

MASTERARBEIT | MASTER'S THESIS

Titel | Title

reComBat-seq: Removing batch effects of large datasets with
regularized negative binomial regression

verfasst von | submitted by

Zhasmina Stoyanova BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree
programme code as it appears on the
student record sheet:

UA 066 875

Studienrichtung lt. Studienblatt | Degree
programme as it appears on the student
record sheet:

Masterstudium Bioinformatik

Betreut von | Supervisor:

Univ.-Prof. Dipl.-Phys. Dr. Jörg Menche

Abstract

The rapid growth of RNA-seq studies has led to an explosion of omics data, offering new research opportunities but also posing computational challenges for accurate analysis. Correcting technical noise, known as batch effects, is especially important when integrating large datasets, as variations in sample processing across batches can obscure true biological signals and hinder meaningful comparisons.

Methods like *ComBat* (for microarray) and *ComBat-seq* (for count data) address batch correction via regression but struggle with large datasets containing many experimental conditions, resulting in sparse, high-dimensional design matrices. When batch designs are highly correlated, *ComBat* may fail to find unique solutions. To tackle this, *reComBat* extends *ComBat* by regularizing the regression with Elastic Net, enabling analysis of sparse, underdetermined data. However, *reComBat* assumes normality and does not preserve the original data distribution, limiting its use with skewed count data and downstream analyses.

This master's thesis investigates the mathematical foundations of Elastic Net regularization and applies it to the negative binomial regression model used in *ComBat-seq*, extending the method for datasets with underdetermined designs. The resulting tool, *reComBat-seq* (regularized *ComBat-seq*), is implemented in R based on the approach of *reComBat* and benchmarked against existing batch correction methods. We demonstrate the potential of *reComBat-seq* on simulated and real RNA-seq datasets and show how it enables the integration and analysis of underdetermined RNA-seq data. The method preserves the integer nature of count data, ensuring compatibility with common differential expression tools and achieves comparable performance to established methods, particularly in simpler experimental designs. With fine-tuning, *reComBat-seq* has the potential to extend its applicability to more complex designs, facilitating more reliable and scalable RNA-seq data integration.

Kurzfassung

Durch die stetig wachsende Zahl an RNA-seq-Studien (RNA-Sequenzierung) steht heute eine große Menge an Omics-Daten zur Verfügung, die vielfältige Forschungsmöglichkeiten bietet, jedoch auch bedeutende rechentechnische Herausforderungen bei der präzisen Analyse mit sich bringt. Die Korrektur technischer Störsignale, sogenannter Batch-Effekte, ist insbesondere bei der Integration großer Datensätze von entscheidender Bedeutung, da Unterschiede in der Probenverarbeitung zwischen Batches wahre biologische Signale überdecken und aussagekräftige Vergleiche erschweren können.

Methoden wie *ComBat* (für Microarray-Daten) und *ComBat-seq* (für Zählraten) korrigieren Batch-Effekte mithilfe von Regressionsansätzen, erreichen jedoch bei großen Datensätzen mit vielen experimentellen Bedingungen ihre Grenzen, da dünnbesetzte hochdimensionale Designmatrizen entstehen. Bei stark korrelierten Batch-Designs kann *ComBat* keine eindeutige Lösung finden. Um dieses Problem zu lösen, erweitert *reComBat* den Ansatz von *ComBat* durch Regularisierung mittels Elastic Net und ermöglicht so die Analyse von unterbestimmten Datensätzen. Da *reComBat* eine Normalverteilung voraussetzt und die ursprüngliche Datenverteilung verändert, ist seine Anwendbarkeit auf nicht normalverteilte Zählraten und Downstream-Analysen begrenzt.

Diese Masterarbeit untersucht die mathematischen Grundlagen der Elastic-Net Regularisierung und integriert sie in das negative binomiale Regressionsmodell von *ComBat-seq*, um auch Datensätze mit stark korrelierten Prädiktoren verarbeiten zu können. Das daraus resultierende Tool *reComBat-seq* (regularisiertes *ComBat-seq*) wurde in R basierend auf dem Ansatz von *reComBat* implementiert und gegen bestehende Batch-Korrekturmethoden getestet. Wir demonstrieren das Potenzial von *reComBat-seq* anhand simulierter und realer RNA-seq-Datensätze und zeigen, wie es die Integration und Analyse von unterbestimmten RNA-seq-Daten ermöglicht. Die Methode bewahrt die ganzzahlige Struktur der Zählraten, was die Kompatibilität mit verschiedenen Pipelines zur differentiellen Expression gewährleistet und erreicht eine vergleichbare Leistung zu etablierten Methoden, insbesondere bei einfacheren experimentellen Designs. Durch Fein-tuning könnte *reComBat-seq* sein Einsatzspektrum auf komplexere Designs ausweiten und so eine zuverlässigere und skalierbare Integration von RNA-seq-Daten ermöglichen.

Contents

Abstract	i
Kurzfassung	ii
1. Introduction	1
1.1. Molecular basis of differential gene expression	1
1.2. From microarrays to RNAseq	3
1.3. Differential expression analysis	4
1.3.1. Microarray data	6
1.3.2. RNA-seq	8
1.4. Batch effects	12
2. Batch correction tools	14
2.1. ComBat	14
2.1.1. Model assumptions	14
2.1.2. Implementation	15
2.1.3. Limitations	16
2.2. ComBat-seq	17
2.2.1. Model assumptions	17
2.2.2. Implementation	18
2.2.3. Limitations	20
2.3. reComBat	20
2.3.1. Model assumptions	20
2.3.2. Implementation	21
2.3.3. Limitations	21
3. Proposition: reComBat-seq	22
3.1. Motivation	22
3.2. Model assumptions	24
3.3. Data Preprocessing	25
4. Methods and materials	26
4.1. Implementation	26
4.2. Simulation study	27
4.2.1. Data simulation	29

Contents

4.2.2. Batch effect adjustment	31
4.2.3. Benchmarking	31
4.3. Real data study	33
5. Results	35
5.1. Simulations	35
5.1.1. Hyperparameter Study	35
5.1.2. Gene Count Study	37
5.1.3. Sample Count Study	39
5.1.4. Batch Count Study	41
5.2. Real data	43
6. Discussion	46
Bibliography	49
A. Appendix - Linear and Generalized Linear Models	54
A.1. Linear regression models	54
A.1.1. Model assumptions	54
A.1.2. Coefficient estimation methods	55
A.2. Generalized linear models (GLM)	55
A.2.1. Model assumptions	56
A.2.2. Maximum likelihood estimation methods	56
A.3. Elastic Net Regularization	61
B. Appendix - Empirical Bayes	64
C. Appendix - Negative Binomial Distribution	65
C.1. Model derivation	65
C.2. Maximum likelihood estimation	66
D. Appendix - Experiments Results	67
D.1. Gene count study	67
D.2. Sample count study	68
D.3. Real data	69

1. Introduction

The construction of the molecular components of an organism starts at the DNA. As the carrier of genetic information it provides the blueprints for the parts that make up cellular systems [Urry et al., 2021]. The process of using the information encoded in the DNA to create functional cellular products is known as gene expression. Comprised of multiple tightly regulated steps, it serves as link between genotype and phenotype. Identical genomes can give rise to distinct phenotypes through regulation of gene expression, translating the genetic information into diverse cell types and traits, with variations across its steps contributing to phenotypic diversity. Thus, these differences in gene expression can tell us a lot about the underlying biology of a specific phenotype and analysing them systematically can reveal condition-specific patterns of gene activity, identify over- or underexpressed genes linked to certain diseases, uncover dysregulated pathways and aid in early diagnosis [Urry et al., 2021].

This section covers the introductory themes and gives an overview of the main topics relevant for this project. It outlines the evolution of gene expression analysis tools and the different types of data they generate, with a focus on microarrays and RNA sequencing. The principles of differential gene expression analysis are explored, along with the biological relevance of differentially expressed genes. Lastly, the central problem of this thesis is introduced, highlighting the challenge of integrating data from multiple studies due to processing differences.

1.1. Molecular basis of differential gene expression

A gene is expressed when the information stored in the DNA is used to synthesize a cellular product, such as proteins or non-coding RNA.

DNA, short for deoxyribonucleic acid, is the molecule that stores genetic information in almost all living organisms. It is composed of four types of nucleotides and the specific sequence of these nucleotides encodes the instructions for building and regulating cellular components [Urry et al., 2021, 16.1]. This information is not used directly but is first transcribed into ribonucleic acid (RNA), which serves as the intermediary between DNA

1. Introduction

and proteins. Among different RNA types, only messenger RNA (mRNA) carries the genetic code needed for protein production. The resulting proteins serve as the main structural and functional components of the cell [Urry et al., 2021, 17.1].

Together, these steps constitute the central flow of genetic information, connecting DNA sequence to cellular function, known also as the central dogma of molecular biology. It states that the general direction of information transfer flows from DNA to RNA to protein [Crick, 1958]. Information transfer occurs via two main processes – transcription and translation.

Transcription

Transcription is the process by which mRNA molecules are synthesized from a DNA template, producing an RNA copy of the DNA sequence that carries the genetic information for protein production. This process involves several interacting components, including enzymes and proteins that help initiate, elongate and terminate the RNA synthesis. The activity of these components can be amplified, reduced or blocked. This regulation allows the cell to control which genes are expressed and to what extent, allowing cells with the same genome to develop distinct functional roles [Alberts et al., 2008].

Translation

Proteins are then synthesized from the information encoded in mRNA through translation, a process in which triplets of nucleotides are read to assemble amino acids into polypeptide chains. Translation efficiency is regulated at multiple steps, including initiation and elongation, which together determine how much protein is produced from a given mRNA [Brown and Clancy, 2008]. Because post-transcriptional regulation, including degradation of mRNA, is closely linked to translation, these processes together shape the mRNA levels that can be measured and influence how gene expression is observed [Huch and Nissan, 2014].

Through these processes, cells employ multiple regulatory mechanisms at transcriptional, post-transcriptional and translational levels to carry out their specific functions and maintain flexibility. This occurs not only by ensuring that the necessary components are expressed, but also by switching to alternative components when some of the standard ones are faulty. As a result, differential gene expression can be observed both under normal biological conditions and in response to malfunctions, such as those occurring in disease. Gene expression analysis allows us to quantify how cells adjust their molecular programs in both normal and disease conditions. A common approach to analysing the molecular basis of a disease is to compare the expression profiles of healthy and diseased cells to identify under- or overexpressed genes and examine how they change over time or in response to treatment. More specifically, this involves measuring the abundance of

1. Introduction

mRNA transcripts and applying statistical methods to determine whether the observed differences are significant compared to healthy cells [Karakach et al., 2010].

1.2. From microarrays to RNAseq

Measuring gene product abundance is central to assess differential gene expression. This can be done using traditional microarrays, which rely on binding to predefined sequences or newer RNA sequencing (RNA-seq) methods, which allow a more comprehensive and sensitive measurement of all transcripts in a cell.

Microarrays

Microarrays are a diagnostic technology that despite being an older and less powerful method, remain in use due to their cost effectiveness compared to newer generation sequencing methods. They are used to detect the expression of multiple different genes via DNA chips. The chips contain small DNA segments representing target genes of interest. To measure the concentration of those genes in studied cells, mRNA is extracted from the samples and reverse transcribed into complementary DNA (cDNA), labeled with fluorescent markers [Kellis et al., 2015]. The cDNA components are washed over the chip, binding to their complementary strands and triggering a fluorescence signal. The position of the signal marks the expressed gene. Using a photodetector the intensity of this signal can be measured and quantified [Termanini, 2023]. This enables measurement of the relative abundance of specific sequences across samples.

There are several ways to represent gene expression data, but it is most commonly structured as a matrix, where genes are organized according to samples, cells or experiments. In this work, we adopt the matrix representation, where rows correspond to genes and columns to samples. The intensity values are modelled as continuous values [Figure 1.1] [Torrenté et al., 2019].

	Sample 1	Sample 2	Sample 3	Sample 4
Gene 1	0.46	1.06	0.04	1.23
Gene 2	-0.2	-1.03	0.3	-0.29
Gene 3	0.15	0.32	1.43	-1.4
Gene 4	0.45	-0.65	0.8	-0.03

Figure 1.1.: An example of a microarray gene expression matrix, 4 genes and 4 samples.

While still in use, microarrays have several limitations. They can only detect sequences that they were designed to detect, as defined by the DNA probes on the microchip.

1. Introduction

Genes that have not been annotated are typically excluded. Additionally, multiple similar genes can bind to the same probe, introducing errors in the expression profiles [Bumgarner, 2013].

RNA sequencing

RNA Sequencing, or short RNA-seq, addresses many of the limitations of microarrays, offering higher sensitivity and broader range of detection.

At the core of this method are the high-throughput DNA sequencing technologies, such as next-generation sequencing (NGS) for short reads or third generation sequencing for long reads. These technologies have been adapted for RNA analysis, enabling accurate measurement of mRNA expression levels. Unlike microarrays, they are not limited to known sequences and are able to uncover novel transcripts [Deshpande et al., 2023]. The main working principle involves identifying the nucleotide sequence base by base, using methods such as chemical labeling or electrical signals [Urry et al., 2021].

As with microarrays, mRNA from samples is first cut into shorter fragments and then reverse transcribed into complementary DNA (cDNA). The resulting segments are amplified via a polymerase chain reaction (PCR) and ran through a sequencing platform, generating the reads. These reads are then reassembled based on sequence overlap or are aligned to a reference genome [Urry et al., 2021].

The final output is a gene expression matrix similar in structure to those generated in microarray studies, but based on non-negative integer count data, rather than continuous intensity values [Figure 1.2].

	Sample 1	Sample 2	Sample 3	Sample 4
Gene 1	46	34	1000	1404
Gene 2	0	1	30	23
Gene 3	150	342	198	500
Gene 4	15000	14987	3489	2350

Figure 1.2.: An example of an RNA-seq gene expression matrix, 4 genes and 4 samples.

1.3. Differential expression analysis

When conducting an RNA-seq or microarray experiment, we most often want to understand how a given condition differs from a control state. To address this, the resulting data matrices must be examined in order to distill biologically meaningful information, often by

1. Introduction

determining whether the abundance of a particular gene differs between the investigated condition and the control. Typically, these differences are assessed by employing statistical tests on the average expression value of a given gene across the conditional replicates. For instance, in the simplest case of comparing a single gene between a control and a treatment group, a standard t-test can be used to assess whether the mean expression levels are significantly different.

Although these tests are suitable for simple two-group comparisons, biological experiments often involve substantial noise and small sample sizes. In such cases, relying only on classical tests may result in reduced statistical power. To address this, more flexible approaches such as linear models are commonly employed. Linear models provide a unified framework for handling complex experimental designs involving multiple conditions and covariates, as well as the ability to borrow information across genes when estimating variability. This is particularly useful in RNA-seq and microarray studies where the number of samples per condition is often low.

In practice, linear modeling involves estimating the effect of a condition on the mean expression level of each gene and then testing whether the estimated coefficient for that condition is significantly different from zero. If the data provides sufficient evidence to reject the hypothesis that the coefficient equals zero, the gene is considered differentially expressed, meaning its abundance is affected by the treatment. This process is illustrated in Figure 1.3, which shows an example of a linear model fitting for a single gene across two experimental conditions.

