. For each benchmark dataset, the average PB is evaluated on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ across 10 repeated completions for each combination of training sample sizes and $obsNum$. The bar indicates the median and the IQR of the average PB values for different variables. In each repetition, the samples are selected randomly. Please note that the maximum $obsNum$ cannot exceed the number of training samples, and the full training set of the MGH COVID proteomics dataset contains only 140 samples.

used for training LEOPARD. Based on our evaluation, at least 80 training samples may be required for robust view completion.

3.5.2 Arbitrary Temporal Knowledge Transfer

In the previous evaluation, we assess the performance of each method on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ from the benchmark datasets. We then extend our analysis by evaluating LEOPARD's performance on individually masked test sets: $\mathcal{D}^{\text{test}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{test}}_{v=v1,t=t2}$, and $\mathcal{D}^{\text{test}}_{v=v2,t=t1}$ from the KORA metabolomics dataset. This approach allows us to assess LEOPARD's capability to arbitrarily complete any views at any timepoints within this dataset. Moreover, LEOPARD is evaluated using the expanded KORA metabolomics dataset to complete the masked test set $\mathcal{D}^{\text{test}}_{v=v2,t=t3}$, using the training data from $\mathcal{D}^{\text{train}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{train}}_{v=v2,t=t1}$, and $\mathcal{D}^{\text{train}}_{v=v1,t=t3}$. This analysis further enables us to explore LEOPARD's capability to complete views across a long time span. LEOPARD is trained using the same hyperparameters as we used in the previous experiments.

Our results reveal that some metabolites exhibit high PB values at $obsNum = 0$ (Fig. 3.15) due to their low concentrations. While different completions show variability in their performances, PB generally decreases as $obsNum$ increases. This evaluation

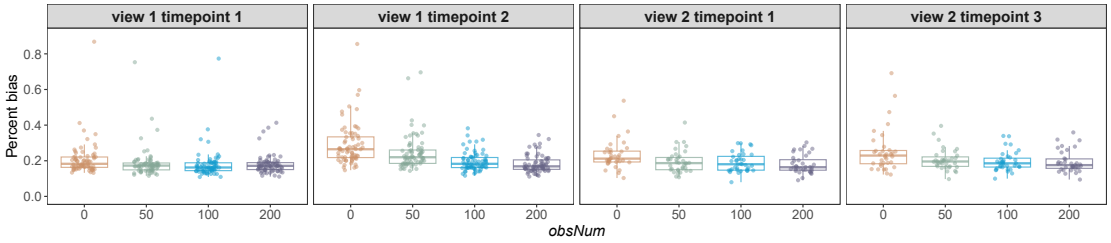

Figure 3.15: Analysis on Arbitrary Style Transfer. The first three panels illustrate LEOPARD's performance evaluated on $\mathcal{D}^{\text{test}}_{v=v1,t=t1}$, $\mathcal{D}^{\text{test}}_{v=v1,t=t2}$, and $\mathcal{D}^{\text{test}}_{v=v2,t=t1}$ of the KORA metabolomics dataset, with varying $obsNum$. The fourth panel displays LEOPARD's performance on the extended KORA metabolomics dataset, where it completes the missing view $\mathcal{D}^{\text{test}}_{v=v2,t=t3}$ using the training data at $t1$. Each dot represents a PB value for a variable.

demonstrates that LEOPARD can transfer extracted temporal knowledge to different content representations in a flexible and generalized way. However, additional observations from the incomplete view may be necessary to ensure robust results, particularly for metabolites with low concentrations.

3.6 Discussion

We developed LEOPARD, a novel architecture for missing view completion designed for multi-timepoint omics data. The performance of LEOPARD was comprehensively assessed through simulations and case studies using three real human omics datasets. Additional interesting findings emerged from our evaluation.

As illustrated in the UMAP plots, the MGH COVID proteomics data (Fig. 3.8 upper row) from D0 ($t1$, in red) and D3 ($t2$, in blue) show relatively low variation, while the KORA metabolomics data (Fig. 3.8 middle row) from F4 ($t1$, in red) and FF4 ($t2$, in blue) exhibit more substantial variations, potentially due to biological variations spanning seven years and technical variation from different analytical kits in the KORA data. In the MGH COVID dataset, we observed that LEOPARD performs slightly worse than its competitors. This can be attributed to the inherent differences in the representation learning process of LEOPARD and its competitors. By directly learning mappings between views, the methods developed for cross-sectional data can exploit the input data to learn detailed, sample-specific patterns, while the representation learning in LEOPARD primarily focuses on more compact and generalized structures related to views and timepoints, potentially neglecting detailed information specific to individual samples. The MGH COVID dataset, having a high similarity between the data from D0

and D3, allows the cross-sectional imputation methods to effectively apply the mappings learned from one timepoint to another. As the data variation between two timepoints increases, LEOPARD's advantages become increasingly evident, while its competitors tend to overfit the training data and fail to generalize well to the new timepoint (Fig. 3.8, middle and lower rows).

For the KORA multi-omics dataset, we observed that the extremely high PB values are associated with low analyte abundances. For the same absolute error in imputed values, a variable with a low analyte abundance will have a higher absolute error ratio, leading to a higher PB than those variables with high abundances. Additionally, protein levels quantified using the Olink platform are represented as relative quantities. Data from different measurements also contain technical variations that arise from experimental factors and normalization methods used for relative quantification. When $obsNum$ is 0, LEOPARD is trained without any data from $v2$ at $t2$, and thus cannot account for the technical variations exclusive to that part. By incorporating a few observed samples from the second timepoint into the training process, the model can better capture the data distribution and technical variation of the missing part, which contributes to a substantial reduction in high PB values.

As data imputation inevitably incurs a loss of information, we conducted case studies to assess the preservation of biological information in the imputed data. Despite all five imputation methods producing similar PB when $obsNum$ is 0 (Fig. 3.7 lower row), the case studies showed that the data imputed by LEOPARD provided performance closest to the observed data, while the data imputed by cGAN, missForest, PMM, and LM showed a substantial loss of biological information (Figs. 3.12 and 3.13). This outcome highlights the importance of case studies for a reliable evaluation of imputed data.

Arbitrary style transfer, a concept from the computer vision field underpinning LEOPARD, allows the style of one image to be transferred to the content of another. This study demonstrates that LEOPARD inherits this capability and has the potential for arbitrary temporal knowledge transfer. Our experiments also demonstrate that LEOPARD can yield robust results with approximately 80 samples. Moreover, LEOPARD not only completes missing views for downstream analyses, but also facilitates the exploration of temporal dynamics. By analyzing the extracted temporal embeddings, LEOPARD could enable the inference of the temporal ordering of omics changes, which would be particularly valuable when there is a discrepancy between biological and chronological order. As the number of data timepoints increases, LEOPARD is expected to offer new opportunities in predictive healthcare with multi-timepoint omics data.

While LEOPARD demonstrates superior performance over existing generic imputation methods on missing view completion, it is important to consider the limitations and caveats of this study. To align with real-world settings, we defined QC criteria based on existing studies when constructing our benchmark datasets, and consequently, only the most detectable proteins and metabolites were selected. This could inflate the metrics of both LEOPARD and other methods reported in this study. The performance on these selected variables may not accurately reflect that of the overall proteins and metabolites, especially those showing more variability in their abundance. LEOPARD typically requires observed views to be complete so that temporal and content representations can be extracted. Considering the common occurrence of missing values in real-world omics data, LEOPARD is designed to tolerate a small proportion while maintaining optimal robustness. However, we observed that LEOPARD struggles to capture the full diversity present in the ground truth as *maskObs* increases to 20% (Fig. 3.11). It is preferable for the input data for LEOPARD to contain less than 10% missing data points. Higher proportions of missing values are ideally addressed by generic imputation methods before processing with LEOPARD. Additionally, we assumed that the missing data were MCAR. Additional bias could be introduced if data points are MAR or MNAR in real-world scenarios. Finally, our experiments were restricted by data availability to three timepoints, but in principle LEOPARD can accommodate additional timepoints and is well-suited for analyses involving multiple omics per timepoint.

With advancements in omics measurement technology and the growing availability of longitudinal data, missing view in multi-view, multi-timepoint data is becoming a prominent issue. Our study demonstrates that established generic methods, originally developed for missing data points or cross-sectional data, do not produce robust results in this new context. This highlights the necessity for specialized methods, and our method, LEOPARD, represents an early attempt to address this issue. We anticipate further developments in imputation methods that exhibit high generalization ability, robustness to low analyte abundance, and preservation of biological variations.

3.7 Reproducibility and Availability

Similar to the previous chapter, we have ensured the reproducibility and accessibility of this computational method by making all relevant files publicly available. Additionally, we provide a script to reproduce the figures presented in this study to facilitate detailed performance evaluation.

3.7.1 Data

The MGH COVID study procedures were approved by the Mass General Brigham (formerly Partners) Human Research Committee, the governing institutional review board at Massachusetts General Hospital. The proteomics data, published by the original authors, are freely available for investigators from Mendeley Data (`http://dx.doi.org/10.17632/nf853r8xsj`). The MGH COVID proteomics dataset constructed in this study is available at our GitHub repository (`https://github.com/HAN-Siyu/LEOPARD`).

The protocol of the KORA study was approved by the Ethics Committee of the Bavarian Chamber of Physicians. All study participants provided written informed consent. The KORA data are governed by the General Data Protection Regulation and national data protection laws, with additional restrictions imposed by the Ethics Committee of the Bavarian Chamber of Physicians to ensure data privacy of the study participants. Therefore, the data cannot be made freely available in a public repository. However, researchers with a legitimate interest in accessing the data may submit a request through an individual project agreement with KORA via the online portal (`https://www.helmholtz-munich.de/en/epi/cohort/kora`). Upon receipt of the request, the data access committee will review the application and, subject to approval, provide the researcher with a data usage agreement.

3.7.2 Python Package

The source code and implementation details of LEOPARD are freely available at our GitHub repository (`https://github.com/HAN-Siyu/LEOPARD`). Detailed documentation and examples can be found in the package manual (Appendix D).

3.7.3 Reproducible Figures

Relevant data and code can be found in a separate script in the jupyter-notebook format to reproduce the main figures in this study. The script can be downloaded from our GitHub repository (`https://github.com/HAN-Siyu/LEOPARD`).

4 Conclusion and Outlook

In this thesis, two novel computational methods, TIGER and LEOPARD, were developed to improve the data reliability and completeness for multi-timepoint omics. This chapter will summarize the contributions and discuss some possibilities for future work.

TIGER was specifically designed for the removal of technical variation in metabolomics data, leveraging an ensemble learning architecture to enhance robustness and mitigate overfitting. This method demonstrated superior performance over four widely-used techniques, effectively preserving biological signals while reducing technical noise. Now TIGER has been used in both cross-sectional and longitudinal biological analyses, such as biomarker discovery for metabolic syndrome [93] and standardized implementation of batch correction [126]. The successful application of TIGER underscores its potential in revealing true biological variations that might otherwise be obscured by technical variations.

LEOPARD, on the other hand, was developed to address the problem of missing view in multi-timepoint omics datasets. Through comprehensive simulations and case studies, LEOPARD showcased its ability to disentangle temporal and content representations, thereby providing insights for investigating temporal dynamics of multi-timepoint omics data. The method's robustness was further validated across various datasets, revealing its robust performance compared to generic imputation techniques.

Both methods were comprehensively evaluated using multiple metrics to ensure their reliability and effectiveness. The reproducibility and accessibility of these methods were ensured by making relevant data, code, and documentation publicly available. By providing comprehensive scripts to reproduce the figures and results, this work facilitates further research and application in the field.

Future research could focus on expanding the application scope of TIGER and LEOPARD:

- TIGER offers greater flexibility than existing tools by supporting different types of QC samples, utilizing injection order and well position for data modeling, and enabling parallel computing. However, as a QC-based method, TIGER's applica-

bility is limited by the availability of QC samples. Developing a QC-free method would be highly beneficial for this field.

- LEOPARD, representing the first attempts for missing view completion in multi-timepoint omics data, shows promise but may not be suitable for datasets with very small sample sizes due to its neural network-based approach. Enhancing LEOPARD's applicability to small datasets by developing strategies to maintain robustness and accuracy with limited data would increase its versatility.

- Expanding the application of TIGER and LEOPARD beyond metabolomics and proteomics to other types of omics data, such as lipidomics and transcriptomics, could provide significant benefits for omics researchers, offering a more comprehensive toolkit for analyzing complex biological systems.

Overall, TIGER and LEOPARD offer effective solutions for addressing technical variations and missing view in multi-timepoint omics analysis. The methodologies presented in this thesis provide valuable insights for future innovations in omics data analysis. The robust performance of TIGER and LEOPARD highlights their potential to enhance omics research and healthcare applications.

Bibliography

[1] R. Holle, M. Happich, et al. KORA - A research platform for population based health research. *Gesundheitswesen*, 67(SUPPL. 1), 2005. doi:10.1055/s-2005-858235.

[2] M. Haid, C. Muschet, et al. Long-Term Stability of Human Plasma Metabolites during Storage at −80 °C. *Journal of Proteome Research*, 17(1):203–211, jan 2018. doi:10.1021/acs.jproteome.7b00518.

[3] Z. Yu, G. Kastenmüller, et al. Differences between human plasma and serum metabolite profiles. *PloS one*, 6(7):e21230, 2011.

[4] C. Herder, J. M. Kannenberg, et al. A Systemic Inflammatory Signature Reflecting Cross Talk Between Innate and Adaptive Immunity Is Associated With Incident Polyneuropathy: KORA F4/FF4 Study. *Diabetes*, 67(11):2434–2442, nov 2018. URL: https://diabetesjournals.org/diabetes/article/67/11/2434/40270/A-Systemic-Inflammatory-Signature-Reflecting-Cross, doi:10.2337/db18-0060.

[5] M. Lundberg, A. Eriksson, et al. Homogeneous antibody-based proximity extension assays provide sensitive and specific detection of low-abundant proteins in human blood. *Nucleic acids research*, 39(15):e102–e102, 2011.

[6] L. Tran, X. Liu, et al. Missing Modalities Imputation via Cascaded Residual Autoencoder. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4971–4980. IEEE, jul 2017. URL: http://ieeexplore.ieee.org/document/8100011/, doi:10.1109/CVPR.2017.528.

[7] L. Cai, Z. Wang, et al. Deep Adversarial Learning for Multi-Modality Missing Data Completion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, pages 1158–1166, New York, NY, USA, jul 2018. ACM. doi:10.1145/3219819.3219963.

[8] V. Swamy, M. Satayeva, et al. MultiModN- Multimodal, Multi-Task, Interpretable Modular Networks. sep 2023. URL: `http://arxiv.org/abs/2309.14118`, `arXiv:2309.14118`.

[9] S. Li, Y. Park, et al. Predicting network activity from high throughput metabolomics. *PLoS Comput Biol*, 9(7):e1003123, 2013.

[10] D. S. Wishart. Emerging applications of metabolomics in drug discovery and precision medicine. *Nature reviews Drug discovery*, 15(7):473, 2016.

[11] P. Sen, S. Lamichhane, et al. Deep learning meets metabolomics: A methodological perspective. *Briefings in Bioinformatics*, 22(2):1531–1542, 2021.

[12] C. H. Johnson, J. Ivanisevic, and G. Siuzdak. Metabolomics: beyond biomarkers and towards mechanisms. *Nature reviews Molecular cell biology*, 17(7):451–459, 2016.