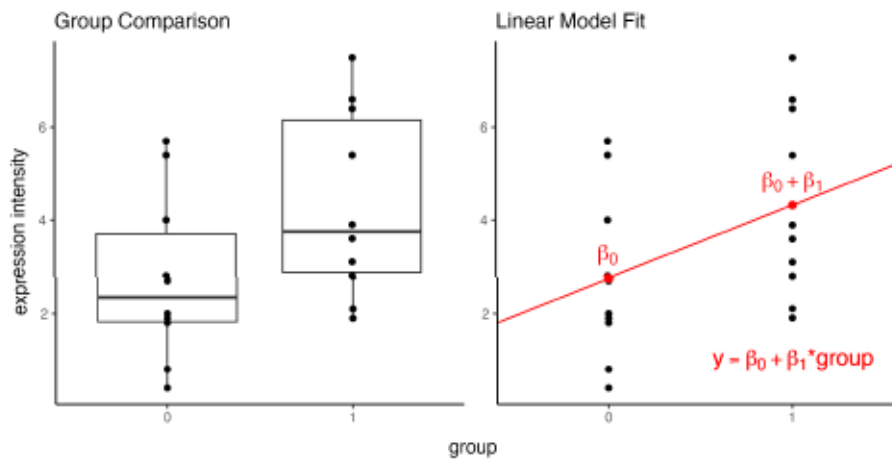


Figure 1.3.: Comparison of two groups using different approaches. **Left:** The boxplot shows the distribution of gene expression intensities between two groups (0 = control, 1 = treatment). **Right:** The same data represented using a linear model, where the intercept (β_0) corresponds to the mean expression in the control group and the slope (β_1) captures the difference in expression between groups.

1. Introduction

The validity of linear modelling also depends on certain assumptions, such as approximately normal residuals and equal variances across groups. Since raw expression data from RNA-seq or microarrays often do not meet these conditions, it is common to apply transformations like log-based normalisation to stabilise variance and make the data more suitable for statistical analysis. Addressing these challenges has led to the development of specialized software tools that implement statistical models, specifically adapted to the nature of high-throughput expression measurements

To this end, several software programs for R and Python have been developed with the purpose of identifying differentially expressed genes. These tools apply statistical tests to gene expression counts or intensity levels to determine which genes show significant differences between various conditions [Rosati et al., 2024]. This section outlines the mathematical foundations for the analysis of RNA-seq and microarray data, using three bioinformatics tools to illustrate a standard workflow - *edgeR* [Robinson et al., 2010], *DESeq2* [Love et al., 2014] and *limma* [Ritchie et al., 2015].

1.3.1. Microarray data

Microarray expression values are typically analyzed using linear models. Tools such as *limma* [Ritchie et al., 2015] operate on the expression matrix (Figure 1.1), fitting a linear model for each gene, represented by the rows of the matrix.

The data is normalized and log-transformed as a preprocessing step to make it more suitable for the assumptions of linear models, ensuring stable variances across groups. *limma* provides several normalization methods, such as *quantile normalization* or *cyclic loess*. Quantile normalization is the default method for simpler study designs and aims to standardize the distributions across samples by replacing each intensity value with the mean of its corresponding quantile [Bolstad et al., 2003].

Each row can be thought of as a vector of gene expression values $y_g = (y_{1g}, \dots, y_{ng})$ with y_{ig} being the log-transformed expression value for gene g , sample i and n amount of samples. The experimental design can be represented as a linear model for each gene under the following assumptions [Smyth, 2004, Ritchie et al., 2015]:

$$\begin{aligned}\mathbb{E}(y_g) &= X\alpha_g \\ \text{var}(y_g) &= \sigma_g^2/w_{ig}\end{aligned}$$

where X is the design matrix, associating each value y_g with a control or test group and α_g being the model coefficients. The term σ_g^2 denotes the residual variances, which are scaled by a weight matrix (w_{ig}) in order to account for variation in data quality [Ritchie et al., 2015, Smyth, 2004].

1. Introduction

For each gene g , an ordinary least squares (OLS) regression model is fitted to estimate α_g . This approach assumes normally distributed errors and the coefficients are obtained by minimizing the residual sum of squares [Davison, 2003]:

$$\hat{\alpha}_g = \arg \min_{\beta} \|y_g - X\alpha_g\|_2^2. \quad (1.1)$$

Details regarding the model fitting procedure are provided in Appendix A.

After the model is fitted, the residual variances s_g^2 and the coefficient estimators $\hat{\alpha}_g$ are estimated. Because the difference between coefficients is of interest, the model defines $\hat{\beta}_g = \hat{\alpha}_{g_{\text{control}}} - \hat{\alpha}_{g_{\text{study}}}$ as the object of analysis.

The gene-wise models can further share information between each other by employing global parameters, estimated by pooling the entire dataset. This is done to accommodate the typically small sample sizes in microarray experiments, where limited information per gene can hinder accurate variance estimation and increase the risk of false negatives. For this purpose *limma* utilizes empirical Bayes methods, introduced mathematically in Appendix B. Following this method a prior estimator s_0^2 with d_0 degrees of freedom is assumed, calculating it using the method of moments across all genes. Under that assumption a posterior $\tilde{s}$ can be calculated as:

$$\tilde{s}_g^2 = \frac{d_0 s_0^2 + d_g s_g^2}{d_0 + d_g}$$

The observed gene-wise variances are moderated by combining the individual variance estimates with the global prior variance, effectively pulling the values closer to a common, more stable estimate, stabilizing the inference even with a smaller number of arrays [Smyth, 2004, Ritchie et al., 2015]. Normal distribution [Smyth, 2004] is further assumed for the log-fold changes for differentially expressed genes $\hat{\beta}_g \neq 0$. Under this assumption and given $v_g = \text{var}(\hat{\beta}_g)$ the *moderated* test statistic is then calculated as:

$$t_g = \frac{\hat{\beta}_g}{\tilde{s}_g \sqrt{v_g}}$$

The null hypothesis $H_0 : \beta_g = 0$ is tested and p-values are obtained to assess the significance of the observed changes. If the hypothesis is rejected the gene can be assumed differentially expressed [Ritchie et al., 2015]. This process can be summarized in Algorithm 1.

1. Introduction

Algorithm 1 limma workflow

```

1:  $G \leftarrow$  normalized log-transformed gene expression matrix (genes x samples)
2:  $X \leftarrow$  design matrix (samples x groups)
3:  $n \leftarrow$  amount of genes
4:  $C \leftarrow$  contrast matrix

5: for  $g = 1, \dots, n$  do
6:   fit linear model:  $G[g,] \sim X$ 
7:    $\hat{\beta}_g \leftarrow (X^T X)^{-1} X^T G[g,]$  //coefficient estimator
8:    $s_g^2 \leftarrow$  residual variance estimator
9: end for

10:  $s_0^2 \leftarrow$  prior estimator

11: for  $g = 1, \dots, n$  do
12:   compute posterior variance  $\tilde{s}_g^2$ 
13:   compute moderated test statistics for each log-fold change
       
$$t_g = \frac{\hat{\beta}_g}{\tilde{s}_g \sqrt{v_g}}$$

14:   compute p-values  $p_g$ 
15: end for

16: apply multiple testing correction
17: return( $t_g, p_g$ , adjusted  $p_g$ , log-fold changes  $\hat{\beta}_g$ ) for each gene

```

1.3.2. RNA-seq

Unlike continuous microarray data, RNA-seq experiments yield non-negative integer counts, making the normal distribution unsuitable for modeling. The Poisson distribution is a common alternative due to its discrete nature, it assumes however that the mean and the variance are equal [Cameron and Trivedi, 1998]. RNA-seq has been shown to be overdispersed, with variance often greater than what would be expected for Poisson data [Figure 1.4][Zhou et al., 2011]. This suggests the usage of a negative binomial model, which relaxes the mean-variance relationship and accounts for overdispersion by defining the variance function as $\mu + \phi\mu^2 > \mu$, with ϕ being a dispersion parameter, controlling the shape. Appendix C provides an overview of the characteristics of the negative binomial distribution and a comparison between the distributions is illustrated in Figure 1.5.

Given a gene expression matrix (Figure 1.2), where each row $y_g = (y_{1g}, \dots, y_{ng})$ represents the expression counts of one gene across n samples, *edgeR* and *DESeq2* use the negative

1. Introduction

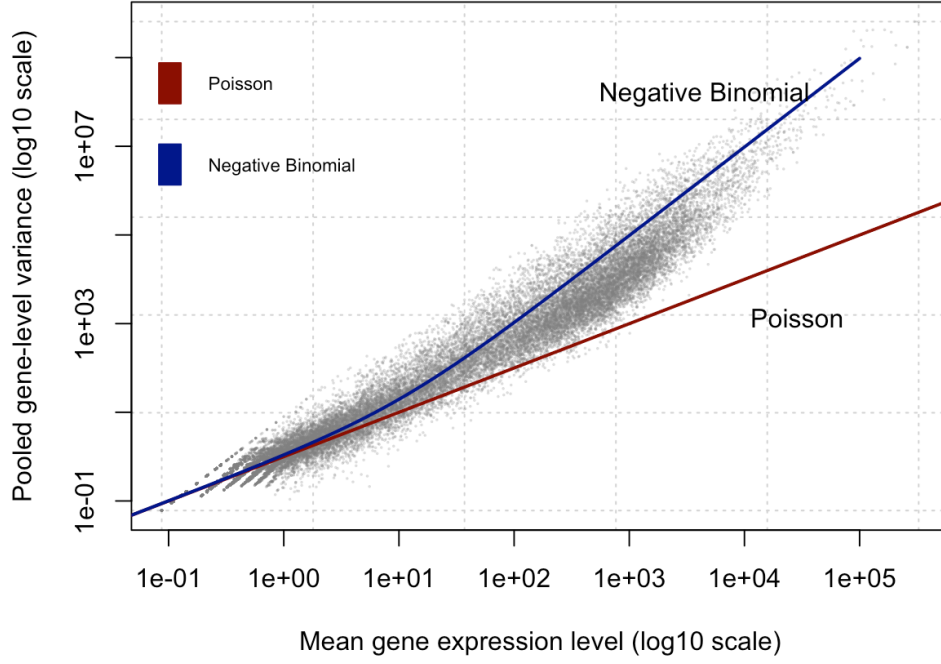


Figure 1.4.: Mean-variance relationship of the airway data from the R package *airway* [Himes et al., 2014].

binomial distribution [Robinson et al., 2010, Love et al., 2014]:

$$y_{gj} \sim NB(N_j p_{gi}, \phi_g)$$

for gene g and sample j . N_j refers to the total number of reads for sample j , known also as the library size. p_{gi} estimates the relative abundance of gene g in the group i , containing sample j . *edgeR* uses the parameterization with mean and variance equal to:

$$\begin{aligned}\mu_{gj} &= N_j p_{gi} \\ \sigma_{gj}^2 &= \mu_{gj}(1 + \mu_{gj}\phi_g)\end{aligned}$$

As briefly discussed, ϕ_g is a gene-specific dispersion parameter, capturing the variability beyond the Poisson model.

The data is normalized as a preprocessing step. Some of the methods that can be used are outlined here:

- Median of ratios (*DESeq2*) - the counts are divided by a normalization factor s , determined by the median ratio of the gene counts and the gene-wise geometric mean [Love et al., 2014].

1. Introduction

- TMM - Trimmed mean of M values (*edgeR*) - this method aims to normalize the counts under the assumption that the majority are not differentially expressed. The highly variable genes are trimmed off. The counts are then adjusted by a normalization factor, calculated using the weighted (trimmed) mean of the log expression ratios [Robinson and Oshlack, 2010].

Both *edgeR* and *DeSeq2* follow the same structure, where after the normalization, the gene-wise dispersions are estimated, employing empirical Bayes procedures to shrink each individual dispersion towards a pooled estimate [Robinson et al., 2010, Love et al., 2014]. In practice, this means that genes with imprecisely estimated dispersions are adjusted more strongly towards the global trend, whereas genes with reliable estimates retain values closer to their observed ones.

Once the gene-wise dispersions have been estimated, they serve as the basis for statistical testing. There are two main options when choosing an available test - an exact test and a quasi-likelihood (QL) F-Test. The former assumes the dispersion is fixed and known, while the latter accounts for uncertainty in the dispersion estimates.

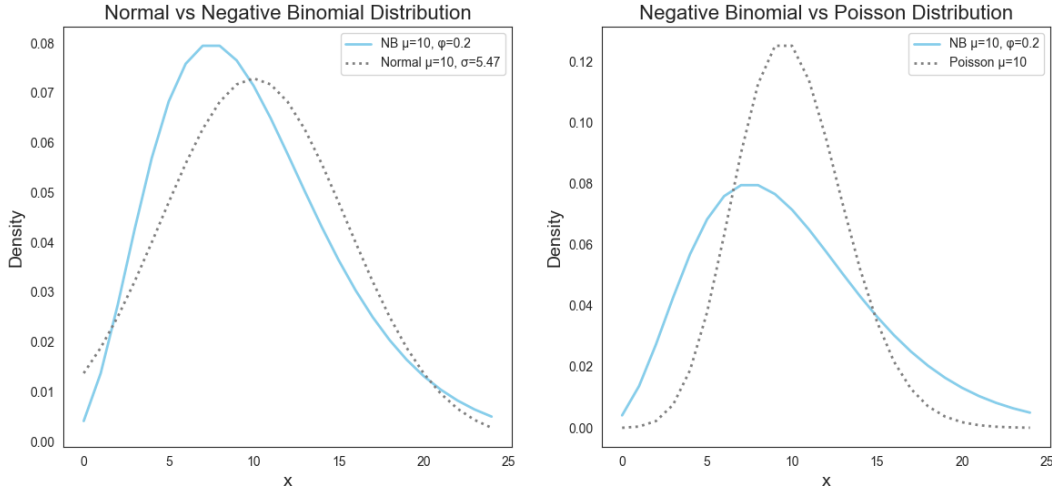


Figure 1.5.: Comparison of Normal, Poisson and Negative Binomial Distribution.

The exact tests are carried out as described in [Robinson and Smyth, 2008], being analogous to Fisher’s exact test but with negative binomial instead of hypergeometric probabilities. The null hypothesis $H_0 : q_{gA} = q_{gB}$ is examined with q_{gX} being the expression value for gene g and group X . The test statistic is formulated as the total counts for each group Z_{gX} and conditioning on the total sum $Z_{gS} = Z_{gA} + Z_{gB}$. The probability of observing a count split that is as or more extreme than the already observed is calculated. The p-value is obtained by summing up the probabilities of all possible values that have

1. Introduction

the same sum as $p(Z_{gA}, Z_{gB})$ but are more extreme [Anders and Huber, 2010].

$$p_g = \sum_{\substack{a+b=Z_{gS} \\ p(a,b) \leq p(Z_{gA}, Z_{gB})}} p(a, b) \quad / \quad \sum_{a+b=Z_{gS}} p(a, b)$$

A small p-value indicates a strong possibility that the gene is differentially expressed, based on the observed counts.

The quasi-likelihood F-Test version implements the methods of [Lund et al., 2012] for quasi-likelihood and [McCarthy et al., 2012] for the generalized linear model (GLM) fitting framework, incorporating uncertainty in the estimated variances.

GLMs provide a more appropriate framework for RNA-seq data than ordinary least squares regression. Unlike OLS, which assumes normally distributed errors with constant variance, GLMs allow the response to follow distributions from the exponential family and incorporate a link function to model the mean–variance relationship. For RNA-seq counts, the negative binomial GLM with a log-link is typically used, as it naturally accommodates the discrete nature of the data and the overdispersion commonly observed in gene expression measurements. Using the parameterization introduced above, the model can be expressed as:

$$\log \mu_{gj} = \log N_j p_{gi} = (X\beta_g)_j + \log N_j,$$

where X is the design matrix, encoding the covariates for each sample, and β_g is the vector of gene-specific regression coefficients, reflecting the estimated relative abundances. In simple terms, the relationship between the linear predictor $X\beta$ and the expected mean μ is logarithmic $\log \mu = X\beta$, or equivalently $\mu = e^{X\beta}$. Appendix A provides further details on the GLM fitting procedure.