[13] Z. Pang, J. Chong, et al. MetaboAnalystR 3.0: Toward an optimized workflow for global metabolomics. *Metabolites*, 10(5):186, 2020.

[14] P. L. Auer and R. W. Doerge. Statistical design and analysis of RNA sequencing data. *Genetics*, 185(2):405–416, 2010.

[15] S. C. Hicks and R. A. Irizarry. quantro: A data-driven approach to guide the choice of an appropriate normalization method. *Genome Biology*, 16(1):1–8, 2015. URL: `http://dx.doi.org/10.1186/s13059-015-0679-0`, `doi:10.1186/s13059-015-0679-0`.

[16] A. M. De Livera, D. A. Dias, et al. Normalizing and integrating metabolomics data. *Analytical chemistry*, 84(24):10768–10776, 2012.

[17] J. Kuligowski, Á. Sánchez-Illana, et al. Intra-batch effect correction in liquid chromatography-mass spectrometry using quality control samples and support vector regression (QC-SVRC). *Analyst*, 140(22):7810–7817, 2015.

[18] A. O. Tokareva, V. V. Chagovets, et al. Normalization methods for reducing interbatch effect without quality control samples in liquid chromatography-mass spectrometry-based studies. *Analytical and Bioanalytical Chemistry*, 413(13):3479–3486, 2021.

[19] A. Scherer. *Batch effects and noise in microarray experiments: sources and solutions*, volume 868. John Wiley & Sons, 2009.

[20] R. Wehrens, J. A. Hageman, et al. Improved batch correction in untargeted MS-based metabolomics. *Metabolomics*, 12(5):88, 2016.

[21] W. Wang, H. Zhou, et al. Quantification of proteins and metabolites by mass spectrometry without isotopic labeling or spiked standards. *Analytical chemistry*, 75(18):4818–4826, 2003.

[22] J. Huang, C. Huth, et al. Machine learning approaches reveal metabolic signatures of incident chronic kidney disease in individuals with prediabetes and type 2 diabetes. *Diabetes*, 69(12):2756–2765, 2020. doi:10.2337/db20-0586.

[23] B. Li, J. Tang, et al. Performance evaluation and online realization of data-driven normalization methods used in LC/MS based untargeted metabolomics analysis. *Scientific reports*, 6(1):1–13, 2016.

[24] W. S. Cleveland and S. J. Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403):596–610, 1988.

[25] S. Fan, T. Kind, et al. Systematic Error Removal Using Random Forest for Normalizing Large-Scale Untargeted Lipidomics Data. *Analytical Chemistry*, 91(5):3590–3596, 2019. doi:10.1021/acs.analchem.8b05592.

[26] K. Deng, F. Zhang, et al. WaveICA: A novel algorithm to remove batch effects for large-scale untargeted metabolomics data based on wavelet analysis. *Analytica chimica acta*, 1061:60–69, 2019.

[27] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[28] M. Sysi-Aho, M. Katajamaa, et al. Normalization method for metabolomics data using optimal selection of multiple internal standards. *BMC bioinformatics*, 8(1):1–17, 2007.

[29] C. Workman, L. J. Jensen, et al. A new non-linear normalization method for reducing variability in DNA microarray experiments. *Genome biology*, 3(9):1–16, 2002.

[30] H. Luan, F. Ji, et al. statTarget: A streamlined tool for signal drift correction and interpretations of quantitative mass spectrometry-based omics data. *Analytica chimica acta*, 1036:66–72, 2018.

[31] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, 1990. URL: `http://ieeexplore.ieee.org/document/57199/`, `doi:10.1109/18.57199`.

[32] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[33] E. Renard, S. Branders, and P.-A. Absil. Independent component analysis to remove batch effects from merged microarray datasets. In *International Workshop on Algorithms in Bioinformatics*, pages 281–292. Springer, 2016.

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

[35] W. B. Dunn, D. Broadhurst, et al. Procedures for large-scale metabolic profiling of serum and plasma using gas chromatography and liquid chromatography coupled to mass spectrometry. *Nature protocols*, 6(7):1060–1083, 2011.

[36] A. M. D. Livera, M. Sysi-Aho, et al. Statistical methods for handling unwanted variation in metabolomics data. *Analytical chemistry*, 87(7):3606–3615, 2015.

[37] C. Brunius, L. Shi, and R. Landberg. Large-scale untargeted LC-MS metabolomics data correction using between-batch feature alignment and cluster-based within-batch signal intensity drift correction. *Metabolomics*, 12(11):1–13, 2016.

[38] B. Li, J. Tang, et al. NOREVA: normalization and evaluation of MS-based metabolomics data. *Nucleic acids research*, 45(W1):W162—-W170, 2017.

[39] A. P. Siskos, P. Jain, et al. Interlaboratory reproducibility of a targeted metabolomics platform for analysis of human serum and plasma. *Analytical chemistry*, 89(1):656–665, 2017.

[40] X. Shen, X. Gong, et al. Normalization and integration of large-scale metabolomics data using support vector regression. *Metabolomics*, 12(5):1–12, 2016.

[41] Z. Yu, G. Zhai, et al. Human serum metabolic profiles are age dependent. *Aging cell*, 11(6):960–967, 2012.

[42] R. Wang-Sattler, Z. Yu, et al. Novel biomarkers for pre-diabetes identified by metabolomics. *Molecular Systems Biology*, 8(615), 2012. doi:10.1038/msb.2012.43.

[43] M. J. Van der Laan, E. C. Polley, and A. E. Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 6(1), 2007.

[44] S. Kim. ppcor: an R package for a fast calculation to semi-partial correlation coefficients. *Communications for statistical applications and methods*, 22(6):665, 2015.

[45] J. Krumsiek, K. Suhre, et al. Gaussian graphical modeling reconstructs pathway reactions from high-throughput metabolomics data. *BMC Systems Biology*, 5, 2011. doi:10.1186/1752-0509-5-21.

[46] A. Beygelzimer, S. Kakadet, et al. *FNN: Fast Nearest Neighbor Search Algorithms and Applications*, 2019. R package version 1.1.3. URL: https://CRAN.R-project.org/package=FNN.

[47] A. Liaw and M. Wiener. Classification and Regression by randomForest. *R News*, 2(3):18–22, 2002. URL: http://cran.r-project.org/doc/Rnews/.

[48] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[49] T. Chen, T. He, et al. *xgboost: Extreme Gradient Boosting*, 2021. R package version 1.4.1.1. URL: https://CRAN.R-project.org/package=xgboost.

[50] D. B. Rubin. Inference and Missing Data. *Biometrika*, 63(3):581, dec 1976. URL: https://www.jstor.org/stable/2335739?origin=crossref, doi:10.2307/2335739.

[51] S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. URL: https://www.jstatsoft.org/v45/i03/.

[52] S. Van Buuren. *Flexible imputation of missing data*. CRC press, 2018.

[53] R. Little and D. Rubin. *Statistical Analysis with Missing Data, Third Edition*, volume 793 of *Wiley Series in Probability and Statistics*. Wiley, apr 2019.

URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119482260, doi:10.1002/9781119482260.

[54] L. L. Doove, S. Van Buuren, and E. Dusseldorp. Recursive partitioning for missing data imputation in the presence of interaction effects. *Computational statistics & data analysis*, 72:92–104, 2014.

[55] L. Torgo. *Data Mining with R, learning with case studies*. Chapman and Hall/CRC, 2010. URL: http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR.

[56] D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*, volume 81 of *Wiley Series in Probability and Statistics*. Wiley, jun 1987. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316696, doi:10.1002/9780470316696.

[57] A. Kuznetsova, P. B. Brockhoff, and R. H. B. Christensen. lmerTest package: Tests in linear mixed effects models. *Journal of Statistical Software*, 82(13):1–26, 2017. doi:10.18637/jss.v082.i13.

[58] B. F. Darst, R. L. Koscik, et al. Longitudinal plasma metabolomics of aging and sex. *Aging*, 11(4):1262–1282, feb 2019. URL: http://www.ncbi.nlm.nih.gov/pubmed/30799310http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6402508, doi:10.18632/aging.101837.

[59] C. Pararasa, J. Ikwuobe, et al. Age-associated changes in long-chain fatty acid profile during healthy aging promote pro-inflammatory monocyte polarization via PPAR γ. *Aging cell*, 15(1):128–139, 2016.

[60] R. Chaleckis, I. Murakami, et al. Individual variability in human blood metabolites identifies age-related differences. *Proceedings of the National Academy of Sciences*, 113(16):4252–4259, 2016.

[61] M. Kuhn. Building Predictive Models in R Using the caret Package. *Journal of Statistical Software*, 28(5), 2008. URL: http://www.jstatsoft.org/v28/i05/, doi:10.18637/jss.v028.i05.

[62] P. Solymos and Z. Zawadzki. *pbapply: Adding Progress Bar to '*apply' Functions*, 2020. R package version 1.4-3. URL: https://CRAN.R-project.org/package=pbapply.

[63] H. Wickham, P. Danenberg, et al. *roxygen2: In-Line Documentation for R*, 2020. R package version 7.1.1. URL: `https://CRAN.R-project.org/package=roxygen2`.

[64] W. Chang, J. Cheng, et al. *shiny: Web Application Framework for R*, 2021. R package version 1.6.0. URL: `https://CRAN.R-project.org/package=shiny`.

[65] W. Chang and B. Borges Ribeiro. *shinydashboard: Create Dashboards with 'Shiny'*, 2018. R package version 0.7.1. URL: `https://CRAN.R-project.org/package=shinydashboard`.

[66] R. Iannone, J. Allaire, and B. Borges. *flexdashboard: R Markdown Format for Flexible Dashboards*, 2020. R package version 0.5.2. URL: `https://CRAN.R-project.org/package=flexdashboard`.

[67] H. Wickham. *ggplot2*. Springer New York, New York, NY, 2009. URL: `https://ggplot2.tidyverse.orghttps://link.springer.com/10.1007/978-0-387-98141-3`, `doi:10.1007/978-0-387-98141-3`.

[68] N. Xiao. *ggsci: Scientific Journal and Sci-Fi Themed Color Palettes for 'ggplot2'*, 2018. R package version 2.9. URL: `https://CRAN.R-project.org/package=ggsci`.

[69] C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. URL: `https://plotly-r.com`.

[70] S. V. Vasaikar, A. K. Savage, et al. A comprehensive platform for analyzing longitudinal multi-omics data. *Nature communications*, 14(1):1684, mar 2023. URL: `http://www.ncbi.nlm.nih.gov/pubmed/36973282http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC10041512`, `doi:10.1038/s41467-023-37432-w`.

[71] B. B. Avants, N. J. Tustison, and J. R. Stone. Similarity-driven multi-view embeddings from high-dimensional biomedical data. *Nature computational science*, 1(2):143–152, feb 2021. `doi:10.1038/s43588-021-00029-8`.

[72] K. Vandereyken, A. Sifrim, et al. Methods and applications for single-cell and spatial multi-omics. *Nature reviews. Genetics*, pages 1–22, mar 2023. `doi:10.1038/s41576-023-00580-2`.

[73] R. Mitra, S. F. McGough, et al. Learning from data with structured missingness. *Nature Machine Intelligence*, 5(1):13–23, jan 2023. URL: `https://www.nature.com/articles/s42256-022-00596-z`, `doi:10.1038/s42256-022-00596-z`.

[74] J. E. Flores, D. M. Claborne, et al. Missing data in multi-omics integration: Recent advances through artificial intelligence. *Frontiers in artificial intelligence*, 6:1098308, 2023. `doi:10.3389/frai.2023.1098308`.

[75] L. Zhang, Y. Zhao, et al. Multi-View Missing Data Completion. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1296–1309, 2018. `doi:10.1109/TKDE.2018.2791607`.

[76] R. Rosato, E. Pagano, et al. Missing data in longitudinal studies: Comparison of multiple imputation methods in a real clinical setting. *Journal of Evaluation in Clinical Practice*, 27(1):34–41, feb 2021. `doi:10.1111/jep.13376`.

[77] R. J. A. Little. Missing-Data Adjustments in Large Surveys. *Journal of Business Economic Statistics*, 6(3):287–296, jul 1988. URL: `http://www.tandfonline.com/doi/abs/10.1080/07350015.1988.10509663`, `doi:10.1080/07350015.1988.10509663`.

[78] D. J. Stekhoven and P. Bühlmann. MissForest–non-parametric missing value imputation for mixed-type data. *Bioinformatics (Oxford, England)*, 28(1):112–8, jan 2012. `doi:10.1093/bioinformatics/btr597`.

[79] O. Troyanskaya, M. Cantor, et al. Missing value estimation methods for DNA microarrays. *Bioinformatics (Oxford, England)*, 17(6):520–5, jun 2001. `doi:10.1093/bioinformatics/17.6.520`.

[80] M. D. Samad and L. Yin. Non-linear regression models for imputing longitudinal missing data. In *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 1–3. IEEE, jun 2019. `doi:10.1109/ICHI.2019.8904528`.

[81] M. H. Huque, J. B. Carlin, et al. A comparison of multiple imputation methods for missing data in longitudinal studies. *BMC Medical Research Methodology*, 18(1):168, dec 2018. URL: `https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-018-0615-6`, `doi:10.1186/s12874-018-0615-6`.

[82] B. Velten, J. M. Braunger, et al. Identifying temporal and spatial patterns of variation from multimodal data using MEFISTO. *Nature Methods*, 19(2):179–186, feb 2022. URL: https://www.nature.com/articles/s41592-021-01343-9, doi:10.1038/s41592-021-01343-9.

[83] B. Velten and O. Stegle. Principles and challenges of modeling temporal and spatial omics data. *Nature Methods*, 20(10):1462–1474, oct 2023. URL: https://www.nature.com/articles/s41592-023-01992-y, doi:10.1038/s41592-023-01992-y.

[84] X. Huang and S. Belongie. Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1510–1519. IEEE, oct 2017. doi:10.1109/ICCV.2017.167.

[85] F.-E. Yang, J.-C. Chang, et al. A Multi-Domain and Multi-Modal Representation Disentangler for Cross-Domain Image Manipulation and Classification. *IEEE Transactions on Image Processing*, 29:2795–2807, 2020. doi:10.1109/TIP.2019.2952707.

[86] Y. Choi, Y. Uh, et al. StarGAN v2: Diverse Image Synthesis for Multiple Domains. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8185–8194. IEEE, jun 2020. doi:10.1109/CVPR42600.2020.00821.

[87] J.-c. Chou and H.-Y. Lee. One-Shot Voice Conversion by Separating Speaker and Content Representations with Instance Normalization. In *Interspeech 2019*, pages 664–668, ISCA, sep 2019. ISCA. doi:10.21437/Interspeech.2019-2663.

[88] M. R. Filbin, A. Mehta, et al. Longitudinal proteomic analysis of severe COVID-19 reveals survival-associated signatures, tissue-specific cell death, and cell-cell interactions. *Cell reports. Medicine*, 2(5):100287, may 2021. URL: https://linkinghub.elsevier.com/retrieve/pii/S2666379121001154http://www.ncbi.nlm.nih.gov/pubmed/33969320http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC8091031, doi:10.1016/j.xcrm.2021.100287.

[89] S. Yu, S. Han, et al. Prediction of Myocardial Infarction Using a Combined Generative Adversarial Network Model and Feature-Enhanced Loss Function. *Metabo-*

lites, 14(5):258, apr 2024. URL: https://www.mdpi.com/2218-1989/14/5/258, doi:10.3390/metabo14050258.

[90] J. M. Ratter-Rieck, M. Shi, et al. Omentin associates with serum metabolite profiles indicating lower diabetes risk: KORA F4 Study. *BMJ Open Diabetes Research Care*, 12(2):e003865, mar 2024. URL: https://drc.bmj.com/lookup/doi/10.1136/bmjdrc-2023-003865, doi:10.1136/bmjdrc-2023-003865.

[91] J. Huang, M. Covic, et al. Validation of Candidate Phospholipid Biomarkers of Chronic Kidney Disease in Hyperglycemic Individuals and Their Organ-Specific Exploration in Leptin Receptor-Deficient db/db Mouse. *Metabolites*, 11(2):89, 2021.

[92] S. Han, J. Huang, et al. TIGER: technical variation elimination for metabolomics data using ensemble learning architecture. *Briefings in bioinformatics*, 23(2), mar 2022. doi:10.1093/bib/bbab535.

[93] M. Shi, S. Han, et al. Identification of candidate metabolite biomarkers for metabolic syndrome and its five components in population-based human cohorts. *Cardiovascular diabetology*, 22(1):141, jun 2023. URL: http://www.ncbi.nlm.nih.gov/pubmed/37328862http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC10276453, doi:10.1186/s12933-023-01862-z.

[94] M. Yang, Y. Li, et al. Partially View-aligned Representation Learning with Noise-robust Contrastive Loss. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1134–1143. IEEE, jun 2021. doi:10.1109/CVPR46437.2021.00119.

[95] J. Wen, Z. Zhang, et al. Unified Embedding Alignment with Missing Views Inferring for Incomplete Multi-View Clustering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5393–5400, jul 2019. doi:10.1609/aaai.v33i01.33015393.

[96] M. Ma, J. Ren, et al. SMIL: Multimodal Learning with Severely Missing Modality. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(3):2302–2310, may 2021. doi:10.1609/aaai.v35i3.16330.

[97] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. nov 2014. arXiv:1411.1784.

[98] I. Goodfellow, J. Pouget-Abadie, et al. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, oct 2020. doi:10.1145/3422622.

[99] C. Shang, A. Palmer, et al. VIGAN: Missing View Imputation with Generative Adversarial Networks. *Proceedings : ... IEEE International Conference on Big Data. IEEE International Conference on Big Data*, 2017:766–775, 2017. doi:10.1109/BigData.2017.8257992.

[100] A. Odena, C. Olah, and J. Shlens. Conditional Image Synthesis with Auxiliary Classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2642–2651. JMLR.org, 2017. doi:10.5555/3305890.3305954.

[101] K. He, X. Zhang, et al. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, jun 2016. doi:10.1109/CVPR.2016.90.

[102] K. He, X. Zhang, et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. IEEE, dec 2015. doi:10.1109/ICCV.2015.123.

[103] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015. doi:10.5555/3045118.3045167.

[104] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. dec 2014. arXiv:1412.6980.

[105] A. Paszke, S. Gross, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. dec 2019. arXiv:1912.01703.

[106] W. Falcon, J. Borovec, and N. Eggert. PyTorchLightning/pytorch-lightning: Bug fixes, new docs, jan 2020. URL: https://doi.org/10.5281/zenodo.3620922, doi:10.5281/zenodo.3620922.

[107] M. Abadi, A. Agarwal, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. mar 2016. URL: http://arxiv.org/abs/1603.04467, arXiv:1603.04467.

[108] T. Chen, S. Kornblith, et al. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020. doi:10.48550/arXiv.2002.05709.

[109] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. nov 2015. arXiv:1511.05440.

[110] H. Zhao, O. Gallo, et al. Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, mar 2017. doi:10.1109/TCI.2016.2644865.

[111] A. Mustafa, A. Mikhailiuk, et al. Training a Task-Specific Image Reconstruction Loss. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 21–30. IEEE, jan 2022. doi:10.1109/WACV51458.2022.00010.

[112] M.-Y. Liu, X. Huang, et al. Few-Shot Unsupervised Image-to-Image Translation. may 2019. arXiv:1905.01723.

[113] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. jul 2016. URL: http://arxiv.org/abs/1607.08022, arXiv:1607.08022.

[114] N. Srivastava, G. Hinton, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014.

[115] A. F. Agarap. Deep Learning using Rectified Linear Units (ReLU). mar 2018. URL: http://arxiv.org/abs/1803.08375, arXiv:1803.08375, doi:10.48550/C.

[116] V. Audigier and M. Resche-Rigon. *micemd: Multiple Imputation by Chained Equations with Multilevel Data*, 2023. URL: https://cran.r-project.org/package=micemd.

[117] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. feb 2018. arXiv:1802.03426.

[118] T. Konopka. umap: Uniform Manifold Approximation and Projection, 2023. URL: https://cran.r-project.org/package=umap.

[119] C. M. Chak, M. E. Lacruz, et al. Ageing investigation using two-time-point metabolomics data from KORA and CARLA studies. *Metabolites*, 9(3):44, 2019.

[120] W. Liu, Y. Liu, et al. Metabolic Biomarkers of Aging and Aging-related Diseases in Chinese Middle-Aged and Elderly Men. *The Journal of nutrition, health and aging*, 22(10):1189–1197, dec 2018. URL: https://linkinghub.elsevier.com/retrieve/pii/S1279770723010540, doi:10.1007/s12603-018-1062-0.

[121] A. S. Levey, L. A. Stevens, et al. A new equation to estimate glomerular filtration rate. *Annals of internal medicine*, 150(9):604–12, may 2009. URL: http://www.ncbi.nlm.nih.gov/pubmed/19414839http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2763564, doi:10.7326/0003-4819-150-9-200905050-00006.

[122] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. URL: http://dl.acm.org/citation.cfm?id=1953048.2078195%5Cnhttp://dl.acm.org/ft_gateway.cfm?id=2078195&type=pdf, arXiv:1201.0490, doi:10.1007/s13398-014-0173-7.2.

[123] G. Lemaître, F. Nogueira, and C. K. Aridas. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL: http://jmlr.org/papers/v18/16-365.html.

[124] J. Nano, B. Schöttker, et al. Novel biomarkers of inflammation, kidney function and chronic kidney disease in the general population. *Nephrology Dialysis Transplantation*, 37(10):1916–1926, sep 2022. doi:10.1093/ndt/gfab294.

[125] T. Saito and M. Rehmsmeier. Precrec: fast and accurate precision-recall and ROC curve calculations in R. *Bioinformatics (Oxford, England)*, 33(1):145–147, jan 2017. URL: http://www.ncbi.nlm.nih.gov/pubmed/27591081http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5408773, doi:10.1093/bioinformatics/btw570.

[126] D. T. Leach, K. G. Stratton, et al. malbacr: a package for standardized implementation of batch correction methods for omics data. *Analytical Chemistry*, 95(33):12195–12199, 2023.

A Supplementary Figures

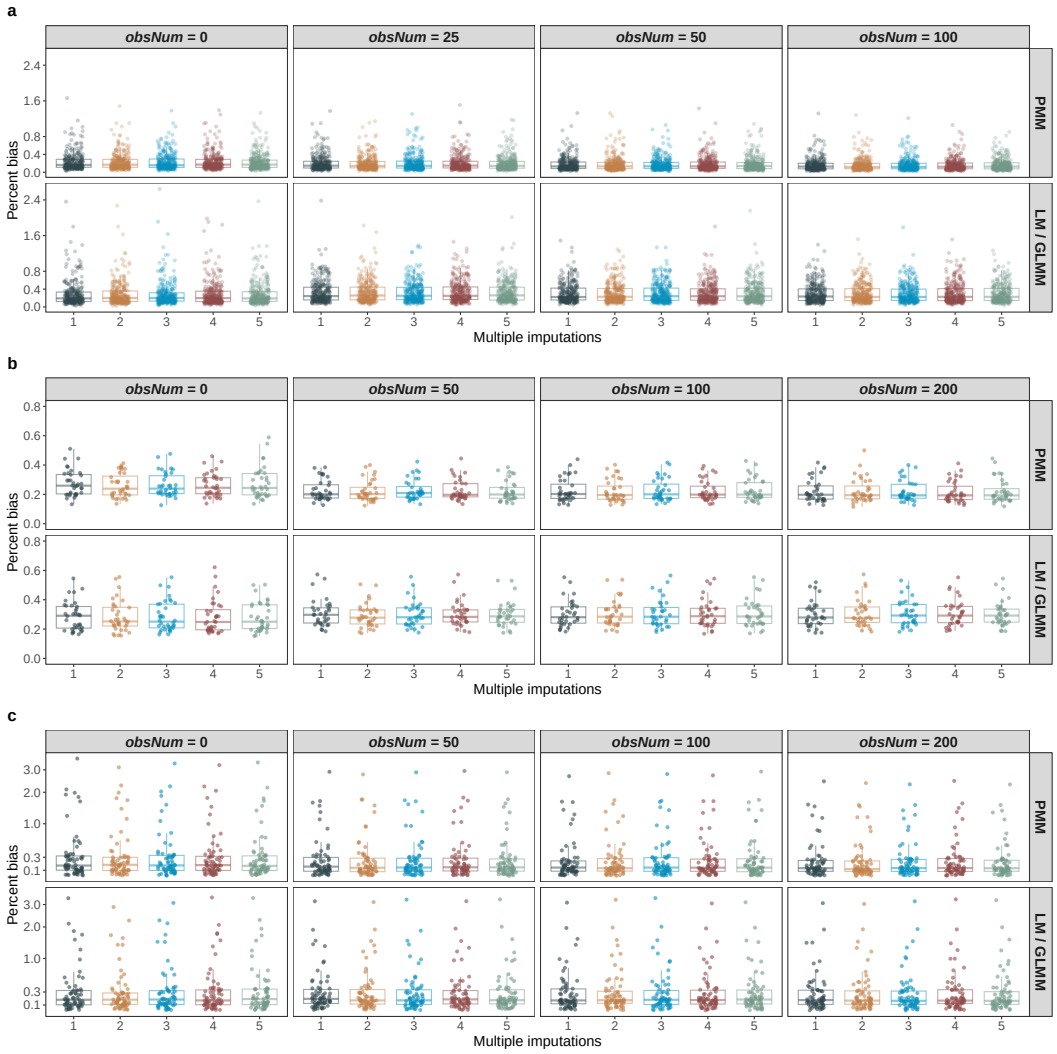

Figure A.1: Evaluation of Each of Multiple Imputations: PB PB values are calculated on the MGH COVID proteomics (**a**), KORA metabolomics (**b**), and KORA multi-omics datasets (**c**), with varying $obsNum$. The imputation is performed five times ($m = 5$). Please note that LM is used when $obsNum = 0$.

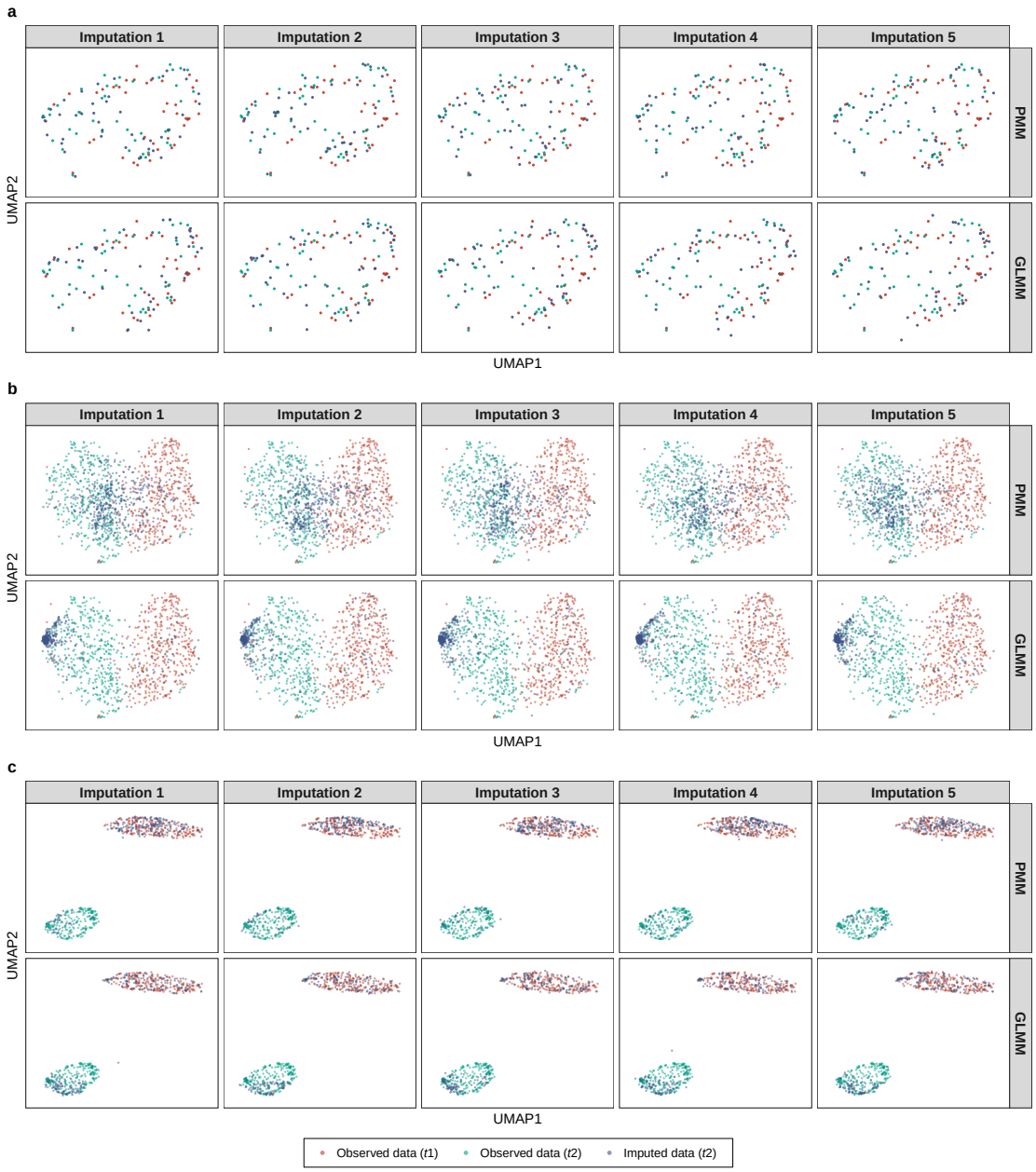

Figure A.2: Evaluation of Each of Multiple Imputation: UMAP UMAP representations of each individual imputation (Imputation 1 to 5) of methods PMM and GLMM and corresponding observed data of three benchmark datasets. UMAP models are initially fitted with the training data from the MGH COVID proteomics dataset (**a**, $t1$: D0, $t2$: D3), KORA metabolomics dataset (**b**, $t1$: F4, $t2$: FF4), and KORA multi-omics dataset (**c**, $t1$: S4, $t2$: F4). Then the trained models are applied to the corresponding observed data (red and blue dots for $t1$ and $t2$) and each individual imputation of PMM and GLMM (green dots) under the setting of $obsNum = 100$ for the MGH COVID dataset and $obsNum = 200$ for the two KORA-derived datasets.