Following the GLM framework, the quasi-likelihood test, in contrast to standard maximum-likelihood methods, treats the dispersion as an uncertain estimate. This is reflected in the process after the GLM is fitted and the common and gene-wise dispersions are estimated. The variance is assumed to be equal to the estimated variance multiplied by a dispersion proportionality constant Φ_g , capturing the unexplained variability after fitting the negative binomial model. This is illustrated in [Lund et al., 2012] as:

$$\text{var}(y_{gj}) = \Phi_g \mu_{gj} (1 + \mu_{gj} \phi_g),$$

where ϕ_g is the dispersion parameter of the negative binomial distribution, y_{gj} is the count number for gene g and sample j . Assuming F-distribution, the following test statistic is used with df as the degrees of freedom:

$$F_g = \frac{LR_g}{df \cdot \Phi_g}.$$

1. Introduction

This is an extension of the standard LR_g method, used in *edgeR*. LR_g refers to the likelihood ratio test, which compares the fit of two models based on the ratio of their likelihoods. The two models in this case are represented by the estimates of the mean under the null and under the alternative hypothesis.

$$LR_g = 2(\ell_g(\hat{\mu}_g|y_g) - \ell_g(\tilde{\mu}_g|y_g)),$$

with $\hat{\mu}$ and $\tilde{\mu}$ being the maximum quasi-likelihood estimates for the null and alternative hypothesis, respectively [Lund et al., 2012].

In summary, two models are fitted for each gene: a null model without the group effect and an alternative model including it. The likelihoods of these models are compared using the likelihood ratio, and the resulting statistic incorporates the gene-specific dispersion to account for uncertainty. Larger F would indicate larger likelihood ratio and larger deviance, indicating that a model with group effect would fit better and thus identifying a differentially expressed gene. The algorithm is summarized in Algorithm 2.

Algorithm 2 edgeR workflow

- 1: $G \leftarrow$ normalized gene expression matrix (genes x samples)
 - 2: $X \leftarrow$ design matrix (samples x groups)
 - 3: $n \leftarrow$ amount of genes
 - 4: **for** $g = 1, \dots, n$ **do**
 - 5: $\phi_g \leftarrow$ gene-wise dispersion
 - 6: shrink ϕ_g towards a common variance estimate
 - 7: fit generalized linear model: $G[g,] \sim X$
 - 8: $\Phi_g \leftarrow$ QL dispersion proportionality constant
 - 9: $LR_g \leftarrow$ likelihood ratio tests
 - 10: compute QL F-test $F_g = \frac{LR_g}{df \cdot \Phi_g}$
 - 11: $p_g \leftarrow$ p-values
 - 12: **end for**
 - 13: return p_g for each gene g
-

1.4. Batch effects

Assessing differential expression of a gene using the aforementioned methods relies on the assumption that observed differences in abundance are purely biological. In practice, however, this assumption is often violated. Data are frequently generated in batches, sometimes due to high costs, logistical constraints or the need to integrate newly generated data with previously collected datasets. These batches may be produced in different

1. Introduction

labs, at different facilities or at different times, introducing unwanted technical variability. These processing differences known as batch effects can obscure the underlying biological signal. This is especially pronounced when integrating data from different sources. Figure 1.6 illustrates a small example of a dataset consisting of 4 biological groups. Testing for differential expression on such a dataset is likely to capture batch-driven differences rather than true biological signal, making the results unreliable for interpreting underlying biology. It is therefore necessary to address and remove batch effects prior to assessing gene expression differences between conditions.

Addressing and correcting batch effects is the central theme of this thesis and an interlude to the next Chapter 2, which aims to provide an overview of three batch correction methods and the mathematical foundation behind them. Chapter 3 proposes an extension to one of these methods, covering the theoretical background and model assumptions. The methods and the implementation details are outlined in Chapter 4. The final two chapters focus on the results of the proposed tool, benchmarked against existing software, followed by a discussion of its limitations and potential further improvements.

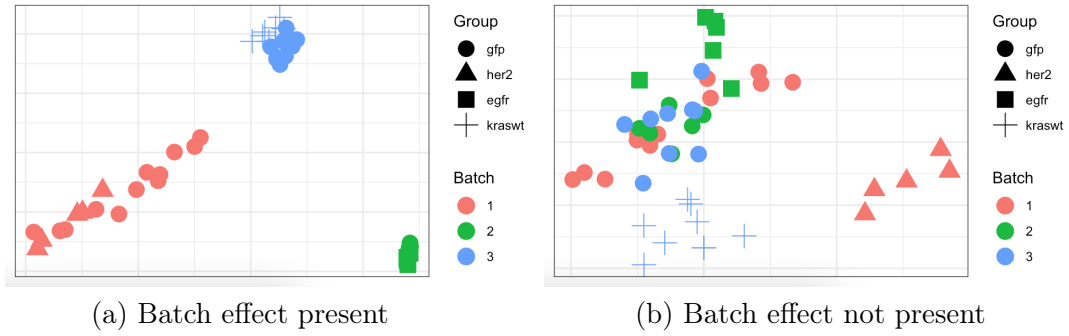


Figure 1.6.: The effects of batch processing on a small dataset of 50 samples and 4 groups. (a) Batch effect is present and causing the data to cluster by batch. (b) Batch effect not present - most of the variability comes from the biological properties.

2. Batch correction tools

A variety of software tools have been developed with the purpose of removing the suspected batch effects from the data, ensuring that the observed variation reflects the biological differences rather than technical noise. This chapter will focus on three of them from the *ComBat* family - *Combat* [Johnson et al., 2006] for microarray data, *ComBat-seq* [Zhang et al., 2020] for RNA-seq data and *reComBat* [Adamer et al., 2022] as an extension to *ComBat* for larger datasets.

2.1. ComBat

ComBat is a batch-correction software developed for R and has since been adapted for use in Python [Behdenna et al., 2023]. Its central assumption is that batch effects are either additive and/or multiplicative and can thus be modelled out of the data by standardizing means and variances across batches [Johnson et al., 2006]. To account for small sample sizes, additional empirical Bayes methods have been introduced, providing more robust estimators for the Location and scale (L/S) parameters.

2.1.1. Model assumptions

Building on top of the models defined in Section 1.3.1, *ComBat* assumes the following model for the expression values Y in a microarray experiment [Johnson et al., 2006]:

$$Y_{ijg} = \alpha_g + (\tilde{X}\beta^X)_{jg} + \gamma_{ig} + \delta_{ig}\epsilon_{ijg} = (X\beta)_{jg},$$

where respectively:

- Y_{ijg} - expression value for gene g for $g = 1, \dots, G$, sample j for $j = 1, \dots, n$, batch i for $i = 1, \dots, m$
- α_g - true gene expression for gene g

2. Batch correction tools

- X - full design matrix for sample conditions/ features with $\tilde{X}$ as the design matrix for desired variation
- β_g - regression coefficients corresponding to X for gene g with β_g^X as the regression coefficients for the desired variation
- γ_{ig} - additive batch effect for gene g and batch i
- δ_{ig} - multiplicative batch effect for gene g and batch i
- ϵ_{ijg} - error terms $\sim N(0, \sigma_g^2)$

The matrix in Figure 1.1 will be used as reference in illustrating the different components. In addition to the count matrix, a batch vector and optionally a covariate matrix have to be supplied, where each row describes one sample and its properties across different features. An example batch vector B and a group vector G assuming there are two batches and two groups can look like the following:

$$B = (1, 1, 2, 2) \quad G = (1, 2, 1, 2) \quad (2.1)$$

The design matrix is defined as a model matrix using both the batch vector and the covariates/ group vector. In this example the Group 2 column is left out, as it is a linear combination of the other three:

$$\begin{array}{c} \text{Batch 1} \quad \text{Batch 2} \quad \text{Group 1} \\ \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample 3} \\ \text{Sample 4} \end{array} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (2.2)$$

2.1.2. Implementation

ComBat has been implemented in the R software package *sva*. A Python distribution has been developed as well, as part of the *InMoose* tool collection package. The following section illustrates details regarding the implementation and methods used, as described in [Johnson et al., 2006].

Data Preprocessing

The data is cleaned as a preliminary step - this includes the removal of genes with zero variance as well as genes that are absent in more than 80% of the samples. It is then further standardized to place all genes on a comparable scale.

2. Batch correction tools

Fitting the L/S model and estimating the parameters

To estimate the model parameters $\hat{\alpha}_g, \hat{\beta}_g, \hat{\gamma}_{ig}$ a gene-wise ordinary least-squares approach (OLS) is employed, fitting the expression matrix to the full design matrix.

This solves the OLS equation $\hat{B} = (X^T X)^{-1} X^T Y$ and returns an estimate for the expression matrix $\hat{Y} = X \hat{B}$.

Standardizing the data

Following parameter estimation, we obtain the gene-wise mean along with the pooled variance, denoted $\hat{\sigma}_g^2$. The standardized data is assumed to follow a normal distribution $Z \sim N(\gamma_{ig}, \delta_{ig}^2)$ and is calculated by:

$$Z_{ijg} = \frac{Y_{ijg} - \hat{\alpha}_g - X \hat{\beta}_g}{\hat{\sigma}_g}.$$

Estimating the batch effect parameters using Empirical Bayes

The hyperparameters $\bar{\gamma}, \tau, \lambda, \theta$ for the prior distributions of the batch effect parameters are estimated from the standardized data using the method of moments.

$$\gamma_{ig} \sim N(Y_i, \tau_i^2) \quad \delta_{ig}^2 \sim \text{Inverse Gamma}(\lambda_i, \theta_i)$$

The new batch effect parameters are then calculated as the conditional posterior means, using Empirical Bayes inference (see Appendix B for more details):

$$\begin{aligned} \gamma_{ig}^* &= \mathbb{E}[\gamma_{ig} | Z_{ig}, \delta_{ig}^{2*}] = \frac{n_i \tau_i^2 \hat{\gamma}_{ig} + \delta_{ig}^{2*} \bar{\gamma}_i}{n_i \tau_i^2 + \delta_{ig}^{2*}} \\ \delta_{ig}^{2*} &= \mathbb{E}[\delta_{ig}^2 | Z_{ig}, \gamma_{ig}^*] = \frac{\theta_i + \frac{1}{2} \sum_j (Z_{ijg} - \gamma_{ig}^*)^2}{\frac{n_i}{2} + \lambda_i - 1}, \end{aligned}$$

with n_i as the amount of samples in batch i .

Adjusting the data

The standardized data is then normalized with respect to the estimated batch effects and calculated as:

$$Y_{ijg}^* = \frac{\hat{\sigma}_g}{\delta_{ig}^*} (Z_{ijg} - \hat{\gamma}_{ig}^*) + \hat{\alpha}_g + X \hat{\beta}_g$$

2.1.3. Limitations

ComBat introduces a flexible Empirical Bayes framework for removing additive and multiplicative batch effects from microarray data. It has several limitations, addressed in later extensions/ software. Some of them include:

2. Batch correction tools

- Lack of support for RNA-seq data and integer output → Addressed in *ComBat-seq* (Section 2.2). The original method assumes continuous, approximately normally distributed data, which is suitable for microarrays but incompatible with the discrete count nature of RNA-seq. Other methods such as *ComBat-seq* use negative binomial distribution to address this.
- Large-scale analysis of integrated datasets → Addressed in *reComBat* (Section 2.3). In high-dimensional settings, sparse or collinear design matrices can lead to rank deficiency, making the system underdetermined and complicating reliable estimation. *reComBat* addresses this by regularizing the OLS.

2.2. ComBat-seq

Similarly to the assumptions outlined in Sections 1.2 and 1.3.2, RNA-seq data require models that account for their discrete, overdispersed nature. Unlike microarray intensities, which use a Gaussian model, RNA-seq counts follow a different distributional pattern that must be reflected in the modeling framework. For this reason, the negative binomial distribution (see Appendix C) is employed in place of the classical bell-shaped Gaussian distribution. *ComBat-seq* builds on this foundation and incorporates methods from *edgeR* and *DESeq* to correct for batch effects in the data while maintaining the integer nature of count values. This also simplifies further downstream analysis, as many differential expression software packages require integer counts.

2.2.1. Model assumptions

ComBat-seq assumes the following genewise model for the expression values Y in an RNA-seq experiment [Zhang et al., 2020], given that $Y_{ijg} \sim NB(\mu_{ijg}, \phi_{ig})$:

$$\begin{aligned} \log \mu_{ijg} &= \alpha_g + (\tilde{X}\beta^X)_{jg} + \gamma_{ig} + \log N_j = (X\beta)_{jg} + \log N_j \\ \text{var } Y_{ijg} &= \mu_{ijg} + \phi_{ig}\mu_{ijg}^2 \end{aligned} \quad (2.3)$$

where respectively:

- Y_{ijg} - expression value for gene g for $g = 1, \dots, G$, sample j for $j = 1, \dots, n$, batch i for $i = 1, \dots, m$
- μ_{ijg} - mean parameter of the negative binomial distribution for gene g , sample j , batch i

2. Batch correction tools

- α_g - average gene expression for gene g
- X_j - full design matrix for sample condition/ features with $\tilde{X}_j$ as the design matrix for desired variation
- β_g - regression coefficients corresponding to X with β_g^X as the regression coefficients for $\tilde{X}$
- γ_{ig} - mean batch effect parameter for gene g and batch i
- ϕ_{ig} - dispersion batch effect parameter for gene g and batch i
- N_j - library size, total counts across all genes in sample j

The matrix in Figure 1.2 will be used as reference in illustrating the different components. In addition to the count matrix, analogous to Section 2.1, a batch vector and optionally a covariate matrix have to be supplied. The design matrix is then the model matrix of both the batch vector and the group vector. An example batch and group vector are shown in Equation 2.1 and a design matrix in Equation 2.2.

2.2.2. Implementation

ComBat-seq has been implemented in the R software package *sva*. A Python distribution has been developed as well, as part of the *InMoose* tool collection package. The following section illustrates details regarding the implementation and methods used, as described in [Zhang et al., 2020].

Data Preprocessing

The untransformed, raw count matrix is taken as input. Additionally, genes with all zero counts in any batch are removed.

Estimating the dispersion parameters

The dispersion parameters ϕ_{ig} are estimated using the Cox-Reid adjusted profile likelihood method from the *edgeR* package. It is based on maximizing the penalized log-likelihood [Chen et al., 2014]:

$$\text{APL}_g(\phi_g) = \ell(\phi_g; y_g, \hat{\beta}_g) - \frac{1}{2} \log \det(\mathcal{I}_g),$$

where ℓ is the log-likelihood function and $\mathcal{I}_g$ is the Fisher information matrix of β_g evaluated at $\hat{\beta}_g$ and ϕ_g .

2. Batch correction tools

The gene-wise dispersion values are estimated using empirical Bayes methods with the help of a common dispersion estimator across all genes. The common value is calculated as the average of the $\text{APL}_g(\phi_g)$ across all genes $g = 1, \dots, G$. The individual gene dispersions are estimated by a weighted likelihood method as an approximation to empirical Bayes [Chen et al., 2014].

Fitting the GLM model and estimating the batch effect parameters

The model parameters for the batch effects γ_{ig} and ϕ_{ig} are estimated by fitting the log-linear model defined in Equation 2.3 for each gene [McCarthy et al., 2012]. Following standard GLM theory [McCarthy et al., 2012] the model is fitted using the Fisher information matrix $\mathcal{I}_g = X^T W_g X$ with W_g being a GLM weight matrix, containing the second derivatives.

Parameter estimation is carried out by an IRLS (Iteratively Reweighted Least Squares) procedure, based on the Fisher scoring method, an iterative optimization algorithm, which is a variation of the Newton-Raphson algorithm. Detailed explanations about each algorithm can be found in Appendix A.2.2. At each iteration the coefficient vector β_g is updated in order to maximize the log-likelihood and converge toward the maximum likelihood estimate.

$$\beta_g^{\text{new}} = \beta_g^{\text{old}} + (X^T W_g X)^{-1} X^T W_g z_g,$$

where z_g is the model working response and $X^T W_g z_g$ represents the first derivative of the log-likelihood with respect to β_g .