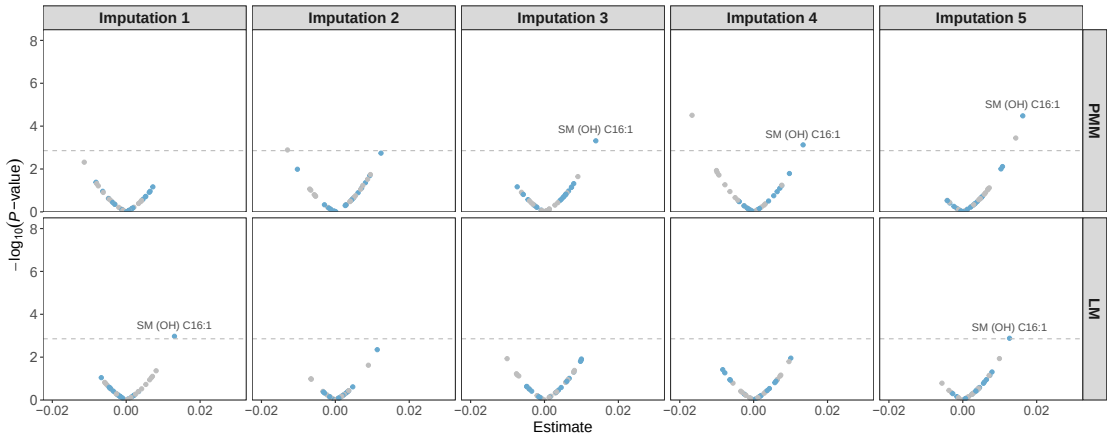

Figure A.3: Age-Associated Metabolites in Each of Multiple Imputations The age-associate metabolites identified from each individual imputation (Imputation 1 to 5) of methods PMM and LM. The evaluation is performed on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ ($N = 417$) of the KORA metabolomics dataset, under $obsNum = 0$. 18 significant metabolites ($P < 0.05/36$) identified from the observed data are shown in blue. Replicated metabolites from the imputed data ($obsNum = 0$) are marked with labels.

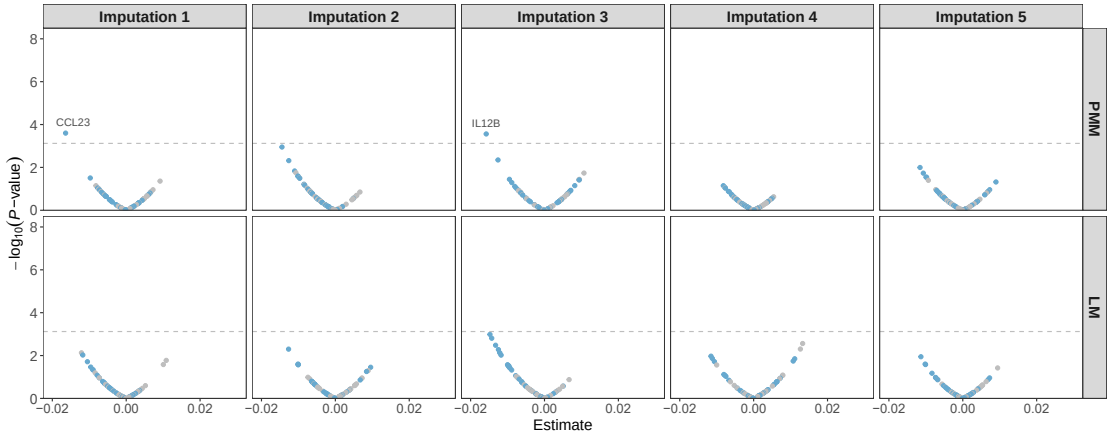

Figure A.4: eGFR-Associated Proteins in Each of Multiple Imputations The eGFR-associated proteins identified from each individual imputation (Imputation 1 to 5) of methods PMM and LM. The evaluation is performed on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ ($N = 212$) of the KORA multi-omics dataset, under $obsNum = 0$. 28 significant metabolites ($P < 0.05/66$) identified from the observed data are shown in blue. Replicated metabolites from the imputed data ($obsNum = 0$) are marked with labels.

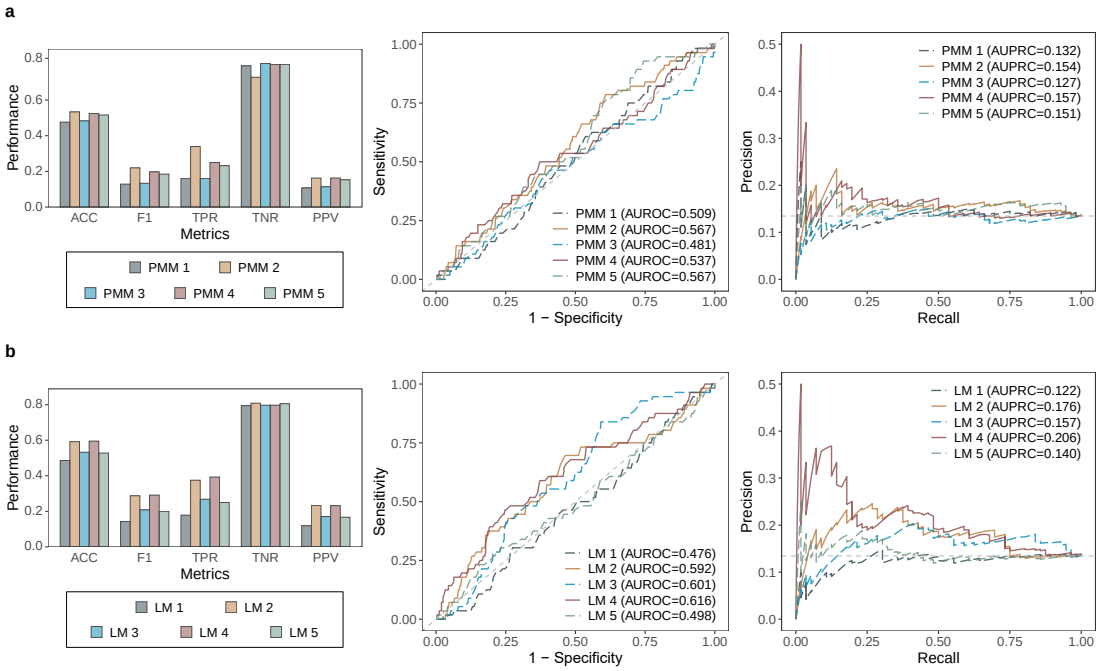

Figure A.5: CKD Prediction in Each of Multiple Imputations: KORA Metabolomics The performance of CKD prediction on the KORA metabolomics dataset using each of the 5 multiple imputations produced by methods PMM (**a**) and LM (**b**). Methods are evaluated on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ ($N = 416$, $N_{positive} = 56$, $N_{negative} = 360$), under $obsNum = 0$. PMM 1 to 5 and LM 1 to 5 indicate different individual imputations from PMM and LM, respectively. Models are trained using the BRF algorithm with identical hyperparameters and evaluated using LOOCV. The barplot (left) shows multi-metric performance. The dashed lines in the ROC (middle) and PR (right) curves represent the performance of a hypothetical model with no predictive capability.

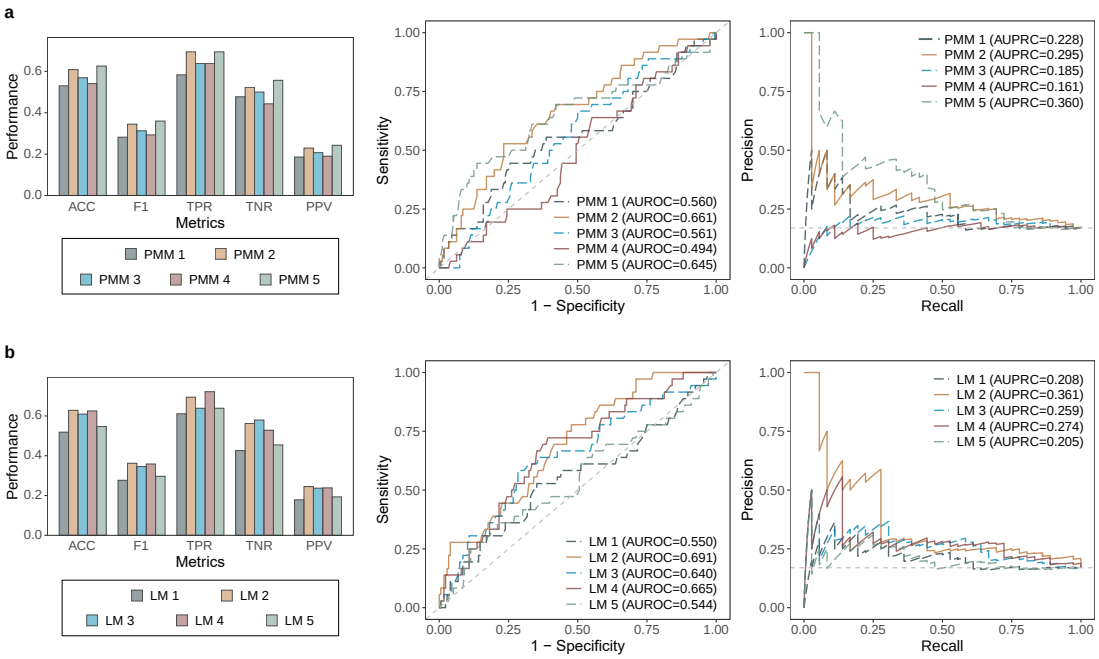

Figure A.6: CKD Prediction in Each of Multiple Imputations: KORA Multi-Omics The performance of CKD prediction on the KORA multi-omics dataset using each of the 5 multiple imputations produced by methods PMM (**a**) and LM (**b**). Methods are evaluated on $\mathcal{D}^{\text{test}}_{v=v2,t=t2}$ ($N = 212$, $N_{positive} = 36$, $N_{negative} = 176$), under $obsNum = 0$. PMM 1 to 5 and LM 1 to 5 indicate different individual imputations from PMM and LM, respectively. Models are trained using the BRF algorithm with identical hyperparameters and evaluated using LOOCV. The barplot (left) shows multi-metric performance. The dashed lines in the ROC (middle) and PR (right) curves represent the performance of a hypothetical model with no predictive capability.

B Supplementary Tables

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	≥ LOD*	RSD	≥ LOD*	CC**	RSD	≥ LOD*
Acylcarnitine	C0	0.056	1.000	0.080	1.000	0.797	0.068	1.000
	C2	0.056	1.000	0.106	1.000	0.872	0.097	1.000
	C3	0.076	1.000	0.115	1.000	0.827	0.065	1.000
	C3:1	0.402	0.053	0.377	0.005	-0.049	0.356	0.006
	C3-DC (C4-OH)	0.244	0.198	0.423	0.099	0.384	0.212	0.803
	C3-OH	0.509	0.030	0.978	0.050	-0.028	1.029	0.000
	C4	0.077	1.000	0.114	1.000	0.830	0.108	1.000
	C4:1	0.201	0.513	0.353	0.105	0.081	0.159	0.562
	C5	0.089	0.991	0.163	1.000	0.722	0.111	0.998
	C5:1	0.250	0.020	0.274	0.018	0.315	0.176	0.022
	C5:1-DC	0.344	0.279	0.468	0.075	0.255	0.206	0.123
	C5-DC (C6-OH)	0.264	0.653	0.368	0.521	0.442	0.174	0.167
	C5-M-DC	0.280	0.026	0.516	0.038	0.220	0.364	0.023

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	$\geq$ LOD*	RSD	$\geq$ LOD*	CC**	RSD	$\geq$ LOD*
Acylcarnitine	C5-OH (C3-DC-M)	0.352	0.216	0.261	0.140	0.300	0.219	0.203
	C6 (C4:1-DC)	0.104	0.665	0.138	0.840	0.694	0.087	0.167
	C6:1	0.305	0.059	0.362	0.026	0.193	0.176	0.002
	C7-DC	0.183	0.719	0.302	0.674	0.806	0.128	0.652
	C8	0.093	0.615	0.179	0.493	0.898	0.080	0.914
	C8:1	NA	NA	0.097	0.991	0.000	NA	NA
	C9	0.214	0.979	0.345	0.930	0.884	0.143	0.966
	C10	0.071	0.994	0.138	0.983	0.914	0.073	1.000
	C10:1	0.082	0.755	0.112	0.354	0.866	0.112	0.585
	C10:2	0.158	0.963	0.154	0.542	0.630	0.142	0.315
	C12	0.066	0.968	0.098	0.887	0.889	0.079	0.883
	C12:1	0.103	0.271	0.133	0.024	0.849	0.114	0.671
	C12-DC	0.123	0.000	0.165	0.000	0.034	0.240	0.000
	C14	0.099	0.977	0.116	0.444	0.840	0.116	0.953
	C14:1	0.106	1.000	0.207	1.000	0.731	0.152	1.000
	C14:1-OH	0.190	0.792	0.183	0.668	0.695	0.127	0.524
	C14:2	0.147	0.990	0.127	0.994	0.850	0.158	0.984
	C14:2-OH	0.257	0.554	0.263	0.441	0.262	0.165	0.402
	C16	0.073	1.000	0.108	1.000	0.754	0.101	1.000
	C16-OH	0.307	0.226	0.258	0.018	0.353	0.196	0.043