Obtaining batch free distributions

The new parameters for the batch-free negative binomial distribution $NB(\mu_{jg}^*, \phi_g^*)$ are calculated as following, with m indicating the number of batches:

$$\begin{aligned} \log \mu_{jg}^* &= \log \hat{\mu}_{ijg} - \hat{\gamma}_{ig} \\ \phi_g^* &= \frac{1}{m} \sum_i \hat{\phi}_{ig} \end{aligned}$$

Adjusting the data

The adjustment method as explained in [Zhang et al., 2020] resembles quantile normalization and is carried out by mapping the empirical counts to the expected batch-free theoretical distribution.

First, the cumulative probability of a count with respect to the non-adjusted parameters is calculated:

$$p = F(Y_{jg} - 1 | \mu_{jg}, \phi_g),$$

where $F(Y_{jg}) = P(Y \leq Y_{jg})$ is the cumulative distribution function with respect to $NB(\mu_{jg}, \phi_g)$. This determines the relative position of the count in the distribution.

2. Batch correction tools

If $p \approx 1$ the old count is retained to avoid numerical instability, as that indicates the count is an extreme outlier. Otherwise, the new count is computed by mapping the quantile p to the new distribution:

$$Y_{ig}^* = 1 + F^{-1}(p|\mu_{jg}^*, \phi_g^*)$$

2.2.3. Limitations

ComBat-seq is specifically adapted to adjust batch effects in count data from RNA-seq studies. While it extends the original *ComBat* method to accommodate integer-valued counts, it also inherits the lack of scalability to large, integrated datasets. In such settings, the increasing number of samples and experimental factors often leads to high-dimensional or sparse design matrices, making the estimation unstable and computationally demanding. Strategies for regularization to guarantee unique and stable solutions are discussed in Chapter 3.

2.3. reComBat

reComBat has been developed as an extension to *ComBat* with the goal of accounting for a large amount of highly correlated biological covariates. It is implemented in Python, resulting in faster computation time compared to its R predecessors. It builds upon the basis outlined in Section 2.1, extending the linear model by adding regularization penalties to account for underdetermined systems [Adamer et al., 2022]. The implementation largely follows the Python version of *ComBat* [Behdenna et al., 2023], as well as the *scikit-learn* Python library.

2.3.1. Model assumptions

Extending the model defined in Section 2.1 *reComBat* assumes the following genewise model for the expression values Y in a microarray experiment [Adamer et al., 2022]:

$$Y_{ijg} = \alpha_g + (\tilde{X}\beta^X)_{jg} + (C\beta^C)_{jg} + \gamma_{ig} + \delta_{ig}\epsilon_{ijg} = (X\beta)_{jg}.$$

The model parameters are defined in the model 2.1.1 with *reComBat* introducing an additional design matrix C , representing unwanted sources of variation, and its associated coefficients β^C .

2. Batch correction tools

The matrix in Figure 1.1 can again be used as reference in illustrating the different components. An example batch and group vector are shown in Equation 2.1 and a design matrix in Equation 2.2.

2.3.2. Implementation

reComBat proposes a modification of the second step of the outlined *ComBat* workflow by fitting an elastic net model. The rest of the workflow follows the steps in Section 2.1.2.

Fitting the regularized model and estimating the parameters

In ordinary least squares (OLS) regression, coefficients are estimated by minimizing the sum of squared errors, but this can be unstable with correlated or high-dimensional predictors. The Elastic Net model expands on this and adds a strictly convex penalty function to the sum to ensure a unique solution to the system. It combines the feature selection property of LASSO with the continuous shrinkage of Ridge, leading to more stable and interpretable solutions. This is explored in more detail in Appendix A.3.

As outlined in [Adamer et al., 2022] the relevant model with λ_1 and λ_2 as the LASSO and ridge regularization penalties is denoted by:

$$\hat{\beta} = \arg \min_{\beta} \|y - X\beta\|_2^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2,$$

reComBat allows for separate regularization with the *Lasso* and *Ridge* models.

2.3.3. Limitations

reComBat offers a solution for large-scale integrated data subject to highly correlated experimental conditions. However, like the original *ComBat*, it does not natively support RNA-seq count data, which are discrete and often overdispersed, making standard assumptions of continuous Gaussian-like data invalid. This motivates the need for methods that extend batch correction strategies to count-based RNA-seq data while remaining suitable for large, complex datasets. Combining *reComBat*'s approach with a negative binomial framework to handle counts is addressed in the proposition explained in detail in Chapter 3.

3. Proposition: reComBat-seq

Although the latest iteration of the *ComBat* method feels more timely in an era of increasingly large and diverse datasets, its reliance on expression values being essentially continuous and normally distributed conflicts with the assumptions of modern RNA-seq differential expression methods. Despite *ComBat-Seq* overcoming this limitation for smaller datasets, there remains no appropriate solution for large-scale RNA-seq studies. To address this gap, this thesis proposes integrating the regularization approach of *reComBat* into the *ComBat-Seq* framework, providing a scalable method for batch correction in large RNA-seq datasets.

3.1. Motivation

In large-scale dataset integration the experimental conditions often result in high-dimensional, sparse design matrices. When the number of features exceeds the number of samples or when some features are highly correlated, the design matrix can become rank-deficient and the system underdetermined [Adamer et al., 2022]. The system may not have a unique solution, because of insufficient data to estimate the model.

Definition 1 (Linear Dependency [Liesen and Mehrmann, 2015]).

Let $\mathcal{V}$ be a vector space. The vectors $v_1, \dots, v_n \in \mathcal{V}$ are **linearly independent**, if and only if the equation:

$$\sum_{i=1}^n \lambda_i v_i = 0 \quad \text{with } \lambda_i \in \mathbb{R}$$

always implies that $\lambda_i = 0 \quad \forall i$. A matrix $A \in \mathbb{R}^{n \times n}$ has **linearly dependent** columns if there exists a vector $v \in \mathbb{R}^{n \times 1} \neq 0$ such that $Av = 0$.

Definition 2 (Rank of a matrix).

The **rank** of a matrix $A \in \mathbb{R}^{n \times p}$ refers to the dimension of the vector space generated by the columns. The following property applies: $\text{rank}(A) \leq \min(n, p)$. A matrix is said to be **rank-deficient** if it does not have full rank: $\text{rank}(A) < \min(n, p)$. This implies that

3. Proposition: reComBat-seq

at least one column of A can be expressed as a linear combination of the others, i.e. the columns are linearly dependent.

Definition 3 (Singularity [Katznelson and Katznelson, 2008]).

A square matrix A is called **invertible**, if and only if there exists a matrix $A^{-1} \in \mathbb{R}^{n \times n}$ such that $AA^{-1} = A^{-1}A = I_n$, with I_n being the identity matrix. A non-invertible matrix is called **singular**.

A necessary and sufficient property of an invertible matrix is that it is of full rank, as described in [Katznelson and Katznelson, 2008][Chapter II: Linear operators and matrices].

Proposition 1.

If $A \in \mathbb{R}^{n \times p}$ is non-invertible and $W \in \mathbb{R}^{n \times n}$ is an arbitrary matrix, then $A^T W A$ is also non-invertible.

Proof. Let $A \in \mathbb{R}^{n \times p}$ be non-invertible and thus not of full rank. By definition, the columns of A are linearly dependent. Therefore, there exists a nonzero vector $v \in \mathbb{R}^{p \times 1}, v \neq 0$ such that:

$$Av = 0$$

The following can be shown for any matrix $W \in \mathbb{R}^{n \times n}$:

$$(A^T W A)v = A^T W (Av) = A^T W \cdot 0 = 0.$$

Since $v \neq 0$ and $(A^T W A)v = 0$, it follows that the columns of $A^T W A$ are also linearly dependent. $A^T W A$ is not of full rank $\Rightarrow$ therefore non-invertible. $\square$

In summary, if the columns of a matrix are linearly dependent, the matrix becomes singular and cannot be inverted. Returning to the fitting procedure, *ComBat-Seq* employs a GLM to model gene expression. Following standard GLM theory, parameter estimation involves solving

$$\hat{\beta}_g = (X^T W_g X)^{-1} X^T W_g z_g,$$

where $X^T W_g X$ is the Fisher information matrix with W_g being a weight matrix, containing the second derivatives. $X^T W_g z_g$ represents the first derivative of the log-likelihood function with respect to β_g (The full derivation can be found in Appendix A). The algorithm therefore relies on finding the inverse of the Fisher information matrix to estimate the parameters. From Proposition 1 follows however that if X is singular, then $X^T W_g X$ will also be singular and thus non-invertible. Following *reComBat*'s approach, this work develops methods to handle singular matrices for batch correction in large-scale RNA-seq experiments.

3.2. Model assumptions

Following the *ComBat-seq* implementation, it can be assumed that the gene counts Y follow a negative binomial distribution, as detailed in (2.3). The gene-wise negative binomial parameters μ_g and ϕ_g can be estimated using maximum likelihood methods. Recall the simplified form of model 2.3:

$$\log \mu_g = X\beta_g + \log N \Rightarrow \mu_g = e^{X\beta_g + N}.$$

In *ComBat-seq* the parameters μ and ϕ are then computed by finding the minimum of the negative log-likelihood function [Hilbe, 2011] (full derivation of the function in Appendix C and of the fitting method in Appendix A.2.2):

$$\begin{aligned} NLL_g(\beta, \phi_g) = -\ell_g(\beta, \phi_g) = & -\sum_{j=1}^n \log \Gamma\left(y_{jg} + \frac{1}{\phi_g}\right) - \log \Gamma(y_{jg} + 1) - \log \Gamma\left(\frac{1}{\phi_g}\right) + \\ & y_{jg} \log \phi_g + y_{jg}(\log \mu_{jg}) - \left(y_{jg} + \frac{1}{\phi_g}\right) \log(1 + \phi_g \mu_{jg}) \end{aligned}$$

This thesis proposes a modified version of the *ComBat-seq* algorithm, using a regularized objective function, instead of the simple log-likelihood function. Due to this modification and borrowing from *reComBat* this approach will be called *reComBat-seq*.

Elastic net regularization introduces penalties analogous to the ordinary least squares regression and ensures the strict convexity of the system. Let $\lambda > 0$ be the regularization strength parameter and $\alpha > 0$ the LASSO/ridge tuning parameter. Assuming that ϕ has already been estimated using the Cox-Reid APL (See Section (2.2.2)), the regularized estimate of β can then be defined as [Zhang and Jia, 2022]:

$$\hat{\beta} = \arg \min_{\beta} \left\{ NLL(\beta) + \lambda \left(\alpha \|\beta\|_1 + \left(\frac{1-\alpha}{2} \right) \|\beta\|_2^2 \right) \right\} = \arg \min_{\beta} \{NLL_{\text{reg}}\}.$$

Here, the l_q norm of β is defined as $\|\beta\|_q = (\sum_{i=1}^p |\beta|^q)^{1/q}$ for $q \geq 1$ with p being the number of factors. The parameter α determines the relative weight of each regularization method: $\alpha = 0$ corresponds to pure Lasso, $\alpha = 1$ to pure Ridge, and values in between yield a mixture of the two as desired. For example, $\alpha = 0.5$ gives an equal balance of Lasso and Ridge in the Elastic Net. More information about elastic net regularization can be found in Appendix A.3.

The coefficients are estimated using standard MLE procedures, as described in Section 2.2.2 and Appendix A.2.2, aiming to minimize the new cost function denoted by NLL_{reg} . The new estimates are calculated using an *IRLS* fitting algorithm, based on *Fisher scoring*

3. Proposition: reComBat-seq

methods. The iterative update step in *reComBat-seq* is reformulated as:

$$\beta^{n+1} = \beta^n - (\mathcal{I}(\beta))^{-1} \left(\frac{\partial NLL_{\text{reg}}(\beta^n; y)}{\partial \beta} \right),$$

with $\mathcal{I}$ being the *expected Fisher information* matrix.

The fitting algorithm and steps remain the same as in *ComBat-seq* (Section 2.2.2). The derivatives of the penalties are calculated as follows:

$$\begin{aligned} \frac{\partial}{\partial \beta_k} \lambda \left(\alpha \|\beta\|_1 + \left(\frac{1-\alpha}{2} \right) \|\beta\|_2^2 \right) &= \lambda \frac{\partial}{\partial \beta_k} \left(\alpha \sum_{i=1}^p |\beta_i| + \left(\frac{1-\alpha}{2} \right) \sum_{i=1}^p |\beta_i|^2 \right) = \\ &= \lambda (\alpha \operatorname{sgn} \beta_k + (1-\alpha) \beta_k) \end{aligned}$$

$$\frac{\partial^2}{\partial \beta_l \partial \beta_k} \lambda \left(\alpha \|\beta\|_1 + \left(\frac{1-\alpha}{2} \right) \|\beta\|_2^2 \right) = \frac{\partial}{\partial \beta_l} \lambda (\alpha \operatorname{sgn} \beta_k + (1-\alpha) \beta_k) = \begin{cases} (1-\alpha) & k = l \\ 0 & k \neq l \end{cases}$$

The updated gradient and *Fisher* matrix can be defined as follows (full derivation for the non-penalized objective function in Appendix A.2.2):

$$\begin{aligned} \left(\frac{\partial NLL_{\text{reg}}(\beta^n; y)}{\partial \beta} \right)_k &= - \sum_{j=1}^n \frac{x_{jk}(y_j - \mu_j)}{1 + \phi \mu_j} + \lambda (\alpha \operatorname{sgn} \beta_k + (1-\alpha) \beta_k) \\ (\mathcal{I}(\beta))_{kk} &= \sum_{j=1}^n x_{jk} x_{jl} \frac{\mu_j}{1 + \phi \mu_j} + (1-\alpha). \end{aligned}$$

3.3. Data Preprocessing

In addition to modifying the objective function, the data preprocessing steps in the original *ComBat-seq* implementation were relaxed to allow for more flexibility. Specifically Step **Data Preprocessing** of the original workflow removes genes with all zero counts in any batch. While this approach ensures numerical stability during parameter estimation, it can lead to the exclusion of a substantial portion of genes, particularly in datasets with a large number of batches. To address this, the preprocessing step was removed, leaving it to the user to determine appropriate filtering criteria based on their specific use case and downstream analysis needs.

4. Methods and materials

4.1. Implementation

The *reComBat-seq* tool is implemented in R and uses the *Combat-seq* [Zhang et al., 2020] and *edgeR* 3.39 [Robinson et al., 2010] libraries as its foundation. The core computational routines of the main fitting function *glmfit* in *edgeR* are implemented in C++ and integrated in R using the Rcpp framework. The *reComBat-seq* tool is available at <https://github.com/jas-st/reComBat-seq>.

The same approach was applied to the Python version as well. While *pyComBat-seq* [Behdenna et al., 2023] does not natively support rank-deficient design matrices, it includes a damping parameter to prevent the algorithm from failing in the presence of singularities. However, once regularization was introduced, the algorithm’s performance degraded significantly. Rather than quickly falling back to damping and skipping most iterations — as it did in the unregularized case — the regularized version attempted to complete all iterations, resulting in a substantial slowdown. In contrast to the R implementation, the core fitting logic in the Python version is implemented entirely in Python rather than C++. Due to this limitation, the focus of this thesis will remain on the R version and its corresponding results, as further optimization and fine-tuning are required for the Python version.

To incorporate regularization, *reComBat-seq* modifies the core fitting function by extending the objective function to include the elastic-net penalties. Specifically, the regularized version alters both the gradient and the Fisher information matrix: first derivatives of the penalty terms are added to the gradient and second derivatives are incorporated into the diagonal of the Fisher information matrix. These changes enable elastic net-style penalization while maintaining the structure of the *IRLS*-based optimization routine.