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	$\geq$ LOD*	RSD	$\geq$ LOD*	CC**	RSD	$\geq$ LOD*
Acylcarnitine	C16:1	0.123	0.799	0.111	0.023	0.743	0.156	0.226
	C16:1-OH	0.258	0.340	0.210	0.013	0.616	0.163	0.590
	C16:2	0.287	0.920	0.197	0.611	0.685	0.184	0.536
	C16:2-OH	0.276	0.081	0.199	0.011	0.210	0.182	0.009
	C18	0.099	1.000	0.113	0.999	0.791	0.133	1.000
	C18:1	0.076	1.000	0.139	0.999	0.789	0.078	1.000
	C18:1-OH	0.265	0.093	0.350	0.010	0.384	0.174	0.006
	C18:2	0.092	0.999	0.119	1.000	0.790	0.100	1.000
Amino acid	Ala	0.131	1.000	NA	NA	NA	0.064	1.000
	Arg	0.140	0.998	0.075	1.000	0.564	0.133	1.000
	Asn	0.108	1.000	NA	NA	NA	0.091	1.000
	Asp	0.118	1.000	NA	NA	NA	0.120	1.000
	Cit	0.125	1.000	NA	NA	NA	0.121	1.000
	Gln	0.150	1.000	0.143	1.000	0.369	0.137	1.000
	Glu	0.151	1.000	NA	NA	NA	0.112	1.000
	Gly	0.130	1.000	0.087	1.000	0.798	0.112	1.000
	His	0.164	0.999	0.112	1.000	0.413	0.111	1.000
	Ile	0.172	1.000	NA	NA	NA	0.091	1.000
	Leu	0.162	0.991	NA	NA	NA	0.106	1.000
	Lys	0.180	1.000	NA	NA	NA	0.136	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	≥ LOD*	RSD	≥ LOD*	CC**	RSD	≥ LOD*
Amino acid	Met	0.150	1.000	0.147	1.000	0.551	0.117	1.000
	Orn	0.168	1.000	0.117	1.000	0.739	0.125	1.000
	Phe	0.140	0.999	0.090	1.000	0.464	0.106	1.000
	Pro	0.119	1.000	0.104	1.000	0.800	0.095	1.000
	Ser	0.150	1.000	0.096	1.000	0.544	0.132	1.000
	Thr	0.164	0.998	0.119	1.000	0.670	0.092	1.000
	Trp	0.146	1.000	0.075	1.000	0.484	0.114	1.000
	Tyr	0.173	0.999	0.086	1.000	0.612	0.102	1.000
	Val	0.158	1.000	0.200	1.000	0.573	0.099	1.000
	xLeu	NA	NA	0.094	1.000	NA	NA	NA
Biogenic amine	Ac-Orn	0.202	0.796	NA	NA	NA	0.219	1.000
	ADMA	0.231	0.670	NA	NA	NA	0.163	1.000
	alpha-AAA	0.597	0.999	NA	NA	NA	0.172	1.000
	c4-OH-Pro	NA	NA	NA	NA	NA	0.122	1.000
	Carnosine	0.898	0.040	NA	NA	NA	0.116	1.000
	Creatinine	0.143	0.999	NA	NA	NA	0.057	1.000
	DOPA	0.183	0.453	NA	NA	NA	0.178	1.000
	Dopamine	0.246	0.001	NA	NA	NA	0.157	1.000
	Histamine	0.438	0.905	NA	NA	NA	0.121	0.366
	Kynurenine	0.122	0.978	NA	NA	NA	0.107	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	≥ LOD*	RSD	≥ LOD*	CC**	RSD	≥ LOD*
Biogenic amine	Met-SO	0.252	0.996	NA	NA	NA	0.144	1.000
	Nitro-Tyr	0.607	0.077	NA	NA	NA	0.114	1.000
	OH-Pro	-	0.023	NA	NA	NA	NA	NA
	PEA	0.226	0.006	NA	NA	NA	0.090	0.000
	Putrescine	0.409	0.939	NA	NA	NA	0.083	0.957
	Sarcosine	0.287	0.044	NA	NA	NA	NA	NA
	SDMA	0.330	0.980	NA	NA	NA	0.175	1.000
	Serotonin	0.464	0.998	NA	NA	NA	0.085	1.000
	Spermidine	0.244	0.992	NA	NA	NA	0.068	1.000
	Spermine	0.162	0.100	NA	NA	NA	0.058	1.000
	t4-OH-Pro	NA	NA	NA	NA	NA	0.137	1.000
	Taurine	0.138	0.975	NA	NA	NA	0.073	1.000
	total DMA	0.201	0.996	NA	NA	NA	0.140	1.000
Glycerophospholipid	lysoPC a C6:0	NA	NA	0.821	0.224	NA	NA	NA
	lysoPC a C14:0	0.061	0.000	0.280	0.201	0.432	0.112	0.000
	lysoPC a C16:0	0.062	1.000	0.114	1.000	0.610	0.105	1.000
	lysoPC a C16:1	0.068	1.000	0.111	1.000	0.833	0.112	1.000
	lysoPC a C17:0	0.073	1.000	0.203	1.000	0.778	0.113	1.000
	lysoPC a C18:0	0.065	1.000	0.111	1.000	0.731	0.087	1.000
	lysoPC a C18:1	0.066	1.000	0.122	1.000	0.818	0.122	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	$\geq$ LOD*	RSD	$\geq$ LOD*	CC**	RSD	$\geq$ LOD*
Glycerophospholipid	lysoPC a C18:2	0.066	1.000	0.101	1.000	0.846	0.124	1.000
	lysoPC a C20:3	0.082	1.000	0.118	1.000	0.815	0.089	1.000
	lysoPC a C20:4	0.070	1.000	0.096	1.000	0.808	0.099	1.000
	lysoPC a C24:0	0.198	0.252	0.250	0.089	0.011	0.119	0.000
	lysoPC a C26:0	0.407	0.454	0.348	0.609	-0.042	0.377	1.000
	lysoPC a C26:1	0.083	0.000	0.111	0.000	0.102	0.265	0.994
	lysoPC a C28:0	0.335	0.254	0.321	0.481	0.016	0.272	0.992
	lysoPC a C28:1	0.327	0.990	0.227	0.998	0.176	0.273	1.000
	PC aa C24:0	0.222	0.716	0.271	0.778	0.000	0.122	0.996
	PC aa C26:0	0.249	0.059	0.386	0.114	-0.109	0.261	0.297
	PC aa C28:1	0.098	1.000	0.108	1.000	0.789	0.113	1.000
	PC aa C30:0	0.092	1.000	0.127	1.000	0.793	0.080	1.000
	PC aa C30:2	1.457	0.558	0.709	0.043	-0.022	2.637	0.828
	PC aa C32:0	0.083	1.000	0.126	1.000	0.688	0.136	1.000
	PC aa C32:1	0.079	1.000	0.132	1.000	0.880	0.209	1.000
	PC aa C32:2	0.113	1.000	0.205	0.999	0.806	0.176	1.000
	PC aa C32:3	0.086	1.000	0.105	1.000	0.715	0.143	1.000
	PC aa C34:1	0.070	1.000	0.125	1.000	0.717	0.124	1.000
	PC aa C34:2	0.063	1.000	0.172	1.000	0.236	0.150	1.000
	PC aa C34:3	0.066	1.000	0.152	1.000	0.766	0.139	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	≥ LOD*	RSD	≥ LOD*	CC**	RSD	≥ LOD*
Glycerophospholipid	PC aa C34:4	0.067	1.000	0.107	1.000	0.882	0.112	1.000
	PC aa C36:0	0.311	1.000	0.200	1.000	0.707	0.264	1.000
	PC aa C36:1	0.071	1.000	0.106	1.000	0.845	0.169	1.000
	PC aa C36:2	0.065	1.000	0.094	1.000	0.664	0.119	1.000
	PC aa C36:3	0.062	1.000	0.117	1.000	0.676	0.107	1.000
	PC aa C36:4	0.066	1.000	0.122	1.000	0.743	0.120	1.000
	PC aa C36:5	0.057	1.000	0.142	1.000	0.910	0.129	1.000
	PC aa C36:6	0.093	1.000	0.156	1.000	0.893	0.105	1.000
	PC aa C38:0	0.112	1.000	0.163	1.000	0.852	0.081	1.000
	PC aa C38:1	0.274	1.000	0.205	0.998	0.544	0.179	1.000
	PC aa C38:3	0.058	1.000	0.087	1.000	0.834	0.100	1.000
	PC aa C38:4	0.059	1.000	0.081	1.000	0.848	0.082	1.000
	PC aa C38:5	0.059	1.000	0.111	1.000	0.814	0.099	1.000
	PC aa C38:6	0.071	1.000	0.116	1.000	0.864	0.084	1.000
	PC aa C40:1	0.157	0.143	0.164	0.091	0.636	0.141	0.178
	PC aa C40:2	0.186	1.000	0.146	1.000	0.649	0.105	1.000
	PC aa C40:3	0.146	1.000	0.134	1.000	0.721	0.126	1.000
	PC aa C40:4	0.063	1.000	0.088	1.000	0.754	0.147	1.000
	PC aa C40:5	0.065	1.000	0.079	1.000	0.841	0.069	1.000
	PC aa C40:6	0.064	1.000	0.076	0.000	0.890	0.072	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	$\geq$ LOD*	RSD	$\geq$ LOD*	CC**	RSD	$\geq$ LOD*
Glycerophospholipid	PC aa C42:0	0.114	1.000	0.140	1.000	0.871	0.089	1.000
	PC aa C42:1	0.135	1.000	0.162	1.000	0.795	0.099	1.000
	PC aa C42:2	0.161	1.000	0.161	1.000	0.733	0.101	0.999
	PC aa C42:4	0.114	1.000	0.136	1.000	0.591	0.124	1.000
	PC aa C42:5	0.089	1.000	0.115	1.000	0.809	0.085	1.000
	PC aa C42:6	0.100	0.961	0.133	0.675	0.755	0.193	1.000
	PC ae C30:0	0.184	1.000	0.325	0.997	0.566	0.135	1.000
	PC ae C30:1	0.512	0.837	0.492	0.941	0.074	1.183	0.492
	PC ae C30:2	0.265	1.000	0.182	0.934	0.226	0.160	0.001
	PC ae C32:1	0.089	1.000	0.105	1.000	0.564	0.168	1.000
	PC ae C32:2	0.104	1.000	0.134	1.000	0.616	0.187	1.000
	PC ae C34:0	0.105	1.000	0.113	1.000	0.787	0.155	1.000
	PC ae C34:1	0.072	1.000	0.123	1.000	0.693	0.129	1.000
	PC ae C34:2	0.072	1.000	0.128	1.000	0.834	0.141	1.000
	PC ae C34:3	0.070	1.000	0.107	1.000	0.901	0.110	1.000
	PC ae C36:0	0.216	1.000	0.380	1.000	0.580	0.167	1.000
	PC ae C36:1	0.126	1.000	0.128	1.000	0.756	0.111	1.000
	PC ae C36:2	0.072	1.000	0.146	1.000	0.808	0.110	1.000
	PC ae C36:3	0.069	1.000	0.138	1.000	0.798	0.123	1.000
	PC ae C36:4	0.063	1.000	0.126	1.000	0.815	0.116	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	≥ LOD*	RSD	≥ LOD*	CC**	RSD	≥ LOD*
Glycerophospholipid	PC ae C36:5	0.067	1.000	0.103	1.000	0.847	0.117	1.000
	PC ae C38:0	0.086	1.000	0.127	1.000	0.875	0.121	1.000
	PC ae C38:1	0.189	0.997	0.147	1.000	0.445	0.232	0.961
	PC ae C38:2	0.143	1.000	0.143	1.000	0.579	0.213	1.000
	PC ae C38:3	0.088	1.000	0.116	1.000	0.717	0.107	1.000
	PC ae C38:4	0.060	1.000	0.132	1.000	0.739	0.107	1.000
	PC ae C38:5	0.061	1.000	0.123	1.000	0.759	0.114	1.000
	PC ae C38:6	0.066	1.000	0.104	1.000	0.850	0.116	1.000
	PC ae C40:0	NA	NA	0.082	0.011	NA	NA	NA
	PC ae C40:1	0.142	1.000	0.137	1.000	0.755	0.090	1.000
	PC ae C40:2	0.116	1.000	0.122	1.000	0.788	0.088	1.000
	PC ae C40:3	0.095	1.000	0.112	1.000	0.688	0.088	1.000
	PC ae C40:4	0.085	1.000	0.107	1.000	0.769	0.080	1.000
	PC ae C40:5	0.061	1.000	0.094	1.000	0.647	0.120	1.000
	PC ae C40:6	0.060	1.000	0.120	1.000	0.809	0.058	1.000
	PC ae C42:0	0.127	0.372	0.183	0.147	0.392	0.119	0.026
	PC ae C42:1	0.196	1.000	0.147	1.000	0.530	0.108	1.000
	PC ae C42:2	0.164	1.000	0.187	1.000	0.659	0.102	1.000
	PC ae C42:3	0.113	1.000	0.130	1.000	0.754	0.105	1.000
	PC ae C42:4	0.079	1.000	0.109	1.000	0.812	0.111	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	$\geq$ LOD*	RSD	$\geq$ LOD*	CC**	RSD	$\geq$ LOD*
Glycerophospholipid	PC ae C42:5	0.065	1.000	0.075	0.999	0.842	0.072	0.983
	PC ae C44:3	0.204	1.000	0.140	1.000	0.466	0.117	1.000
	PC ae C44:4	0.120	1.000	0.128	1.000	0.776	0.141	1.000
	PC ae C44:5	0.056	1.000	0.084	1.000	0.860	0.086	1.000
	PC ae C44:6	0.064	1.000	0.090	1.000	0.859	0.081	1.000
Sphingolipid	SM (OH) C14:1	0.111	1.000	0.137	1.000	0.745	0.141	1.000
	SM (OH) C16:1	0.110	1.000	0.096	1.000	0.789	0.136	1.000
	SM (OH) C22:1	0.104	1.000	0.152	1.000	0.673	0.165	1.000
	SM (OH) C22:2	0.110	1.000	0.136	1.000	0.697	0.101	1.000
	SM (OH) C24:1	0.148	1.000	0.186	1.000	0.650	0.192	1.000
	SM C16:0	0.105	1.000	0.137	1.000	0.547	0.088	1.000
	SM C16:1	0.098	1.000	0.124	1.000	0.713	0.152	1.000
	SM C18:0	0.136	1.000	0.106	1.000	0.702	0.073	1.000
	SM C18:1	0.090	1.000	0.118	1.000	0.783	0.107	1.000
	SM C20:2	0.131	0.999	0.174	0.999	0.526	0.139	1.000
	SM C22:3	4.690	0.434	0.583	0.558	0.019	0.931	0.825
	SM C24:0	0.109	1.000	0.152	1.000	0.431	0.168	1.000
	SM C24:1	0.105	1.000	0.157	1.000	0.565	0.146	1.000
	SM C26:0	0.405	1.000	0.537	1.000	0.567	0.274	1.000
	SM C26:1	0.197	1.000	0.259	1.000	0.669	0.262	1.000

Table B.1: QC Results of Targeted Metabolomics: KORA S4-F4-FF4 (continued)

Biochemical Class	Metabolite	KORA S4		KORA F4			KORA FF4	
		RSD	$\geq$ LOD*	RSD	$\geq$ LOD*	CC**	RSD	$\geq$ LOD*
Monosaccharide	H1	0.056	1.000	0.077	1.000	0.624	0.066	1.000

*the proportion of subject samples metabolite values $\geq$ LOD.