Internally, *edgeR* computes the matrix product $X^T W X$ using the function `compute_xtwx`. *reComBat-seq* appends the penalty terms to the diagonals of this matrix, consistent with the derivations presented in Chapter 3, as the off-diagonal components of the regularization terms are zero.

Algorithm 3 update XtWX with penalties

Require: `ncoefs` - number of coefficients β
`xtwx` - `compute_xtwx()`
 λ - strength of regularization
 α - LASSO/ridge tuning

- 1: **for** $k = 1, \dots, \text{ncoefs}$ **do**
- 2: `xtwx`[$k \cdot \text{ncoefs} + k$] += $\lambda(1 - \alpha)$
- 3: **end for**

The gradients are updated with the corresponding penalty terms within an inner loop that computes the first derivatives for each coefficient individually. The notations used follow those established in the model definition in Chapter 3.

Algorithm 4 update gradient with penalties

Require: `ncoefs` - number of coefficients β
`nlib` - number of libraries/ samples
 λ - strength of regularization
 α - LASSO/ridge tuning

- 1: // Setting up the derivative vector U
- 2: **for** $j = 1, \dots, \text{nlib}$ **do**
- 3: $U[j] \leftarrow \frac{Y[j] - \mu[j]}{1 + \mu[j]\phi[j]}$
- 4: **end for**
- 5: // Setting up the gradient dl
- 6: **for** $k = 1, \dots, \text{ncoefs}$ **do**
- 7: $dl[k] = \sum_{j=1}^{\text{nlib}} U[j] \cdot X[j, k]$
- 8: $dl[k] -= \lambda(\alpha \text{sgn}\beta_k + (1 - \alpha)\beta_k)$
- 9: **end for**

4.2. Simulation study

The performance of *reComBat-seq* was evaluated by performing a series of simulation experiments, borrowing from the design used in [Zhang et al., 2020] and [Tran et al., 2020]. Four main experiments were designed with the goal of comparing the performance of this tool to the already established methods. First, a hyperparameter study was carried out to examine the effect of the regularization tuning parameters on the overall performance. Subsequent experiments evaluated performance across key dataset characteristics: number

4. Methods and materials

of genes, number of samples and number of batches. In each case, one variable was varied while the others were held constant to isolate its effect on performance. An overview of the experimental design is outlined in Table 4.1. Code for reproducing the experiments can be found at https://github.com/jas-st/reComBatseq_Studies.

Each simulation experiment follows a structured pipeline consisting of three main steps: data simulation, batch effect adjustment and benchmarking, described in detail in this section. Figure 4.1 summarizes the design of the simulation pipeline.

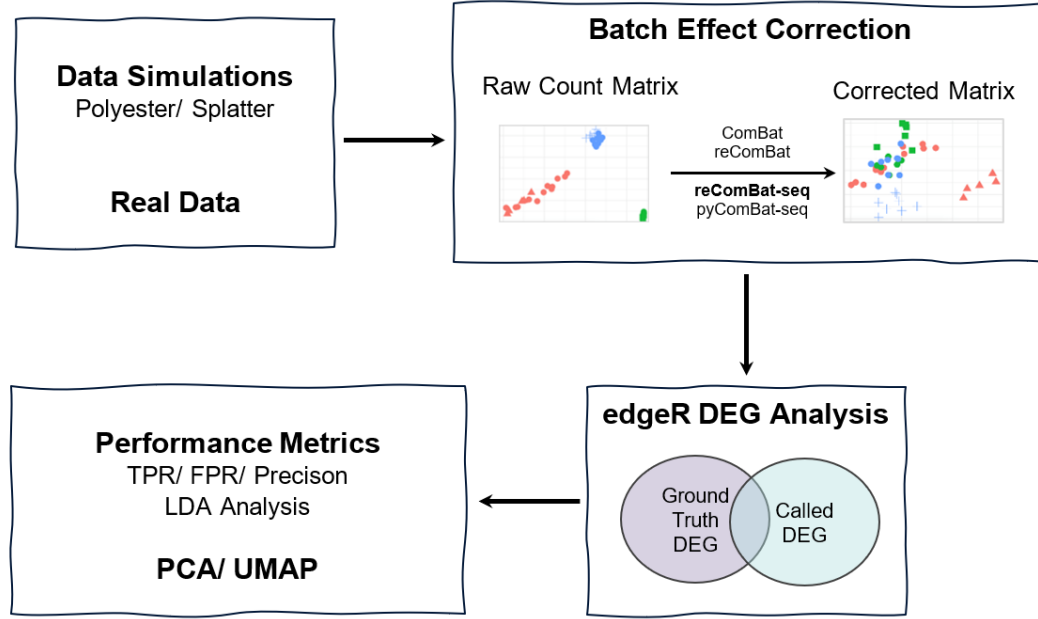


Figure 4.1.: Pipeline implementation, following the design introduced in [Tran et al., 2020]

1. **Data simulation** - Synthetic RNA-seq datasets are generated under controlled conditions. These datasets include known biological variation corresponding to different experimental groups, as well as artificially introduced batch effects. Detailed procedures are described in Section 4.2.1.
2. **Batch effect adjustment** - The next step involves applying the proposed method *reComBat-seq* to the simulated datasets in order to remove the unwanted variation. For comparison, several well-established batch correction methods including *reComBat*, *ComBat-seq* and *pyComBat-seq* are also applied to the same datasets. Further details are provided in Section 4.2.2.
3. **Benchmarking** - Finally, the corrected datasets are evaluated against the known

4. Methods and materials

ground truth using quantitative metrics. Additionally, results are compared across methods (see Section 4.2.3).

Table 4.1.: Overview of simulation experiment designs and variable settings
Each column corresponds to an experiment type in which one variable of interest is varied while others are held constant. The "Study variable" row indicates the main parameter explored in each experiment. The "Range" row provides the values over which the study variable is varied, while the "Fixed variables" row lists the parameters that are held constant.

Experiment design				
	Hyperparameter study	Gene count study	Sample count study	Batch count study
Study variable	α, λ	G	n	m
Range	$\alpha \in (0, 0.9)$ $\lambda \in (0.0001, 1)$	$G \in (128, 20171)$	$n \in (16, 16384)$	$m \in (2, 375)$
Fixed variables	$G = 128$ $n = 128$ $m = 2$	$\lambda = 0.8$ $\alpha = 0.3$ $n = 1024$ $m = 2$	$\lambda = 0.8$ $\alpha = 0.3$ $G = 5000$ $m = 2$	$\lambda = 0.8$ $\alpha = 0.3$ $G = 1024$ $n = 6000$

4.2.1. Data simulation

Two different simulation strategies were employed based on the specific requirements of each experiment. All simulations were generated with two underlying biological groups.

4.2.1.1. Two-batch simulation

polyester is an R package, designed for simulating RNA experiments. Given a set of annotated transcripts, it replicates the fragmentation and sequencing steps in order to generate raw RNA-seq reads. It supports simulation of a case-control design with biological replicates, allowing users to specify both the transcripts and the magnitude of differential expression [Frazee et al., 2015].

We adapt the generating function from *polyester* to allow for data simulation by directly sampling transcript lengths, which serve as baseline expression values. Ten percent of these genes are randomly chosen as truly differentially expressed (DE), evenly split into up-regulated and down-regulated subsets, while the rest remain non-DE. A two-group experimental design is implemented, where certain genes have higher expression in one batch and lower in the other, introducing a batch effect in the composition of expression that cannot be fully corrected by standard normalization [Zhang et al., 2020].

4. Methods and materials

To model both biological and technical variation, two fold change matrices are constructed (Figure 4.2):

- Base fold matrix - encodes differential expression between the two groups without batch effects.
- Batch fold matrix - incorporates batch-specific scaling factors that modify both DE and non-DE genes.

$$\begin{array}{c} \text{Gene 1} \\ \text{Gene 2} \\ \text{Gene 3} \\ \text{Gene 4} \end{array} \begin{pmatrix} \text{Batch 1} & \text{Batch 1} & \text{Batch 2} & \text{Batch 2} \\ 1 & 1 & 1 & 1 \\ 1 & 1.5 & 1 & 1.5 \\ 1 & 1 & 1 & 1 \\ 1.5 & 1 & 1.5 & 1 \end{pmatrix}$$

(a) Base fold matrix

$$\begin{array}{c} \text{Gene 1} \\ \text{Gene 2} \\ \text{Gene 3} \\ \text{Gene 4} \end{array} \begin{pmatrix} \text{Batch 1} & \text{Batch 1} & \text{Batch 2} & \text{Batch 2} \\ 2 & 2 & 1 & 1 \\ 2 & 3 & 1 & 1.5 \\ 1 & 1 & 2 & 2 \\ 1.5 & 1 & 3 & 2 \end{pmatrix}$$

(b) Batch fold matrix

Figure 4.2.: Base (a) and batch-affected (b) fold change matrices. Columns 1 and 3 represent control samples, while column 2 and 4 represent corresponding case samples. Gene 2 is up-regulated, while Gene 4 is down-regulated.

These matrices act multiplicatively on the baseline expression values, so that for each gene g in sample i , the expected mean is calculated as the element-wise product:

$$\mu_{ig} = \text{baseline}_g \times \text{fold change}_{ig},$$

Dispersion values ϕ_{ig} are similarly scaled to generate gene- and batch-specific dispersion estimates.

Finally, read counts are generated by sampling from the negative binomial distribution,

$$Y_{ig} \sim \text{NB}(\mu_{ig}, \phi_{ig}),$$

producing two datasets: one containing only biological signal, serving as the ground truth, and one affected by batch effects. Additionally, a fixed random seed is used to ensure comparability between the two. This approach follows the design used in *ComBat-seq* [Zhang et al., 2020], explicitly modeling variation at the parameter level rather than by adding noise directly to observed counts.

4. Methods and materials

4.2.1.2. Multi-batch simulation

The *polyester* approach relies on a fixed two-batch design and is not easily extendable to scenarios with more than two batches, which motivated the use of the more flexible *splatter* package for simulating multiple batch effects [Zappia et al., 2017]. Like *polyester*, *splatter* generates counts by sampling from a negative binomial distribution, with gene-wise baseline expression values scaled by fold-change factors. However, *splatter* can generate datasets with multiple batches and more complex experimental designs, providing greater flexibility for simulating realistic RNA-seq datasets.

Simulation parameters were initially estimated from the airway [Himes et al., 2014] dataset. The batch effect factors were drawn from a normal distribution with $\mathcal{N}(\mu = 2.5, \sigma^2 = 2.5)$. The datasets were then generated using a framework similar to the one outlined above, sampling read counts from the negative binomial distribution to produce both ground-truth and batch-affected datasets.

4.2.2. Batch effect adjustment

The next step of the pipeline involves applying the newly developed method alongside existing batch correction tools to the simulated datasets. For each dataset containing batch effects, four versions of corrected count data were produced: one using *ComBat-seq*, one using *reComBat-seq* with elastic net penalties, one using the Python-adaptation *pyComBat-seq* and one using the Python-based *reComBat* framework with ridge regularization.

To evaluate the robustness of each method to complex and potentially ill-conditioned designs, a rank-deficient covariate matrix was constructed in order to introduce structured confounding between covariates. It was supplied as input to both the regularized *reComBat-seq* and *reComBat*, as well as the modified *pyComBat-seq*. The standard *ComBat-seq* method, which does not accommodate singular matrices, was applied without it. For *ComBat-seq* and *pyComBat-seq*, which normally do not allow singular matrices, this restriction was removed to enable testing with the rank-deficient design. Execution time was recorded for each method to assess computational efficiency.

4.2.3. Benchmarking

To assess the effectiveness of each batch correction method, two complementary benchmarking strategies were employed: gene-level differential expression (DE) analysis and sample-level classification using Linear Discriminant Analysis (LDA).

4. Methods and materials

4.2.3.1. Differential Expression Analysis

DE performance was evaluated by comparing the set of genes identified as differentially expressed after correction to a ground-truth DE gene set extracted from the batch-free simulated data. For count-based methods (*ComBat-seq* and *reComBat-seq*), DE analysis was performed using the *edgeR* [Robinson et al., 2010] package. A standardized pipeline, *edgeR_DEpipe* [Zhang et al., 2020], was used to perform TMM normalization, estimate dispersion under a quasi-likelihood negative binomial model and conduct group comparisons. P-values were adjusted for multiple testing using the Benjamini–Hochberg procedure to control the FDR, with a significance threshold set at 0.01.

Unlike *ComBat-seq*, *reComBat* operates on log-transformed or normalized data and outputs continuous values. As a result, standard DE pipelines designed for count data (like *edgeR*) are not directly applicable. Instead, the *limma* package was used to perform DE analysis on the corrected *reComBat* output.

The resulting DE gene calls were compared against the known ground truth and evaluated using true positive and false positive rates.

True Positive Rate (TPR) - The proportion of true DE genes that were correctly identified by the method. It is computed as:

$$\text{TPR} = \frac{|\{\text{called}\} \cap \{\text{truth}\}|}{N_{\text{DE}}}$$

False Positive Rate (FPR) - The proportion of non-DE genes that were incorrectly identified as DE. It is calculated relative to the number of truly non-DE genes:

$$\text{FPR} = \frac{|\{\text{called}\} \setminus \{\text{truth}\}|}{N_{\text{nonDE}}}$$

Precision - The proportion of called DE genes that are truly DE:

$$\text{Prec} = \frac{|\{\text{called}\} \cap \{\text{truth}\}|}{|\{\text{called}\}|}$$

Here, N_{DE} indicates the number of differentially expressed genes and N_{nonDE} - the number of non-differentially expressed genes. The $\{\text{called}\}$ set describes the set of all genes, identified as DE by the corresponding method, while $\{\text{truth}\}$ - the set of all true DE genes.

4. Methods and materials

4.2.3.2. Linear Discriminant Analysis (LDA)

In addition to assessing biological signal recovery, LDA was used to evaluate the extent to which batch effects remained in the corrected datasets. This analysis was implemented in Python using the `scikit-learn` library [Pedregosa et al., 2011, Buitinck et al., 2013]. For each dataset, a linear discriminant classifier was trained to predict batch labels from the gene expression matrix.

The classifier assumes that the samples follow a multivariate Gaussian distribution within each batch, with all batches sharing the same design matrix. Using Bayes' rule, the classifier computes the posterior probability of each batch for a given sample, and the batch with the highest probability is selected as the prediction.

Here, a high LDA score indicates that batch structure remains in the data (i.e. samples are still easily separable by batch), while a low LDA score suggests effective batch effect removal, as batch labels become harder to predict from the corrected expression values. In this context, a score approaching $1/m$, where m is the number of batches, reflects random guessing, indicating that batch information is no longer present in the data.

4.3. Real data study

To test the *reComBat-seq* approach in a realistic setting, the method was applied to a real RNA-seq dataset, consisting of transcriptomic profiles from samples representing a variety of diseases, including cancer, collected across multiple tissue types. The full preprocessed dataset consists of 59 552 genes across 14 112 samples spanning 386 distinct batches. For each tissue subset, raw count matrices were extracted from `.h5ad` files, and relevant metadata, such as batch assignment and disease, was aligned with the expression data. An overview of the data design by tissue can be seen in Figure 4.3.