**the Spearman CC between the metabolite values measured with the tow different kits.

Table B.2: QC Results of Proteomics from Inflammation Panel: KORA S4-F4

Biomarker	UniProt ID	KORA S4			KORA F4		
		IntraRSD	InterRSD	≥ LOD*	IntraRSD	InterRSD	≥ LOD*
4E-BP1	Q13541	0.058	0.642	1.000	0.038	0.086	1.000
ADA	P00813	0.069	0.294	1.000	0.037	0.088	1.000
ARTN	Q5T4W7	NA	NA	0.061	NA	NA	0.047
AXIN1	O15169	0.051	0.484	0.984	0.045	0.087	0.899
BDNF	P23560	NA	NA	NA	0.028	0.081	0.731
Beta-NGF	P01138	NA	NA	0.014	0.036	0.076	1.000
CASP-8	Q14790	0.062	0.480	0.863	0.071	0.359	0.993
CCL11	P51671	0.058	0.142	1.000	0.028	0.069	1.000
CCL19	Q99731	0.055	0.157	1.000	0.029	0.070	1.000
CCL20	P78556	0.073	0.213	1.000	0.032	0.071	1.000
CCL23	P55773	0.054	0.123	1.000	0.029	0.061	1.000
CCL25	O15444	0.060	0.108	1.000	0.032	0.076	1.000
CCL28	Q9NRJ3	0.074	0.120	1.000	0.041	0.146	0.990
CCL3	P10147	0.065	0.127	1.000	0.033	0.090	1.000
CCL4	P13236	0.061	0.146	1.000	0.028	0.058	1.000
CD244	Q9BZW8	0.072	0.213	1.000	0.022	0.089	1.000
CD40	P25942	0.053	0.253	1.000	0.025	0.078	1.000
CD5	P06127	0.061	0.208	1.000	0.031	0.093	1.000
CD6	P30203	0.116	0.286	1.000	0.038	0.129	1.000
CD8A	P01732	0.084	0.142	1.000	NA	NA	NA

Table B.2: QC Results of Proteomics from Inflammation Panel: KORA S4-F4 (continued)

Biomarker	UniProt ID	KORA S4			KORA F4		
		IntraRSD	InterRSD	≥ LOD*	IntraRSD	InterRSD	≥ LOD*
CDCP1	Q9H5V8	0.103	0.125	1.000	0.039	0.096	1.000
CSF-1	P09603	0.057	0.082	1.000	0.022	0.088	1.000
CST5	P28325	0.048	0.119	1.000	0.024	0.066	1.000
CX3CL1	P78423	0.084	0.130	1.000	0.037	0.108	1.000
CXCL1	P09341	0.055	0.430	1.000	0.025	0.058	1.000
CXCL10	P02778	0.060	0.157	1.000	0.031	0.060	1.000
CXCL11	O14625	0.056	0.380	1.000	0.025	0.058	1.000
CXCL5	P42830	0.045	0.401	1.000	0.027	0.058	1.000
CXCL6	P80162	0.056	0.302	1.000	0.024	0.089	1.000
CXCL9	Q07325	0.055	0.128	1.000	0.032	0.054	1.000
DNER	Q8NFT8	0.042	0.096	1.000	0.023	0.080	1.000
EN-RAGE	P80511	0.079	0.282	1.000	0.049	0.110	1.000
FGF-19	O95750	0.057	0.133	1.000	0.029	0.084	1.000
FGF-21	Q9NSA1	0.063	0.111	1.000	0.031	0.073	1.000
FGF-23	Q9GZV9	0.083	0.092	0.186	0.050	0.076	0.998
FGF-5	P12034	0.089	0.085	0.234	0.043	0.087	0.975
Flt3L	P49771	0.063	0.116	1.000	0.027	0.081	1.000
GDNF	P39905	0.089	0.121	0.541	0.085	0.084	0.905
HGF	P14210	0.053	0.148	1.000	0.025	0.075	1.000
IFN-gamma	P01579	0.069	0.141	1.000	NA	NA	0.015

Table B.2: QC Results of Proteomics from Inflammation Panel: KORA S4-F4 (continued)

Biomarker	UniProt ID	KORA S4			KORA F4		
		IntraRSD	InterRSD	≥ LOD*	IntraRSD	InterRSD	≥ LOD*
IL-1 alpha	P01583	NA	NA	0.050	NA	NA	0.031
IL-10	P22301	0.095	0.162	0.997	0.053	0.104	1.000
IL-10RA	Q13651	0.082	0.097	0.674	0.026	0.092	0.735
IL-10RB	Q08334	0.065	0.096	1.000	0.031	0.104	1.000
IL-12B	P29460	0.066	0.133	1.000	0.032	0.069	1.000
IL-13	P35225	0.070	0.119	0.094	NA	NA	0.051
IL-15RA	Q13261	0.087	0.099	1.000	0.056	0.105	0.976
IL-17A	Q16552	0.100	0.131	0.539	0.049	0.093	0.670
IL-17C	Q9P0M4	0.067	0.058	0.441	0.062	0.069	0.777
IL-18	Q14116	0.056	0.149	1.000	0.030	0.064	1.000
IL-18R1	Q13478	0.053	0.111	1.000	0.030	0.076	1.000
IL-2	P60568	NA	NA	0.002	NA	NA	0.001
IL-20	Q9NYY1	NA	NA	0.036	NA	NA	0.046
IL-20RA	Q9UHF4	0.076	0.092	0.113	NA	NA	0.070
IL-22RA1	Q8N6P7	0.090	0.147	0.129	NA	NA	0.002
IL-24	Q13007	NA	NA	0.045	NA	NA	0.065
IL-2RB	P14784	NA	0.018	0.082	NA	NA	0.046
IL-33	O95760	NA	NA	0.019	NA	NA	0.020
IL-4	P05112	NA	0.024	0.103	NA	NA	0.081
IL-5	P05113	0.084	0.156	0.099	NA	NA	0.274

Table B.2: QC Results of Proteomics from Inflammation Panel: KORA S4-F4 (continued)

Biomarker	UniProt ID	KORA S4			KORA F4		
		IntraRSD	InterRSD	≥ LOD*	IntraRSD	InterRSD	≥ LOD*
IL-6	P05231	0.053	0.124	0.985	0.044	0.107	0.979
IL-7	P13232	0.077	0.204	1.000	0.027	0.060	0.999
IL-8	P10145	0.057	0.139	1.000	0.030	0.089	1.000
LAP TGF-beta-1	P01137	0.053	0.173	1.000	0.100	0.166	0.999
LIF	P15018	0.054	0.074	0.043	NA	NA	0.050
LIF-R	P42702	0.072	0.111	1.000	0.034	0.101	1.000
MCP-1	P13500	0.049	0.112	1.000	0.029	0.060	1.000
MCP-2	P80075	0.089	0.165	1.000	0.027	0.080	1.000
MCP-3	P80098	0.077	0.091	0.895	0.065	0.103	0.970
MCP-4	Q99616	0.048	0.327	1.000	0.030	0.095	1.000
MMP-1	P03956	0.047	0.282	1.000	0.023	0.054	1.000
MMP-10	P09238	0.049	0.114	1.000	0.027	0.088	1.000
NRTN	Q99748	NA	NA	0.044	NA	NA	0.028
NT-3	P20783	0.077	0.128	0.996	0.048	0.105	0.993
OPG	O00300	0.046	0.120	1.000	0.023	0.050	1.000
OSM	P13725	0.065	0.169	1.000	0.023	0.069	0.996
PD-L1	Q9NZQ7	0.060	0.155	1.000	0.048	0.096	1.000
SCF	P21583	0.042	0.089	1.000	0.021	0.058	1.000
SIRT2	Q8IXJ6	0.070	1.002	0.870	0.054	0.097	1.000
SLAMF1	Q13291	NA	0.044	0.359	0.064	0.107	1.000

Table B.2: QC Results of Proteomics from Inflammation Panel: KORA S4-F4 (continued)

Biomarker	UniProt ID	KORA S4			KORA F4		
		IntraRSD	InterRSD	≥ LOD*	IntraRSD	InterRSD	≥ LOD*
ST1A1	P50225	NA	NA	0.870	0.064	0.140	0.955
STAMBP	O95630	0.067	0.864	1.000	0.036	0.092	1.000
TGF-alpha	P01135	0.090	0.105	1.000	0.031	0.095	1.000
TNF	P01375	0.061	0.095	1.000	NA	NA	0.041
TNFB	P01374	0.077	0.103	1.000	0.044	0.088	1.000
TNFRSF9	Q07011	0.063	0.109	1.000	0.030	0.097	1.000
TNFSF14	O43557	0.066	0.352	1.000	0.028	0.078	1.000
TRAIL	P50591	0.049	0.092	1.000	0.027	0.072	1.000
TRANCE	O14788	0.082	0.134	1.000	0.046	0.088	1.000
TSLP	Q969D9	NA	NA	0.061	NA	NA	0.008
TWEAK	O43508	0.065	0.139	1.000	0.023	0.062	1.000
uPA	P00749	0.044	0.115	1.000	0.025	0.046	1.000
VEGF-A	P15692	0.067	0.132	1.000	0.029	0.076	1.000

*the proportion of subject samples protein values ≥ LOD.

C Package Manual: TIGER

Package 'TIGERr'

January 4, 2022

Type Package

Title Technical Variation Elimination with Ensemble Learning Architecture

Version 1.0.0

Author Siyu Han [aut, cre], Jialing Huang [aut], Francesco Foppiano [aut], Cornelia Prehn [aut], Jerzy Adamski [aut], Karsten Suhre [aut], Ying Li [aut], Giuseppe Matullo [aut], Freimut Schliess [aut], Christian Gieger [aut], Annette Peters [aut], Rui Wang-Sattler [aut]

Maintainer Siyu Han <siyu.han@helmholtz-muenchen.de>

Acknowledgments TAI Yun-hsiu, WANG Ruoyu, CHENG Ming, GUO Yuan, LI Han, FAN Linrui

Description The R implementation of TIGER.
TIGER integrates random forest algorithm into an innovative ensemble learning architecture. Benefiting from this advanced architecture, TIGER is resilient to outliers, free from model tuning and less likely to be affected by specific hyperparameters. TIGER supports targeted and untargeted metabolomics data and is competent to perform both intra- and inter-batch technical variation removal. TIGER can also be used for cross-kit adjustment to ensure data obtained from different analytical assays can be effectively combined and compared.
Reference: Han S. et al. (2022) <doi:10.1093/bib/bbab535>.

License GPL (>= 3)

Depends R (>= 3.5.0)

Imports parallel (>= 2.1.0),
pbapply (>= 1.4-3),
ppcor (>= 1.1),
randomForest (>= 4.6-14),
stats (>= 3.0.0)

BugReports https://github.com/HAN-Siyu/TIGER/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

R topics documented:

`compute_RSD` *Compute RSD (relative standard deviation)*

Description

This function computes the RSD (relative standard deviation) of the values in `input_data`. Missing values are removed before the computation automatically.

Usage

```
compute_RSD(input_data)
```

Arguments

`input_data`	a numeric vector

Details

The RSD in this function is computed by:

`sd(input_data,na.rm = TRUE) / mean(input_data,na.rm = TRUE)`.

Value

The RSD of the values in `input_data` is computed, as a numeric of length one.

Examples

```
RSD_1 <- compute_RSD(c(1:10))

data(FF4_qc) # load demo dataset

# RSD of QC:
RSD_2 <- sapply(FF4_qc[FF4_qc$sampleType == "QC", -c(1:5)], compute_RSD)
quantile(RSD_2)

# RSD of different types of QC samples:
# (each metabolote has its own RSD)
RSD_3 <- aggregate(FF4_qc[-c(1:5)], by = list(Type = FF4_qc$sampleType),
                   FUN = compute_RSD)
```

compute_targetVal *Compute target values for ensemble learning architecture*

Description

This function provides an advanced option to calculate the target values of one reference dataset (i.e. QC_num, numeric values of quality control samples). The generated target values (a list) can be further passed to argument targetVal_external in function run_TIGER such that TIGER can align the test_samples with the reference dataset. This is useful for longitudinal datasets correction and cross-kit adjustment. See case study section of our original paper for detailed explanation.

Usage

```
compute_targetVal(
  QC_num,
  sampleType,
  batchID = NULL,
  targetVal_method = c("mean", "median"),
  targetVal_batchWise = FALSE,
  targetVal_removeOutlier = !targetVal_batchWise,
  coerce_numeric = FALSE
)
```

Arguments

QC_num	a numeric data.frame including the metabolite values of quality control (QC) samples. Missing values and infinite values will not be taken into account. Row: sample. Column: metabolite variable. See Examples.
sampleType	a vector corresponding to QC_num to specify the type of each QC sample. QC samples of the **same type** should have the **same type name**. See Examples.
batchID	a vector corresponding to QC_num to specify the batch of each sample. Ignored if targetVal_batchWise = FALSE. See Examples.
targetVal_method	a character string specifying how the target values are computed. Can be "mean" (default) or "median". See Details.
targetVal_batchWise	logical. If TRUE, the target values will be computed based on each batch, otherwise, based on the whole dataset. Setting TRUE might be useful if your dataset has very obvious batch effects, but this may also make the algorithm less robust. See Details. Default: FALSE.
targetVal_removeOutlier	logical. If TRUE, outliers will be removed before the computation. Outliers are determined with 1.5 * IQR (interquartile range) rule. We recommend turning this off when the target values are computed based on batches. See Details. Default: !targetVal_batchWise.
coerce_numeric	logical. If TRUE, values in QC_num will be coerced to numeric before the computation. The columns cannot be coerced will be removed (with warnings). See Examples. Default: FALSE.

Details

See run_TIGER.

Value

If targetVal_batchWise = FALSE, the function returns a list of length one containing the target values computed on the whole dataset.

If targetVal_batchWise = TRUE, a list containing the target values computed on different batches is returned. The length of the returned list equals the number of batch specified by batchID.

Examples

```
data(FF4_qc) # load demo dataset
QC_num <- FF4_qc[-c(1:5)] # only contain numeric metabolite values.

# target values computed on the whole dataset:
tarVal_1 <- compute_targetVal(QC_num = QC_num,
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = FALSE,
                              targetVal_removeOutlier = TRUE)

# target values computed on batches:
tarVal_2 <- compute_targetVal(QC_num = QC_num,
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = TRUE,
                              targetVal_removeOutlier = FALSE)

# If coerce_numeric = TRUE,
# columns cannot be coerced to numeric will be removed (with warnings):
tarVal_3 <- compute_targetVal(QC_num = FF4_qc[-c(4:5)],
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = TRUE,
                              targetVal_removeOutlier = FALSE,
                              coerce_numeric = TRUE)
identical(tarVal_2, tarVal_3)  # identical to tarVal_2

## Not run:

# will throw errors if input data have non-numeric columns
# and coerce_numeric = FALSE:

tarVal_4 <- compute_targetVal(QC_num = FF4_qc,
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = TRUE,
                              targetVal_removeOutlier = FALSE,
                              coerce_numeric = FALSE)

## End(Not run)
```

FF4_qc *Accompanying QC samples of KORA FF4 (demo data)*

Description

This demo dataset, a data.frame with 232 samples (rows) and 108 variables (columns). The dataset includes four types of quality control (QC) samples from 29 kit plates:

- QC1 (N = 29, one per plate),
- QC2 (N = 29, one per plate),
- QC3 (N = 29, one per plate),
- QC (N = 145, five per plate).

The columns include sample ID, sample type, plate ID, well position, injection order and the concentrations of 103 selected targeted metabolites. These QC samples are measured with the cohort samples of KORA FF4 (Cooperative Health Research in the Augsburg Region, the second follow-up study, 2013–2014) using the analytical assay Biocrates Absolute*IDQ*® p180 (BIOCRATES Life Sciences AG, Innsbruck, Austria).

In our paper, we used QC as training samples, while QC1, QC2, QC3 and cohort samples were used as test samples. The cohort data are operated by Helmholtz Zentrum München and available via KORA platform https://www.helmholtz-muenchen.de/en/kora/index.html upon reasonable request. See Reference for detailed information.

Usage

```
data(FF4_qc)
```

Reference

Han S. *et al*. TIGER: technical variation elimination for metabolomics data using ensemble learning architecture. *Briefings in Bioinformatics* (2022) bbab535. https://doi.org/10.1093/bib/bbab535.

run_TIGER *Run TIGER to eliminate technical variation*

Description

Use TIGER algorithm to eliminate the technical variation in metabolomics data. TIGER supports targeted and untargeted metabolomics data and is competent to perform both intra- and inter-batch technical variation removal.

Usage

```
run_TIGER(
  test_samples,
  train_samples,
  col_sampleID,
  col_sampleType,
  col_batchID,
  col_order = NULL,
  col_position = NULL,
  targetVal_external = NULL,
  targetVal_method = c("mean", "median"),
  targetVal_batchWise = FALSE,
  targetVal_removeOutlier = !targetVal_batchWise,
  selectVar_external = NULL,
  selectVar_corType = c("cor", "pcor"),
  selectVar_corMethod = c("pearson", "spearman"),
  selectVar_minNum = 5,
  selectVar_maxNum = 10,
  selectVar_batchWise = FALSE,
  mtry_percent = seq(0.2, 0.8, 0.2),
  nodesize_percent = seq(0.2, 0.8, 0.2),
  ...,
  parallel.cores = 2
)
```

Arguments

`test_samples` (required) a data.frame containing the samples to be corrected (for example, subject samples). This data.frame should contain columns of

- sample ID (required): name or label for each sample,
- sample type (required): indicating the type of each sample,
- batch ID (required): the batch of each sample,
- order information (optional): injection order or temporal information of each sample,
- position information (optional): well position of each sample,
- metabolite values (required): values to be normalised. Infinite values are not allowed.