4. Methods and materials

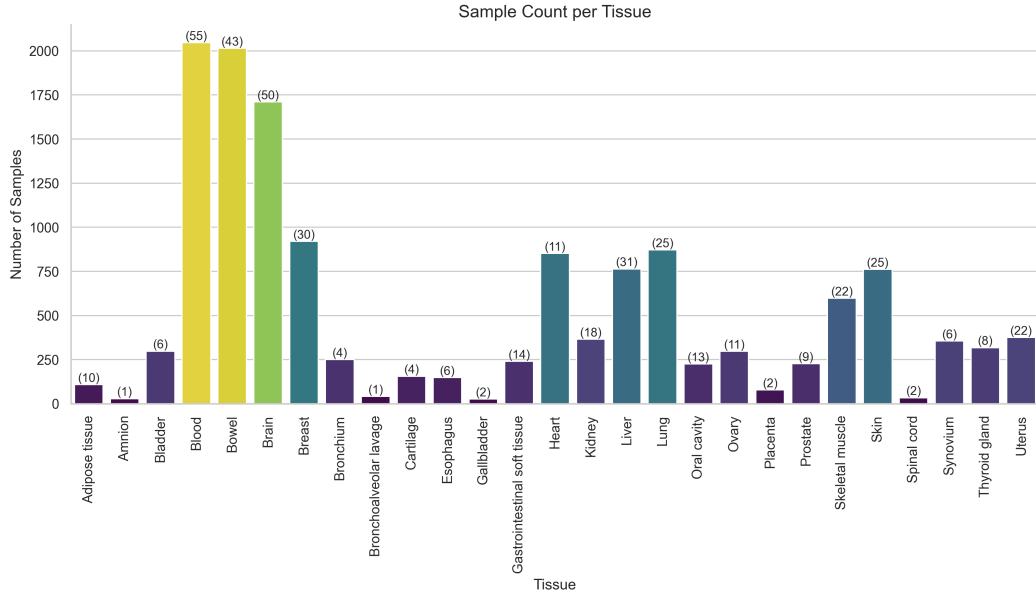


Figure 4.3.: Sample composition of the RNA-seq dataset by tissue. Horizontal bars indicate the total number of samples available for each of the 27 tissues. The annotation at the top of every bar reports the corresponding number of batches.

Before applying batch correction, batches containing only a single sample were excluded. Disease information was used as a biological grouping variable. Following the pipeline for the simulation study, batch correction tools were applied and evaluated across different metrics (Section 4.2.3). To maintain the original dataset structure, corrected values were reintegrated after correction.

5. Results

In this chapter the reComBat-seq implementation described in Chapter 3 is applied to simulated and real data examples to assess its performance. The experiments are carried out following the pipeline outlined in Chapter 4. Additionally, *reComBat-seq* is compared to *ComBat-seq*, *pyComBat-seq* and *reComBat* to show how comparable its performance is to the already established methods and to check whether it can serve as an extension in cases where big datasets are integrated together.

5.1. Simulations

This section covers examples taken from simulation studies, performed with *reComBat-seq*. A hyperparameter study was carried out, comparing the performance of different regularization settings. Afterwards, suitable parameters were chosen and the effect of different setups were explored such as varying gene counts, sample counts and batch counts.

5.1.1. Hyperparameter Study

The aim of this hyperparameter screen is to identify the optimal settings for carrying out the rest of the experiments. This includes the regularization strength λ ($\lambda = 0$ for no regularization up to $\lambda = 1$) and the type of regularization α ($\alpha = 0$ for pure ridge regression and $\alpha = 1$ for LASSO), as well as the levels of batch effect in the mean and the variance. The study focuses on **precision** as the evaluation metric. Figure 5.1 showcases the results of the hyperparameter study across the different settings.

reComBat-seq performed worse when there was little to no dispersion batch effect present, which is consistent with the results observed with *ComBat-seq* [Zhang et al., 2020]. The regularization strength λ had some effect on the performance. However, no consistent trend was observed and further experiments are needed to draw reliable conclusions. A key observation from the hyperparameter study is the sharp degradation in model

5. Results

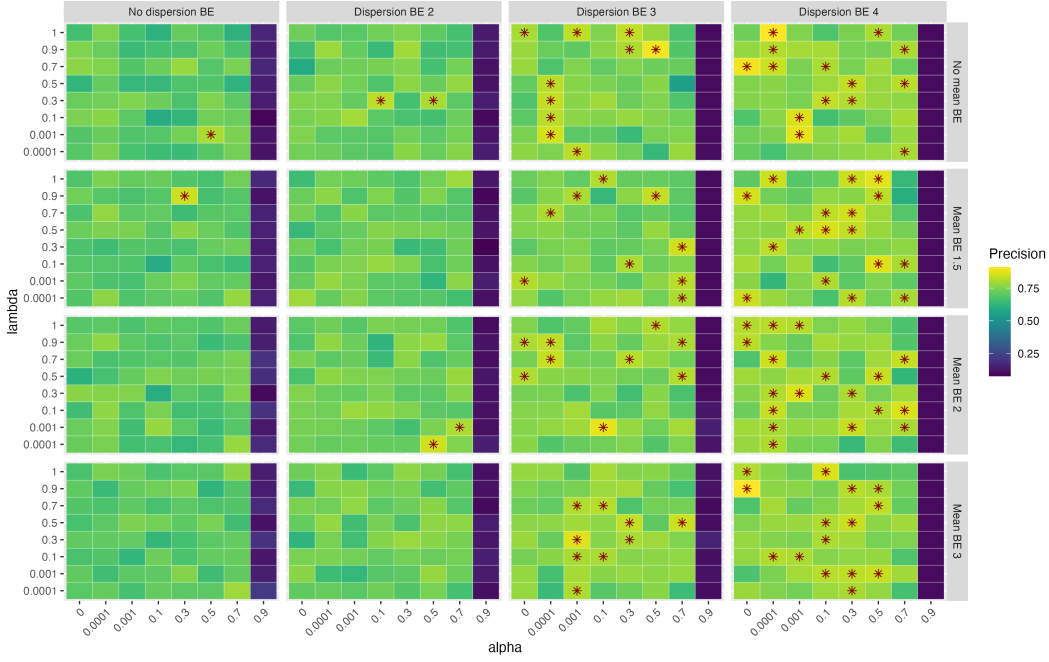


Figure 5.1.: Overview over precision score across different regularization parameters and batch fold effects. The heat-map showcases mean values for the precision over 5 repeated runs evaluated on synthetic data (128 samples, 64 samples/batch, 2 biological groups). Parameter settings that yielded a mean precision score higher than 0.8 were marked with a star.

performance when applying LASSO regression. Values of $\alpha > 0.9$ led to unstable behavior, resulting in extremely low precision scores (as low as 0.1) or, in some cases, infinite loops that required manual interruption of the algorithm.

Given the collinearity introduced by the underdetermined design matrix, the application of LASSO regularization can result in non-convergent behavior and non-unique solutions. This is a consequence of LASSO's feature selection method, which focuses on shrinking some covariate effects to zero. However, in the presence of confounding between batch and group effects this selection may be arbitrary and not necessarily aligned with the most relevant features. In contrast, ridge regression remains stable under such conditions due to its smooth, convex regularization. The clustering of the highest precision scores around intermediate hyperparameter values further supports the use of a more balanced regularization approach.

The rest of the experiments were carried out with the following arbitrary settings: $\alpha = 0.3$ and $\lambda = 0.8$.

5. Results

5.1.2. Gene Count Study

To assess the impact of increasing data dimensionality, each method was tested on synthetic datasets with varying numbers of genes, ranging from 128 to 20,000. The results are summarized in Figure 5.2.

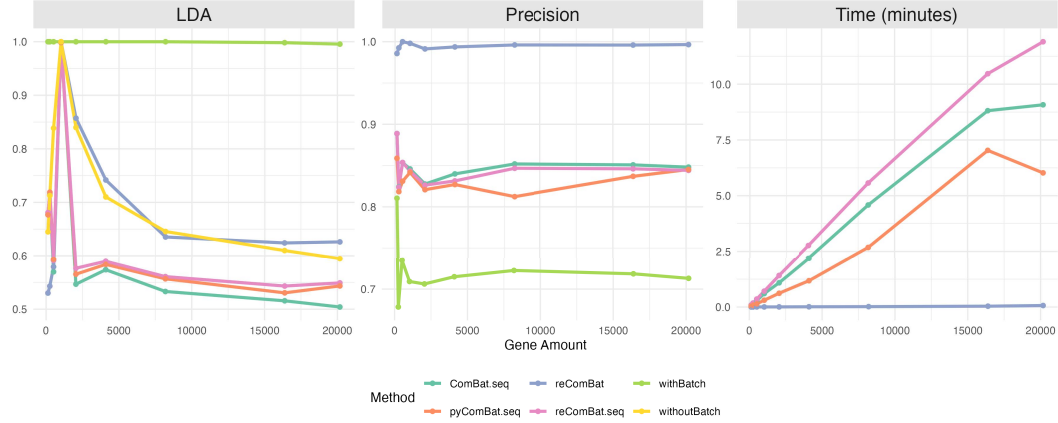


Figure 5.2.: Performance comparison across different number of genes. Scores represent mean values over 5 repeated runs. LDA score (left), Precision (middle), Runtime in minutes (right)

LDA scores show a clear improvement over the uncorrected data containing batch effects. *reComBat* brought the data distribution closer to that of the batch-free reference, while the other versions tended to over-randomize the structure. *reComBat-seq* performed similarly to the other RNAseq methods. Overall, all methods benefited from increasing the number of genes, with performance stabilizing as gene amount grew.

Precision scores remained stable beyond 5 000 genes, with all methods showing improvement over the uncorrected data. *reComBat* consistently achieved the highest precision, while the remaining methods fluctuated between 0.8 and 0.9. True positive rates (TPR) are not shown as they were consistently high across all methods and gene counts, indicating that sensitivity was not affected. The observed variation in precision was driven by differences in false positive rates (FPR).

As mentioned above, the same regularization strategy was applied to the Python variant, *pyComBat-seq*. However, this implementation was not completed due to significantly increased computation times, which exceeded even those of the R-based version. As a result, *reComBat-seq* remains the most computationally expensive method among those shown, followed by *ComBat-seq* and non-regularized *pyComBat-seq*. This is expected, as all three share the same algorithmic structure. In contrast, *reComBat*, developed natively in Python, demonstrated substantially lower runtimes due to its optimized design.

5. Results

Figure 5.3 showcases the performance of the different methods on a dataset with 20171 genes. *reComBat-seq* performed comparably to other methods, achieving both good separation of biological groups and mixing across batches. Notably, *pyComBat-seq* also performed well even with the rank-deficient matrix and without regularization. This is likely due to the aggressive damping used to ensure convergence. While producing good results here, it can be numerically unstable, as demonstrated in the Sample Count Study. A PCA plot from an experiment with only 128 genes is provided in Figure D.1, illustrating how all methods struggle to separate the data by group when fewer genes are available.

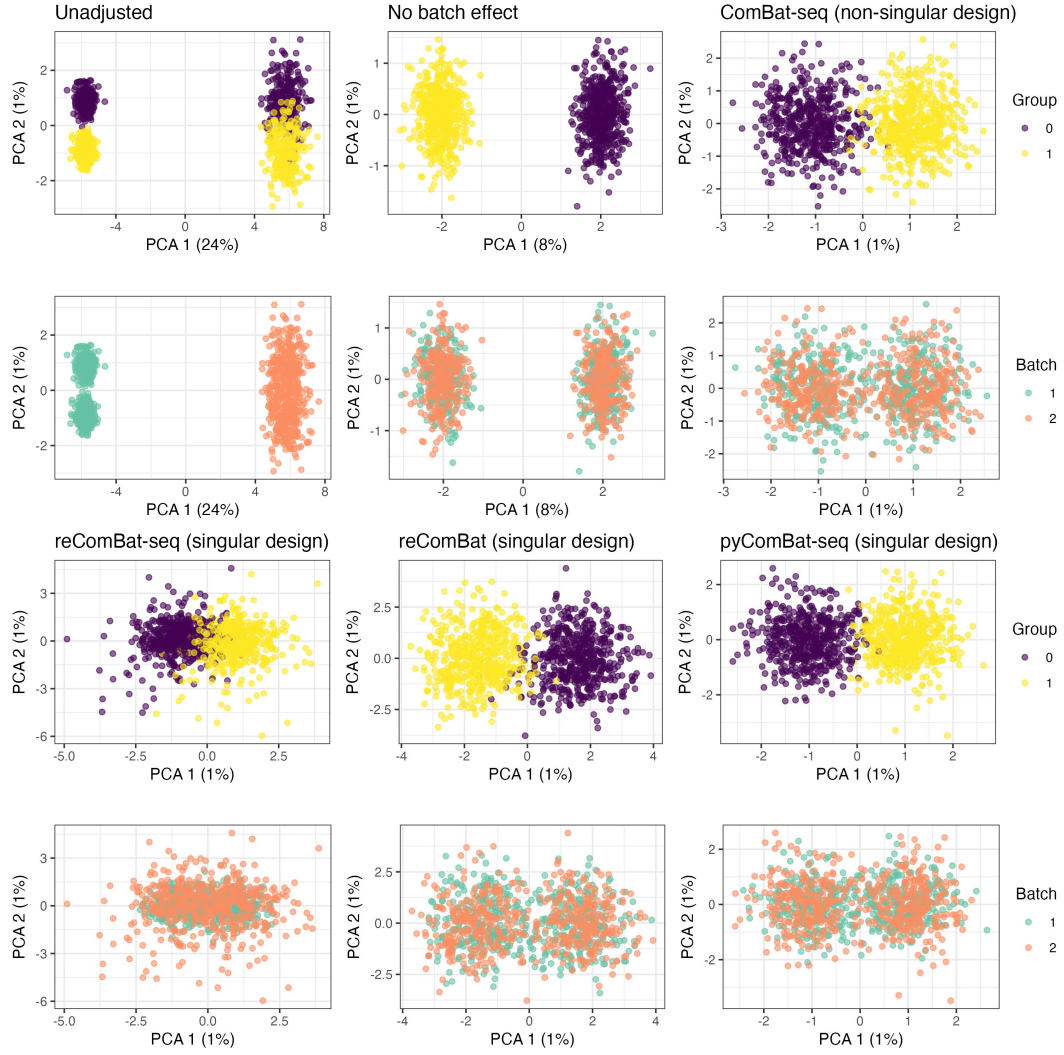
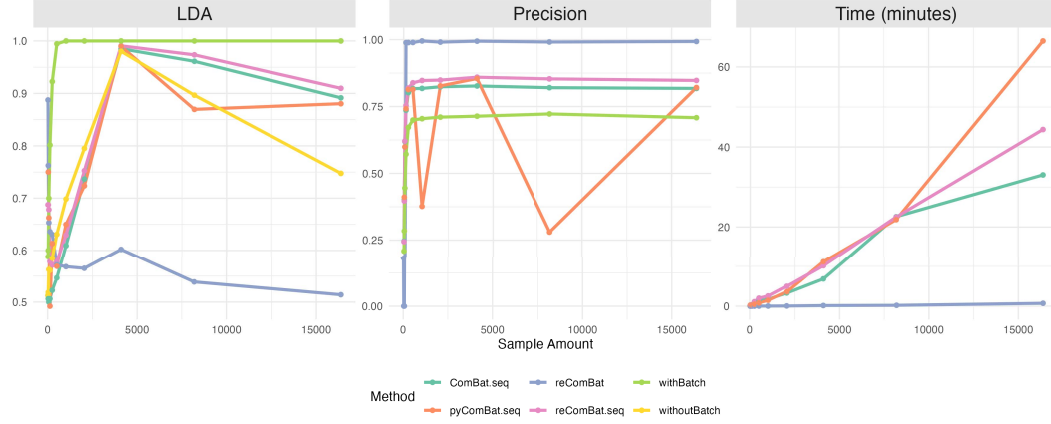


Figure 5.3.: PCA plots of the simulated data with 20171 genes, 1024 samples, 2 batches.

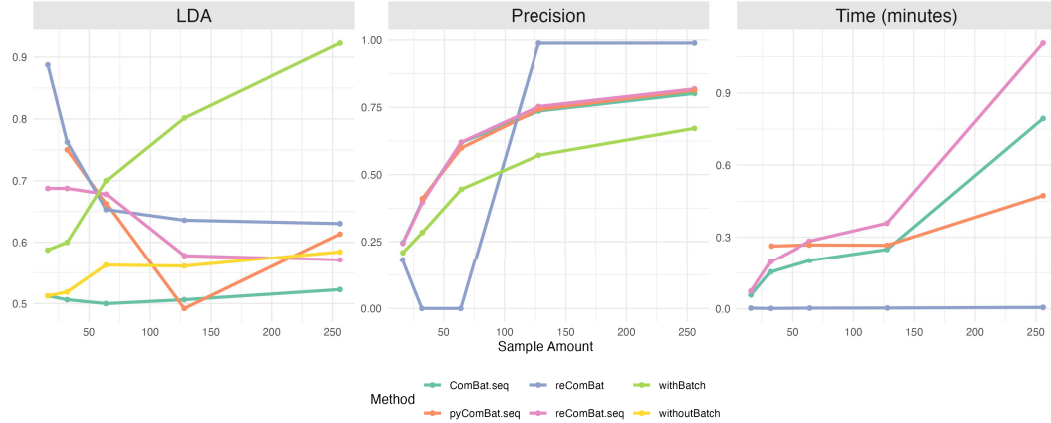
5. Results

5.1.3. Sample Count Study

Continuing with the individual experiments, each method was tested on synthetic datasets with varying number of samples, ranging from 16 to 16384, while keeping the number of genes constant at 5000. The results are summarized in Figure 5.4.



(a) Scores across the complete range (16-16384 samples)



(b) Scores focusing on a subset range (16-250 samples)

Figure 5.4.: Performance comparison across different number of samples. Scores represent mean values over 5 repeated runs. LDA score (left), Precision (middle), Runtime in minutes (right)

The LDA analysis again showed improvement over the uncorrected data. While *reComBat* effectively randomized the batch structure, the RNA-seq methods resembled the batch-free reference but exhibited reduced performance. This trend is further reflected in the PCA plots shown in Figure D.2 for small sample sizes, where none of the methods fully separated the biological signal from the batch effect. As the number of samples increased,

5. Results

precision and overall performance stabilized across all methods, which is expected as larger sample sizes support more robust model estimation.

Precision scores remained stable beyond 100 samples, with all methods outperforming the raw data. *reComBat* achieved the highest precision, while the RNA-seq-based methods hovered around 0.8. As with the gene count experiments, differences in performance were primarily driven by the false positive rate (FPR), which *reComBat* consistently minimized. Notably, the RNA-seq methods demonstrated a steadier improvement at low sample counts, whereas *reComBat* showed a more abrupt performance jump. A sudden precision drop was observed for *pyComBat-seq* due to a spike in false positives, indicating instability in the absence of regularization.

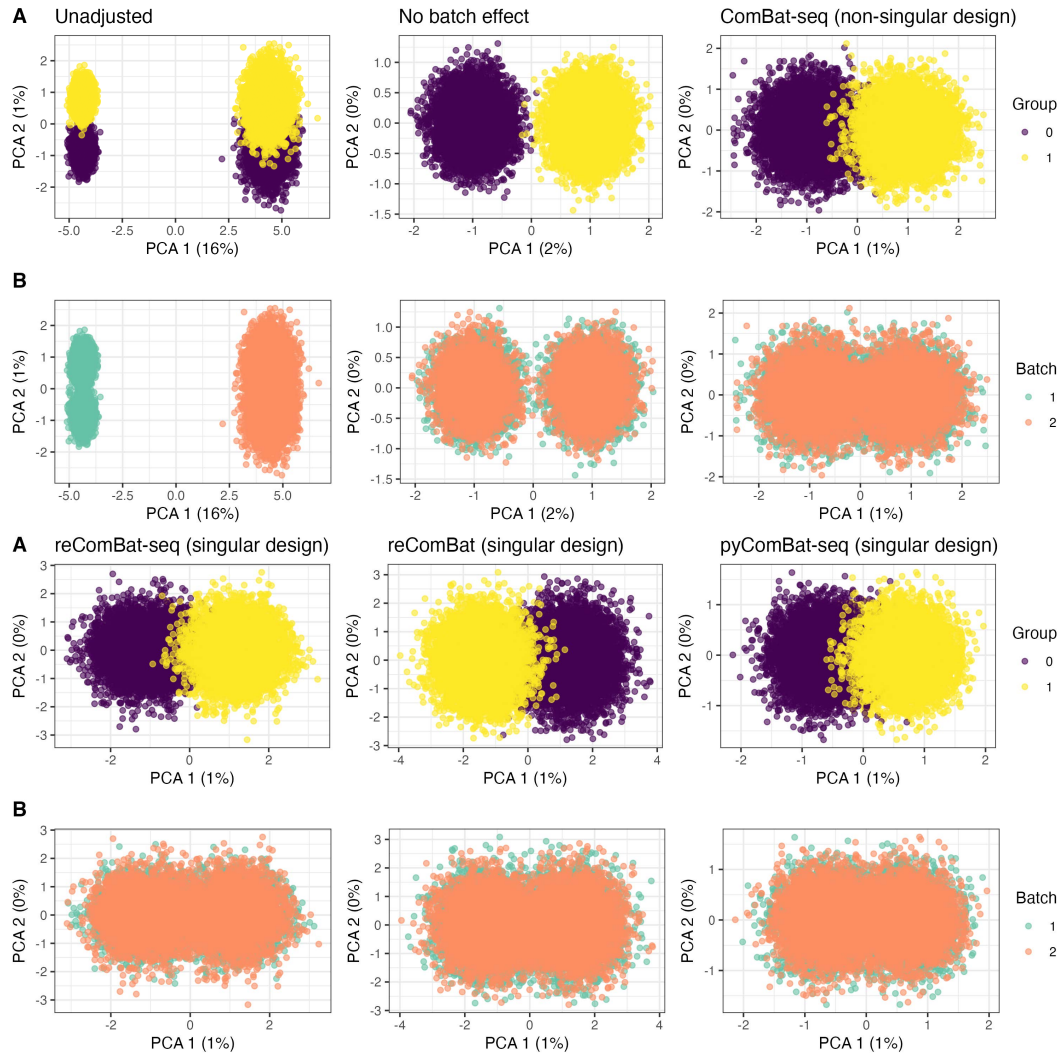


Figure 5.5.: PCA plots of the simulated data with 5000 genes, 16 384 samples, 2 batches

5. Results

The PCA in Figure 5.5 demonstrates improved separation of biological groups and better batch mixing compared to settings with fewer samples as in Figure D.2. *reComBat-seq* performed on par with the other methods, achieving clear group separation and effective batch correction.

5.1.4. Batch Count Study

The last simulation experiment tested performance with varying numbers of batches. Synthetic datasets with 2–40 batches were generated, using 1024 genes. This choice enabled us to test performance under limited gene counts while keeping runtime manageable, as analyses with many batches can take several hours per iteration. Since the implementation is not yet optimized, further validation with larger datasets is needed. *reComBat-seq* was then benchmarked against *reComBat*, while the other two methods failed to complete the analysis due to convergence problems. The results are summarized in Figure 5.6.

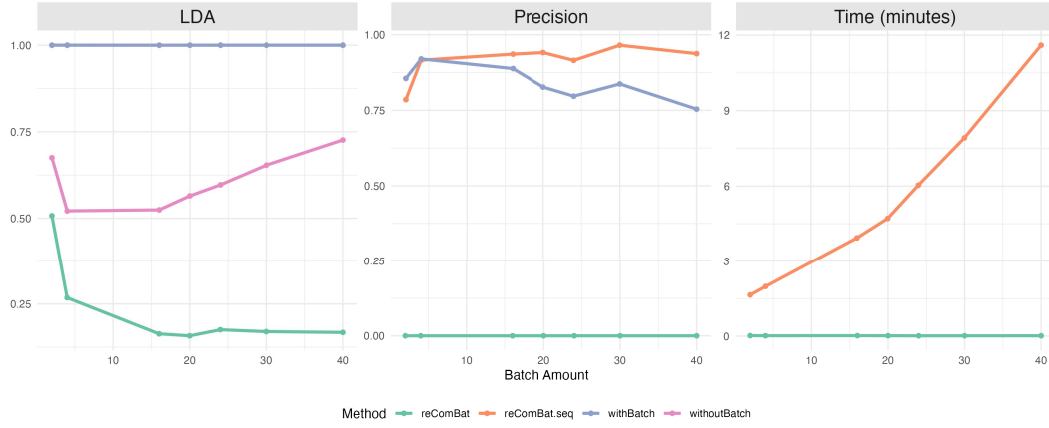


Figure 5.6.: Performance comparison across different number of batches. Scores represent mean values over 5 repeated runs. LDA score (left), Precision (middle), Runtime in minutes (right)

As shown in Figure 5.7, both methods successfully separated the data by biological group and removed batch effects when using 4 batches. With 16 batches however the performance of *reComBat-seq* declined: it brought batches of the same group closer together but failed to fully separate groups or achieve effective batch mixing. A similar pattern was reported in [Tran et al., 2020], where they find that *ComBat*, alongside other methods such as *limma* were able to bring similar cell types across batches close, but with little to no mixing. In contrast, *reComBat* maintained good mixing and successfully grouped samples by biological condition.

5. Results

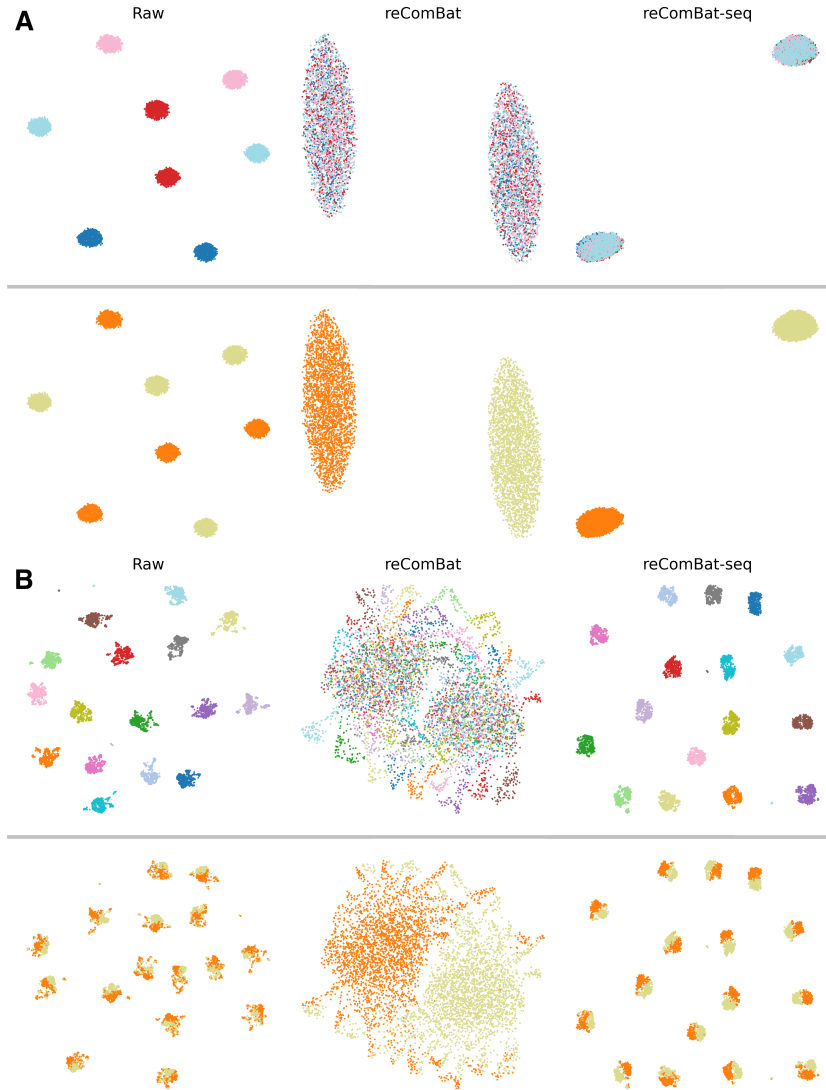


Figure 5.7.: UMAP plots of the simulated data with 1024 genes, 6000 samples. **A** 4 batches **B** 16 batches. The top row is colored by batch, while the bottom row is colored by group.

It is noteworthy however that after carrying out the differential expression analysis, *reComBat* performed poorly, identifying fewer DE genes than *reComBat-seq*, as shown in Figure 5.6. This result was obtained using the same analysis pipeline as in the other simulation experiments, showing the drawbacks of excessive data processing. For *reComBat*, the raw counts had to be standardized and log-transformed before correction and then transformed again for the *limma* DEA workflow, which likely affected the results.

5. Results

5.2. Real data

The *reComBat-seq* approach was applied to a real RNA-seq dataset to evaluate its ability to correct batch effects in a non-controlled environment. The dataset comprises approximately 14 000 samples across 386 batches, as described in the Materials section. To optimize computation time, the data were processed separately for each tissue type.

Out of the 27 analyzed tissues, the results varied in how effectively *reComBat-seq* mitigated batch effects. Based on performance, the tissues could be grouped into three distinct patterns. Three representative cases are presented in the main text, with four additional examples provided in Appendix D. Results from the complete analysis are available at <https://github.com/jas-st/reComBat-seq>.

The *Heart tissue* dataset (Figure 5.8) contains RNA-seq profiles from healthy samples and patients with cardiomyopathy, heart failure, myotonic dystrophy or tetralogy of Fallot. In this case *reComBat-seq* achieves clear clustering by disease while reducing batch effects, whereas *reComBat* yields stronger inter-batch mixing. A similar pattern is observed in the *Esophagus tissue* dataset (Figure D.6). In both examples, the experimental design is relatively simple, with fewer than 15 batches and limited disease diversity conditions.

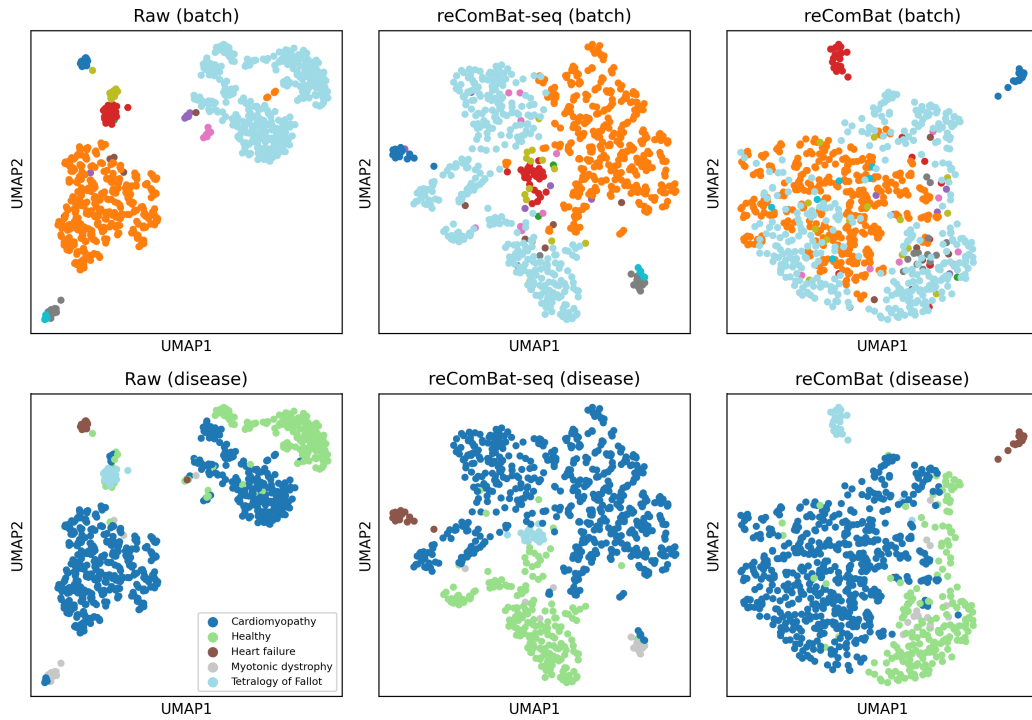


Figure 5.8.: UMAP plots of the Heart tissue data with 852 samples, 11 batches

5. Results

The *Skin tissue* dataset shown in Figure 5.9 is an example of the case described in the Batch count Simulation Study section, where a more complex experimental design leads *reComBat-seq* to bring some clusters closer together, but still not mix them completely, leaving a significant part of the batch effect present. In this case the skin cancer cells are grouped together but full mixing was not achieved. Healthy and psoriasis samples remain intermixed, in contrast to *reComBat*, which separates these three groups more distinctly. A similar outcome is observed in the *Bowel tissue* in Figure D.3, where colorectal cancer samples form a clearer but partially disconnected cluster. Such results are typically seen in datasets with moderately high complexity, involving 20–40 batches and some disease group variation.