Row: sample. Column: variable. See Examples.

`train_samples` (required) a data.frame containing the quality control (QC) samples used for model training. The columns in this data.frame should correspond to the columns in `test_samples`. And `test_samples` and `train_samples` should have the identical column names.

`col_sampleID` (required) a character string indicating the name of the column that specifies the sample ID of each sample. The values in this column will not affect the data correction process but can act as labels for different samples. See Examples.

`col_sampleType` (required) a character string indicating the name of the column that specifies the type (such as QC1, QC2, subject) of each sample. This column can be used to indicate different kinds of QC samples in `train_samples`. QC samples of the **same type** should have the **same type name**. See Examples.

col_batchID (required) a character string indicating the name of the column that specifies the batch ID of each sample. See Examples.

col_order (optional) NULL or a character string indicating the name of the column that contains the injection order or temporal information (numeric values). This can explicitly ask the algorithm capture the technical variation introduced by injection order, which might be useful when your data have very obvious temporal drifts. If NULL (default), train_samples and test_samples should have **No** column contains injection order information.

col_position (optional) NULL or a character string indicating the name of the column that contains the well position information (numeric values). This can explicitly ask the algorithm capture the technical variation introduced by well position, which might be useful when the well position has a great impact during data acquisition. If NULL (default), train_samples and test_samples should have **No** column contains well position information.

targetVal_external (optional) a list generated by function compute_targetVal. See Details.

targetVal_method a character string specifying how target values are to be computed. Can be "mean" (default) or "median". Ignored if a list of external target values has been assigned to targetVal_external.

targetVal_batchWise logical. If TRUE, the target values will be computed based on each batch, otherwise, based on the whole dataset. Setting TRUE might be useful if your dataset has very obvious batch effects, but this may also make the algorithm less robust. Default: FALSE. Ignored if a list of external target values has been assigned to targetVal_external.

targetVal_removeOutlier logical. If TRUE, outliers will be removed before the computation. Outliers are determined with 1.5 * IQR (interquartile range) rule. We recommend turning this off when the target values are computed based on batches. Default: !targetVal_batchWise. Ignored if a list of external target values has been assigned to targetVal_external.

selectVar_external (optional) a list generated by function select_variable. See Details.

selectVar_corType a character string indicating correlation ("cor", default) or partial correlation ("pcor") is to be used. Can be abbreviated. Ignored if a list of selected variables has been assigned to selectVar_external. **Note**: computing partial correlations of a large dataset can be very time-consuming.

selectVar_corMethod a character string indicating which correlation coefficient is to be computed. One of "spearman" (default) or "pearson". Can be abbreviated. Ignored if a list of selected variables has been assigned to selectVar_external.

selectVar_minNum an integer specifying the minimum number of the selected metabolite variables (injection order and well position are not regarded as metabolite variables). If NULL, no limited, but 1 at least. Default: 5. Ignored if a list of selected variables has been assigned to selectVar_external.

selectVar_maxNum an integer specifying the maximum number of the selected metabolite variables (injection order and well position are not regarded as metabolite variables). If

NULL, no limited, but no more than the number of all available metabolite variables. Default: 10. Ignored if a list of selected variables has been assigned to selectVar_external.

selectVar_batchWise
: (advanced) logical. Specify whether the variable selection should be performed based on each batch. Default: FALSE. Ignored if a list of selected variables has been assigned to selectVar_external. **Note**: the support of batch-wise variable selection is provided for data requiring special processing (for example, data with strong batch effects). But in most case, batch-wise variable selection is not necessary. Setting TRUE can make the algorithm less robust.

mtry_percent
: (advanced) a numeric vector indicating the percentages of selected variables randomly sampled as candidates at each split when training random forest models (base learners). **Note**: providing more arguments will include more base learners into the ensemble model, which will increase the processing time. Default: seq(0.2,0.8,0.2).

nodesize_percent
: (advanced) a numeric vector indicating the percentages of sample size used as the minimum sizes of the terminal nodes in random forest models (base learners). **Note**: providing more arguments will include more base learners into the ensemble model, which will increase the processing time. Default: seq(0.2,0.8,0.2).

...
: (advanced) optional arguments (except mtry and nodesize) to be passed to randomForest for model training. Arguments mtry and nodesize are determined by mtry_percent and nodesize_percent. See randomForest and Examples. **Note**: providing more arguments will include more base learners into the ensemble model, which will increase the processing time.

parallel.cores
: an integer (== -1 or >= 1) specifying the number of cores for parallel computation. Setting -1 to run with all cores. Default: 2.

Details

TIGER can effectively process the datasets with its default setup. The following hyperparameters are provided to customise the algorithm and achieve the best possible performance. These hyperparameters are also practical for some special purposes (such as cross-kit adjustment, longitudinal dataset correction) or datasets requiring special processing (for example, data with very strong temporal drifts or batch effects). We recommend users to examine the normalised result with different metrics, such as RSD (relative standard deviation), MAPE (mean absolute percentage error) and PCA (principal component analysis), especially when using advanced options of TIGER.

Hyperparameters for target value computation

- targetVal_external

  TIGER by default captures and eliminate the technical variation within the input dataset, and the target values are automatically computed from train_samples. The target values can also be calculated from a reference dataset using function compute_targetVal and then passed to this function as an argument. This will enable TIGER to align test_samples with the reference dataset. In this case, train_samples is still the accompanying QC samples of test_samples. And argument targetVal_external accepts external target values (a list). If the list of external target values is provided, values in targetVal_method, targetVal_batchWise and targetVal_removeOutlier will be ignored.

- targetVal_method

The target values can be the mean or median values of metabolite values. The target values of different kinds of QC samples are computed separately. "mean" is recommended here, but the optimal selection can differ for different datasets.

- targetVal_batchWise

  The target values can be computed from the whole dataset or from different batches. By default, the target values are computed based on the whole dataset. Computing based on batches (targetVal_batchWise = TRUE) is only recommended when the samples has very strong batch effects. For example, we set this as TRUE when normalising WaveICA's Amide dataset in our original paper.

- targetVal_removeOutlier

  If computing is based on the whole dataset (targetVal_batchWise = TRUE), users can remove the outliers in each metabolite by setting targetVal_removeOutlier as TRUE. This can weaken the impact of extreme values. If targetVal_batchWise = FALSE, it is generally not recommended to remove outliers, as we assume the input data have strong batch effects and contain extreme values—we hope TIGER can take these into account. Code for checking outliers is adapted from boxplot.stats.

Hyperparameters for variable selection

- selectVar_external:

  This argument accepts a list of selected variables generated by select_variable. This is helpful when you want to use the same selected variables to correct several datasets. You can also pass a self-defined list to this argument, as long as the self-defined list has similar data structure as the one generated by select_variable.

- selectVar_corType and selectVar_corMethod:

  TIGER supports Pearson product-moment correlation ("pearson") and Spearman's rank correlation ("spearman") to compute correlation coefficients ("cor") or partial correlation coefficients ("por") for variable selection. See cor and pcor for further details.

- selectVar_minNum and selectVar_maxNum:

  For an objective metabolite to be corrected, the intersection of its top t highly-correlated metabolites calculated from training and test samples are selected to train the ensemble model. The highly-correlated metabolites are the ones with correlation coefficients greater than 0.5 (the objective metabolite itself will not be regarded as its highly-correlated metabolite). Arguments selectVar_minNum and selectVar_maxNum are used to avoid selecting too many or too few metabolites. Selecting too many metabolites can lower the process, sometimes even lower the accuracy.

- selectVar_batchWise:

  Advanced option designed for special cases. Setting it TRUE might be useful when your data have very obvious batch effects.

Hyperparameters for model construction

- mtry_percent, nodesize_percent and ...:

  Advanced options to specify mtry, nodesize and other related arguments in randomForest for a customised ensemble learning architecture. See Examples.

Value

This function returns a data.frame with the same data structure as the input test_samples, but the metabolite values are the normalised/corrected ones. NA and zeros in the original test_samples will not be changed or normalised.

Reference

Han S. *et al*. TIGER: technical variation elimination for metabolomics data using ensemble learning architecture. *Briefings in Bioinformatics* (2022) bbab535. https://doi.org/10.1093/bib/bbab535.

Examples

```
data(FF4_qc) # load demo dataset

# QC as training samples; QC1, QC2 and QC3 as test samples:
train_samples <- FF4_qc[FF4_qc$sampleType == "QC",]
test_samples  <- FF4_qc[FF4_qc$sampleType != "QC",]

# col_sampleID includes labels. You can assign names for different samples:
train_samples$sampleID <- "train"
test_samples$sampleID  <- "test"

# Use default setting and
# include injection order and well position into feature set:
test_norm_1 <- run_TIGER(test_samples = test_samples,
                         train_samples = train_samples,
                         col_sampleID  = "sampleID",     # input column name
                         col_sampleType = "sampleType",  # input column name
                         col_batchID = "plateID",        # input column name
                         col_order = "injectionOrder",   # input column name
                         col_position = "wellPosition",  # input column name
                         parallel.cores = 2)

# If the information of injection order and well position is not available,
# or you don't want to use them:
train_data <- train_samples[-c(4:5)]  # remove the two columns
test_data  <- test_samples[-c(4:5)]   # remove the two columns

test_norm_2 <- run_TIGER(test_samples = test_data,
                         train_samples = train_data,
                         col_sampleID  = "sampleID",
                         col_sampleType = "sampleType",
                         col_batchID = "plateID",
                         col_order = NULL,                # set NULL
                         col_position = NULL,             # set NULL
                         parallel.cores = 2)

# If use external target values and selected variables with
# customised settings:
target_val <- compute_targetVal(QC_num = train_samples[-c(1:5)],
                                sampleType = train_samples$sampleType,
                                batchID = train_samples$plateID,
                                targetVal_method = "median",
                                targetVal_batchWise = TRUE)

select_var <- select_variable(train_num = train_samples[-c(1:5)],
                              test_num = test_samples[-c(1:5)],
                              train_batchID = train_samples$plateID,
                              test_batchID = test_samples$plateID,
                              selectVar_corType = "pcor",
                              selectVar_corMethod = "spearman",
```

```
                                selectVar_minNum = 10,
                                selectVar_maxNum = 30,
                                selectVar_batchWise = TRUE)

test_norm_3 <- run_TIGER(test_samples = test_samples,
                         train_samples = train_samples,
                         col_sampleID  = "sampleID",
                         col_sampleType = "sampleType",
                         col_batchID = "plateID",
                         col_order = "injectionOrder",
                         col_position = "wellPosition",
                         targetVal_external = target_val,
                         selectVar_external = select_var,
                         parallel.cores = 2)

# The definitions of other hyperparameters correspond to
# randomForest::randomForest().
# If want to include more hyperparameters into model training,
# put hyperparameter values like this:
mtry_percent <- c(0.4, 0.8)
nodesize_percent <- c(0.4, 0.8)
replace <- c(TRUE, FALSE)
ntree <- c(100, 200, 300)

test_norm_4 <- run_TIGER(test_samples = test_data,
                         train_samples = train_data,
                         col_sampleID  = "sampleID",
                         col_sampleType = "sampleType",
                         col_batchID = "plateID",
                         mtry_percent = mtry_percent,
                         nodesize_percent = nodesize_percent,
                         replace = replace,
                         ntree = ntree,
                         parallel.cores = 2)

# test_norm_4 is corrected by the ensemble model consisted of base learners
# trained with (around) 24 different hyperparameter combinations:
expand.grid(mtry_percent, nodesize_percent, replace, ntree)

# Note: mtry and nodesize are calculated by mtry_percent and nodesize_percent,
#       duplicated hyperparameter combinations, if any, will be removed.
#       Thus, the total number of hyperparameter combinations can be less than 24.
#       This is determined by the shape of your input datasets.
```

`select_variable` *Select variables for ensemble learning architecture*

Description

This function provides an advanced option to select metabolite variables from external dataset(s). The selected variables (as a list) can be further passed to argument `selectVar_external` in function `run_TIGER` for a customised data correction.

Usage

```
select_variable(
  train_num,
  test_num = NULL,
  train_batchID = NULL,
  test_batchID = NULL,
  selectVar_corType = c("cor", "pcor"),
  selectVar_corMethod = c("spearman", "pearson"),
  selectVar_minNum = 5,
  selectVar_maxNum = 10,
  selectVar_batchWise = FALSE,
  coerce_numeric = FALSE
)
```

Arguments

`train_num`	a numeric data.frame **only** including the metabolite values of training samples (can be quality control samples). Information such as injection order or well position need to be excluded. Row: sample. Column: metabolite variable. See Examples.
`test_num`	an optional numeric data.frame including the metabolite values of test samples (can be subject samples). If provided, the column names of `test_num` should correspond to the column names of `train_num`. Row: sample. Column: metabolite variable. If `NULL`, the variables will be selected based on `train_num` only. See Examples.
`train_batchID`	`NULL` or a vector corresponding to `train_num` to specify the batch of each sample. Ignored if `selectVar_batchWise = FALSE`. See Examples.
`test_batchID`	`NULL` or a vector corresponding to `test_num` to specify the batch of each sample. Ignored if `selectVar_batchWise = FALSE`. See Examples.
`selectVar_corType`	a character string indicating correlation (`"cor"`, default) or partial correlation (`"pcor"`) is to be used. Can be abbreviated. See Details. **Note**: computing partial correlations of a large dataset can be very time-consuming.
`selectVar_corMethod`	a character string indicating which correlation coefficient is to be computed. One of `"spearman"` (default) or `"pearson"`. Can be abbreviated. See Details.
`selectVar_minNum`	an integer specifying the minimum number of the selected variables. If `NULL`, no limited, but 1 at least. See Details. Default: 5.
`selectVar_maxNum`	an integer specifying the maximum number of the selected variables. If `NULL`, no limited, but `ncol(train_num) -1` at most. See Details. Default: 10.
`selectVar_batchWise`	(advanced) logical. Specify whether the variable selection should be performed based on each batch. Default: `FALSE`. **Note**: if `TRUE`, batch ID of each sample are required. The support of batch-wise variable selection is provided for data requiring special processing (for example, data with strong batch effects). But in most case, batch-wise variable selection is not necessary. Setting `TRUE` might make the algorithm less robust. See Details.

coerce_numeric logical. If TRUE, values in train_num and test_num will be coerced to numeric before the computation. The columns cannot be coerced will be removed (with warnings). See Examples. Default: FALSE.

Details

See run_TIGER.

Value

If selectVar_batchWise = FALSE, the function returns a list of length one containing the selected variables computed on the whole dataset.

If selectVar_batchWise = TRUE, a list containing the selected variables computed on different batches is returned. The length of the returned list equals the number of batch specified by test_batchID and/or train_batchID.

Examples

```
data(FF4_qc) # load demo dataset

# QC as training samples; QC1, QC2 and QC3 as test samples:
train_samples <- FF4_qc[FF4_qc$sampleType == "QC",]
test_samples  <- FF4_qc[FF4_qc$sampleType != "QC",]

# Only numeric data of metabolite variables are allowed:
train_num = train_samples[-c(1:5)]
test_num  = test_samples[-c(1:5)]

# If the selection is performed on the whole dataset:
# based on training samples only:
selected_var_1 <- select_variable(train_num = train_num,
                                  test_num  = NULL,
                                  selectVar_batchWise = FALSE)

# also consider test samples:
selected_var_2 <- select_variable(train_num = train_num,
                                  test_num  = test_num,
                                  selectVar_batchWise = FALSE)

# If the selection is based on different batches:
# (In selectVar_batchWise, batch ID is required.)
selected_var_3 <- select_variable(train_num = train_num,
                                  test_num  = NULL,
                                  train_batchID = train_samples$plateID,
                                  test_batchID  = NULL,
                                  selectVar_batchWise = TRUE)

# If coerce_numeric = TRUE,
# columns cannot be coerced to numeric will be removed (with warnings):
# (In this example, columns of injection order and well position are excluded.
# Because we don't want to calculate the correlations between metabolites and
# injection order/well position.)
selected_var_4 <- select_variable(train_num = train_samples[-c(4,5)],
                                  train_batchID = train_samples$plateID,
                                  selectVar_batchWise = TRUE,
                                  coerce_numeric = TRUE)
```

```
identical(selected_var_3, selected_var_4)  # identical to selected_var_3

## Not run:

# will throw errors if input data have non-numeric columns
# and coerce_numeric = FALSE:

selected_var_5 <- select_variable(train_num = train_samples[-c(4,5)],
                                  coerce_numeric = FALSE)

## End(Not run)
```

Index

D Package Manual: LEOPARD

How to Train Your LEOPARD

August 2, 2024

1 Intro

This notebook provides a brief instruction for training your own LEOPARD. You can also use the following code to reproduce the result of the MGH COVID dataset reported in our paper. The result may vary slightly with each run due to the stochastic mechanisms involved. Any questions regarding the code, please contact the zookeeper: Siyu Han (siyu.han@tum.de)

1.1 Step 1: Import required modules

Before proceeding, ensure that the following dependencies are properly installed: - python: 3.79 - numpy: 1.23.5 - pandas: 1.4.4 - scikit-learn: 1.0.2 - pytorch: 1.11.10 - pytorch_lightning: 1.6.4 - tensorboard: 2.10.0 - cuda (if use GPU): 11.3

The listed version numbers represent those utilized during our development. We cannot guarantee the compatibility or identical outcomes with different versions.

[1]:
```
import torch
import pytorch_lightning as pl
from pytorch_lightning.loggers import TensorBoardLogger

from src.data import *
from src.train import TrainLEOPARD
```

1.2 Step 2: Prepare dataset

This is done by the function `prepare_dataset()`. This function does the following things: 1. load data and format them into training/validation/test sets with the function `load_split_data()`. 2. scale data use a scaler with the function `scale_data()`. 3. create an instance of `OmicsDataset` class using scaled data for each data split.

The function `prepare_dataset()` receives arguments for the following parameters: - `data_dir`: a string to specify the folder containing the data of two views (A and B) and two timepoints (1 and 2). The data are saved in .csv files with names like "vA_t1_test.csv", "vB_t1_train.csv". "v" and "t" denote views and timepoints. "v*_t*_train.csv" used for model training and validation. Missing values can be encoded as NA. Even if data from view B at timepoint 2 are completely missing, you still need to provide a "vB_t2_train.csv" file with the corresponding sample ID and variable ID, and missing values are indicated with NA. "v*_t*_test.csv" is optional and only used for performance evaluation. Default: `"data\MGH_COVID"` - `valSet_ratio`: a numeric value between 0 and 1 to specify the ratio of data from "v*_t*_train.csv" used for constructing the validation

set. Default: `0.2` - `trainNum`: a numeric value or `"all"` indicating how many samples will be randomly selected from the training data for training. Default: `"all"` - `obsNum`: a numeric value or `"all"` indicating how many samples from "vB_t2_train.csv" will be used for training. Default: `0` - `use_scaler`: a string to indicate which scaler is used to scale data. Support `"standard"`, `"robust"`, and `"minmax"`. Description of the scalers please refer to the User Guide of `sklearn.preprocessing`. Default: `"standard"` - `set_seed`: a numeric value to set seed for reproducible results. Default: `1` - `save_data_dir`: `None` or a path used for saving the indices of samples used in training. Default: "

[2]:
```
obsNum = 0
trainNum = "all"

loaded_data = prepare_dataset(data_dir="data/MGH_COVID", valSet_ratio=0.2,
                              trainNum=trainNum, obsNum=obsNum,
                              use_scaler="standard", set_seed=1,␣
 ↪save_data_dir=None)
```

```
+ loading data:
  -load vA_t1 as vA_t1
  -load vA_t2 as vA_t2
  -load vB_t1 as vB_t1
  -load vB_t2 as vB_t2
+ Data scaling using standard scaler:
  - vA_t1: use scaler_vA
  - vA_t2: use scaler_vA
  - vB_t1: use scaler_vB
  - vB_t2: use scaler_vB
```

1.3 Step 3: Create an instance of the `TrainLEOPARD` class

The pytorch code of LEOPARD is organized into `TrainLEOPARD`, a `LightningModule`. LEOPARD is fully customizable. You can adapt your LEOPARD using the following parameters when instantiating an instance of the `TrainLEOPARD` class: - `train_set`, `val_set`, `test_set`: data prepared by `prepare_dataset()` for training, validation and test - `scaler_viewA`, `scaler_viewB`: scaler used by `prepare_dataset()` to transform data in two views - `pre_layers_viewA`, `pre_layers_viewB`: pre-layers for view A and view B to convert them to the same dimension. Default: `[64]`, `[64]` - `post_layers_viewA`, `post_layers_viewB`: post-layers for view A and view B to convert embeddings back to data in the original dimension. Default: `[64]`, `[64]`

- `encoder_content_layers`: layers for the content encoder. A list where the length indicates the total number of layers, and each element specifies the size of the corresponding layer. Default: `[64, 64, 64]`
- `encoder_content_norm`: a list indicates if using normalization for the layers in the content encoder. Supported `"instance"`, `"batch"`, and `"none"`. Default: `["instance", "instance", "instance"]`
- `encoder_content_dropout`: a list specifies dropout rate for each layer in the content encoder. Default: `[0, 0, 0]`
- `encoder_temporal_layers`: layers for the temporal encoder. Default: `[64, 64, 64]`

- `encoder_temporal_norm`: if use normalization for the layers in the temporal encoder? Supported `"instance"`, `"batch"`, and `"none"`. Default: `["none", "none", "none"]`
- `encoder_temporal_dropout`: dropout rate for each layer in the temporal encoder. Default: `[0, 0, 0]`
- `generator_block_num`: how many layers/blocks are used for the generator. Default: `3`
- `generator_norm`: if use normalization for the layers in the generator? Supported `"instance"`, `"batch"`, and `"none"`. Default: `["none", "none", "none"]`
- `generator_dropout`: dropout rate for each layer in the generator. Default: `[0, 0, 0]`
- `merge_mode`: re-entangle content and temporal representations by concatenation (`"concat"`) or AdaIN (`"adain"`)? Default: `"adain"`
- `discriminator_layers`: layers for the multi-task discriminator. Default: `[128, 128]`
- `discriminator_norm`: if use normalization for the layers in the discriminator? Supported `"instance"`, `"batch"`, and `"none"`. Default: `["none", "none"]`
- `discriminator_dropout`: dropout rate for each layer in the discriminator. Default: `[0, 0]`
- `reconstruction_loss`: use `"MSE"` or `"MAE"` to compute reconstruction loss? Default: `"MSE"`
- `adversarial_loss`: use `"MSE"` or `"BCE"` to compute adversarial loss? Default: `"MSE"`
- `weight_reconstruction`, `weight_adversarial`, `weight_representation`, `weight_contrastive`: weights for different losses. Default: `1`, `1`, `0.1`, `0.1`
- `contrastive_temperature`: temperature for NT-Xent-based contrastive loss. Default: `0.05`
- `lr_G`, `lr_D`: learning rate for generator process (encoders and generator) and discrimination process (discriminator). ***You need to tune this for your own datasets.*** Default: `0.0005`, `0.0005`
- `b1_G`, `b1_D`: beta_1 for Adam Optimizer. Default: `0.9`, `0.9`
- `lr_scheduler_G`, `lr_scheduler_D`: `"none"` or use `"LambdaLR"` or `"SGDR"` as lr scheduler? Default: `"none"`, `"none"`
- `use_projection_head`: if use projection head for contrastive learning? Default: `False`
- `projection_output_size`: set output size of projection head. Ignored if `use_projection_head=False`. Default: `0`
- `batch_size`: batch size. ***You need to adjust this based on your sample size.*** Default: `32`
- `note`: add some additional texts as a hyperparameter to label each run. Default: `"obsNum_" + str(obsNum)`

Some hyperparameters (especially `lr_G`, `lr_D`, and `batch_size`) may need to be tuned for your own datasets.

[3]:
```
my_leopard = TrainLEOPARD(train_set=loaded_data['train_set'],␣
↪val_set=loaded_data['val_set'],
```

```
                          test_set=loaded_data['test_set'],
                          scaler_viewA=loaded_data['scaler_viewA'],␣
↪scaler_viewB=loaded_data['scaler_viewB'],

                          pre_layers_viewA=[64], pre_layers_viewB=[64],
                          post_layers_viewA=[64], post_layers_viewB=[64],

                          encoder_content_layers=[64, 64, 64],
                          encoder_content_norm=['instance', 'instance',␣
↪'instance'],
                          encoder_content_dropout=[0, 0, 0],

                          encoder_temporal_layers=[64, 64, 64],
                          encoder_temporal_norm=['none', 'none', 'none'],
                          encoder_temporal_dropout=[0, 0, 0],

                          generator_block_num=3,
                          generator_norm=['none', 'none', 'none'],
                          generator_dropout=[0, 0, 0],
                          merge_mode='adain',

                          discriminator_layers=[64, 64],
                          discriminator_norm=['none', 'none'],
                          discriminator_dropout=[0, 0],

                          reconstruction_loss='MSE', adversarial_loss='MSE',
                          weight_reconstruction=1, weight_adversarial=1,
                          weight_representation=0.1,
                          weight_contrastive=0.1,
                          contrastive_temperature=0.05,

                          lr_G=0.0005, lr_D=0.0005, b1_G=0.9, b1_D=0.9,
                          lr_scheduler_G='none', lr_scheduler_D='none',

                          use_projection_head=False, projection_output_size=0,
                          batch_size=32, note="obsNum_" + str(obsNum))
```

1.4 Step 4: Create an instance of the `Trainer` class

This is done by calling `Trainer()` from `pytorch_lightning`. `Trainer` can help you train your LEOPARD. Here we use the following settings (please refer to its Docs for a comprehensive parameter explanation): - `enable_progress_bar`: show progress bar? Default: `True` - `log_every_n_steps`: a numeric value that specifies the interval, in steps, at which metrics should be logged. Default: `3` - `max_epochs`: a numeric value that defines the maximum number of epochs the training loop should run. Default: `100` - `gpus`: a value indicating which GPUs to use. Default: `1 if torch.cuda.is_available() else None` - `logger`: a tensorboard logger responsible for logging training/validation metrics and other experiment details.

[4]:
```
save_dir = os.path.join("lightning_logs", "trainNum_" + str(trainNum))
name = "obsNum_" + str(obsNum)

trainer = pl.Trainer(
    enable_progress_bar=False,
    log_every_n_steps=3,
    max_epochs=199,
    gpus=1 if torch.cuda.is_available() else None,
    logger=TensorBoardLogger(save_dir=save_dir, name=name)
)
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
```

1.5 Step 5: Train your LEOPARD

Now let's train your LEOPARD!

Optional: you can also visualize the training process with the logger. Use the `%tensorboard` magic command or call it in command line: `tensorboard --logdir *save_dir* --port 8080` (*use your own* ***save_dir*** *and port number*)

In tensorboard, you can monitor the losses computed on the training set and validation set (if you have one), which can help mitigate the risk of overfitting. For example, you might want to stop the training process if the reconstruction loss starts to increase.

[ ]:
```
# optional: invoke TensorBoard with the %tensorboard magic command.

%load_ext tensorboard
%tensorboard --logdir lightning_logs
```

[5]:
```
# train a LEOPARD

trainer.fit(my_leopard)
```

```
Missing logger folder: lightning_logs\trainNum_all\obsNum_0
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name                                | Type    | Params
-------------------------------------------------------------
0 | loss_F_reconstruction_noReduction | MSELoss | 0
1 | loss_F_reconstruction             | MSELoss | 0
2 | loss_F_adversarial                | MSELoss | 0
3 | leopard                           | LEOPARD | 134 K
-------------------------------------------------------------
134 K     Trainable params
0         Non-trainable params
```

```
134 K     Total params
0.538     Total estimated model params size (MB)
C:\Users\SiyuHan\anaconda3\envs\LEOPARD\Lib\site-
packages\pytorch_lightning\trainer\connectors\data_connector.py:245:
PossibleUserWarning: The dataloader, val_dataloader 0, does not have many
workers which may be a bottleneck. Consider increasing the value of the
`num_workers` argument` (try 16 which is the number of cpus on this machine) in
the `DataLoader` init to improve performance.
  category=PossibleUserWarning,
C:\Users\SiyuHan\anaconda3\envs\LEOPARD\Lib\site-
packages\pytorch_lightning\trainer\connectors\data_connector.py:245:
PossibleUserWarning: The dataloader, train_dataloader, does not have many
workers which may be a bottleneck. Consider increasing the value of the
`num_workers` argument` (try 16 which is the number of cpus on this machine) in
the `DataLoader` init to improve performance.
  category=PossibleUserWarning,
```

1.6 Step 6: Impute and export data

You can impute the missing data and writing them into a .csv file. If you have ground truth for "vB_t2_test.csv", you can also export percent bias of the imputed data.

[6]:
```
# impute data
imputed_data = trainer.predict(my_leopard, my_leopard.test_dataloader())[0]
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
C:\Users\SiyuHan\anaconda3\envs\LEOPARD\Lib\site-
packages\pytorch_lightning\trainer\connectors\data_connector.py:245:
PossibleUserWarning: The dataloader, predict_dataloader 0, does not have many
workers which may be a bottleneck. Consider increasing the value of the
`num_workers` argument` (try 16 which is the number of cpus on this machine) in
the `DataLoader` init to improve performance.
  category=PossibleUserWarning,
```

[7]:
```
# create folder for output
output_dir = os.path.join(save_dir, name, "version_" + str(trainer.logger.
 ↪version), "results")

if not os.path.exists(output_dir):
    os.mkdir(output_dir)

# export imputed data
pd.DataFrame(imputed_data["generated_data"]).to_csv(
            os.path.join(output_dir, "imputedData_obs" + str(obsNum) + ".csv"),␣
 ↪index=False
)
```

[ ]:
```
# export percent bias (only when groundtruth is available)
pd.DataFrame(imputed_data["raw_percentBias"]).to_csv(
            os.path.join(output_dir, "PB_obs" + str(obsNum) + ".csv"),␣
  ↪index=False
)
```

1.7 End

This manual is prepared based on our analysis and has been tested on our benchmark datasets. Please let us know if you found any issues.