A similar pattern was also observed in the *Liver tissue* dataset in Figure D.4, where *reComBat-seq* successfully groups healthy cells together and partially connects the cancer cluster.

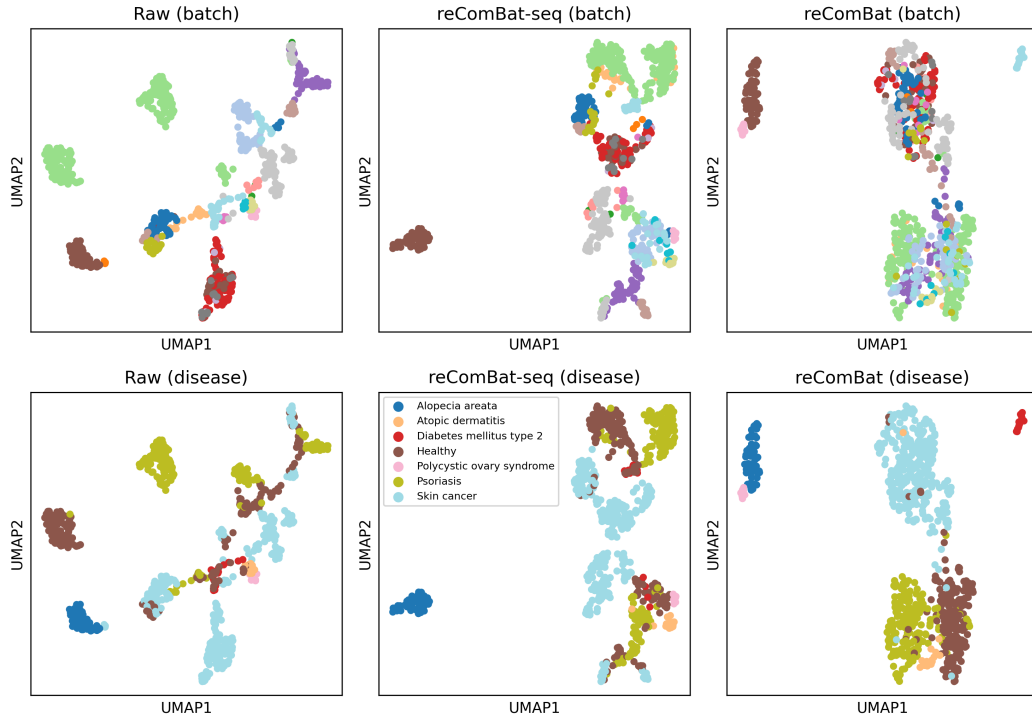


Figure 5.9.: UMAP plots of the Skin tissue data with 761 samples, 25 batches

Finally, the *Brain tissue* dataset (Figure 5.10) represents a case where *reComBat-seq* fails to reliably remove batch effects, leaving the clustering structure largely unchanged compared to the raw data. While some sample groups are brought closer together, the improvement is limited and far less pronounced than with *reComBat*. This pattern is characteristic of datasets with high complexity with more than 40 batches and large

5. Results

diversity in disease types. The Skeletal muscle dataset (Figure D.5) shows the same issue, where healthy cells are brought together, while cancer cells remain split into batch-specific clusters, unlike the stronger integration seen with *reComBat*.

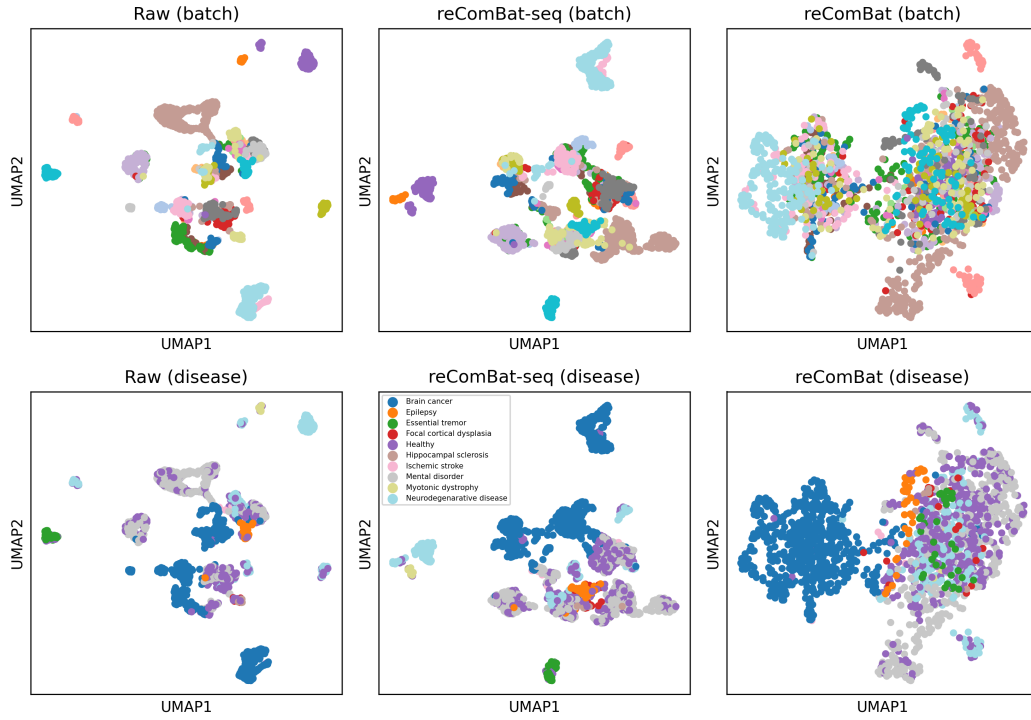


Figure 5.10.: UMAP plots of the Brain tissue data with 1710 samples, 50 batches

6. Discussion

Integrating RNA-seq data from multiple sources offers several advantages, including increased sample sizes, improved statistical power for model fitting and the ability to detect more subtle patterns of biological variation. By pooling datasets across studies, researchers can broaden the diversity of samples and enhance the generalizability of their findings. However, combining data from independent experiments also introduces technical variation, commonly referred to as batch effects. They stem from variations in experimental procedures between studies, including sample collection, library preparation, sequencing platforms, and laboratory-specific practices. These unwanted sources of variation can obscure underlying biological signals and significantly impact downstream analyses, such as differential expression. While several tools exist to address batch effects, many such as *ComBat-seq* do not support the analysis of highly correlated design features or are not adapted for RNA-seq data (*reComBat*).

This thesis explored the regularization approach used in *reComBat* [Adamer et al., 2022] and adapted it for *ComBat-seq* to get a tool that both preserves the integer nature of the RNA-seq data and relaxes the strict requirements on collinearity in the design matrix, offering greater flexibility in experimental design. The mathematical foundations were formulated to enable testing of alternative implementations in the future. An R implementation of the proposed method was further developed as a proof of concept and benchmarked against the existing tools. While *reComBat* consistently outperformed the baseline *ComBat-seq* tools, the proposed *reComBat-seq* method demonstrated comparable performance in many scenarios, highlighting its potential for handling underdetermined RNA-seq count data.

The hyperparameter screening indicated that the regularized methods should be applied primarily in scenarios where there is clear evidence of batch effects and where such effects negatively influence downstream analysis. In the absence of significant batch-related variation, the added complexity and runtime may not justify their use. It is worth noting that tools such as *edgeR* and *DESeq2* provide the flexibility to include confounding variables directly in their statistical models, in order to account for unwanted intergroup variability in downstream analyses. In some cases, carefully modeling known confounders may serve as an effective alternative or complement to dedicated batch-correction approaches.

6. Discussion

Experiments varying the number of samples and genes further demonstrated that batch effects can be corrected even when using underdetermined design matrices. While *reComBat-seq* did not outperform other RNA-seq-specific methods in these settings, it achieved comparable performance despite the introduced collinearity. However, these simulations were conducted under highly controlled conditions. Broader testing across diverse settings and more complex batch structures is needed to fully evaluate robustness and generalizability.

One key limitation emerged when increasing the number of batches. While *reComBat* remained effective even with 16 and more batches, *reComBat-seq* struggled to remove the batch effect and retain the biological separation. Notably, the original *ComBat-seq* method failed entirely in these cases due to collinearity in the design matrix, highlighting where *reComBat-seq* provides a viable alternative. Surprisingly, differential expression analysis showed reduced performance after *reComBat* correction, while the RNA-seq based methods remained stable. This reiterates the importance of using a method that preserves the integer structure of RNA-seq data to maintain compatibility with downstream analysis pipelines.

When tested on real datasets with more complex biological structures, *reComBat-seq* tended to under-correct, often leaving residual batch signal. This effect was particularly noticeable when batch effects were stronger or when the data included multiple groups and covariates. One likely reason is that *reComBat-seq* fits the model separately for each group, which effectively reduces the number of samples used in each fit and therefore the statistical power to accurately estimate and remove batch effects. This issue appears analogous to the trend observed in our simulation studies: as the number of batches increases, the performance of the method decreases, reflecting a similar limitation in handling many sources of variation simultaneously. It is also observed when handling a small number of samples. While these patterns suggest a connection between experimental complexity and performance, further investigation is needed to fully define the limitations under different study designs.

The most significant limitation of *reComBat-seq* remains its runtime. The current implementation is not optimized for large datasets and may require several days to complete analyses that *reComBat* can finish in hours. However, the algorithm is well-suited for parallelization: it independently fits models for each gene and condition, offering opportunities for GPU acceleration or CPU-based parallel computation via gene-wise batching. At the time of development, *edgeR* 3.39 was used, but since then, version 4.0 has been released. The updated version includes improvements to the quasi-likelihood fitting pipeline and introduces C-level optimizations that reduce runtime and memory usage for large-scale datasets, potentially benefiting future implementations of *reComBat-seq*.

Overall, the regularization-based approach implemented in *reComBat-seq* shows promise

6. Discussion

and the underlying mathematical framework provides a solid foundation for further extensions and methodological improvements. While performance remains behind *reComBat* in certain metrics, it is comparable to its RNA-seq counterparts, offering a possible alternative when count-preserving correction is required and traditional methods are inapplicable due to matrix singularity or collinearity. It is worth noting, however, that although *reComBat* outperforms all other methods on several correction benchmarks, its strong assumptions about the structure of batch effects appear to make it less reliable for downstream analyses, where overcorrection can obscure true biological signal. By contrast, *reComBat-seq* is less restrictive in its modeling and thus avoids some of the degradation in biological signal, offering a more balanced trade-off between effective batch correction and preservation of biological variation.

Bibliography

- [Abramowitz and Stegun, 1972] Abramowitz, M. and Stegun, I. A., editors (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications.
- [Adamer et al., 2022] Adamer, M. F., Brüningk, S. C., Tejada-Arranz, A., Estermann, F., Basler, M., and Borgwardt, K. (2022). recombat: batch-effect removal in large-scale multi-source gene-expression data integration. *Bioinformatics Advances*, 2(1):vbac071.
- [Alberts et al., 2008] Alberts, B., Wilson, J., and Hunt, T. (2008). *Molecular Biology of the Cell*. Garland Science.
- [Anders and Huber, 2010] Anders, S. and Huber, W. (2010). Anders s, huber w.. differential expression analysis for sequence count data. *genome biol* 11: R106. *Genome biology*, 11:R106.
- [Barry, 2020] Barry, T. (2020). Tim barry: Gamma, poisson, and negative binomial distributions.
- [Behdenna et al., 2023] Behdenna, A., Colange, M., Haziza, J., Bergel, M., Maman, S., Richard, H., Leclercq, S., Vallot, C., and Zerbib, J.-C. (2023). pyComBat, a Python tool for batch effects correction in high-throughput molecular data using empirical Bayes methods. *BMC Bioinformatics*, 24(1):459.
- [Bolstad et al., 2003] Bolstad, B., Irizarry, R., Åstrand, M., and Speed, T. (2003). A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193.
- [Brown and Clancy, 2008] Brown, W. A. C. and Clancy, S. (2008). Translation: Dna to mrna to protein. In *Nature Education*.
- [Buitinck et al., 2013] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine

Bibliography

- learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.
- [Bumgarner, 2013] Bumgarner, R. (2013). Overview of dna microarrays: Types, applications, and their future. *Current Protocols in Molecular Biology*, 101(1):22.1.1–22.1.11.
- [Cameron and Trivedi, 1998] Cameron, A. C. and Trivedi, P. K. (1998). *Regression Analysis of Count Data*, volume 30 of *Econometric Society Monographs*. Cambridge University Press, Cambridge, UK.
- [Carlin and Louis, 2000] Carlin, B. P. and Louis, T. A. (2000). *Bayes and Empirical Bayes Methods for Data Analysis*. Chapman & Hall/CRC, 2nd edition.
- [Chen et al., 2014] Chen, Y., Lun, A. T. L., and Smyth, G. K. (2014). *Differential Expression Analysis of Complex RNA-seq Experiments Using edgeR*, pages 51–74. Springer International Publishing, Cham.
- [Crick, 1958] Crick, F. H. (1958). On protein synthesis. In *Symp Soc Exp Biol*, volume 12, page 8.
- [Davison, 2003] Davison, A. C. (2003). *Linear Regression Models*, page 353–416. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.
- [Deshpande et al., 2023] Deshpande, D., Chhugani, K., Chang, Y., Karlsberg, A., Loeffler, C., Zhang, J., Muszyńska, A., Munteanu, V., Yang, H., Rotman, J., Tao, L., Balliu, B., Tseng, E., Eskin, E., Zhao, F., Mohammadi, P., P. Łabaj, P., and Mangul, S. (2023). Rna-seq data science: From raw data to effective interpretation. *Frontiers in Genetics*, Volume 14 - 2023.
- [Emmert-Streib and Dehmer, 2019] Emmert-Streib, F. and Dehmer, M. (2019). High-dimensional lasso-based computational regression models: Regularization, shrinkage, and selection. *Machine Learning and Knowledge Extraction*, 1:359–385.
- [Frazee et al., 2015] Frazee, A. C., Jaffe, A. E., Langmead, B., and Leek, J. T. (2015). Polyester: simulating rna-seq datasets with differential transcript expression. *Bioinformatics*, 31(17):2778–2784.
- [Hardin and Hilbe, 2012] Hardin, J. W. and Hilbe, J. M. (2012). *Generalized Estimating Equations*. Chapman and Hall/CRC, 2 edition.
- [Hilbe, 2011] Hilbe, J. M. (2011). *Negative Binomial Regression*. Cambridge University

Bibliography

Press, New York, 2 edition.

- [Himes et al., 2014] Himes, B. E., Jiang, X., Wagner, P., Hu, R., Wang, Q., Klanderman, B., Whitaker, R. M., Duan, Q., Lasky-Su, J., Nikolos, C., Jester, W., Johnson, M., Panettieri, R. A., Tantisira, K. G., Weiss, S. T., and Lu, Q. (2014). Rna-seq transcriptome profiling identifies crispld2 as a glucocorticoid responsive gene that modulates cytokine function in airway smooth muscle cells. *PLoS ONE*, 9(6):e99625.
- [Huch and Nissan, 2014] Huch, S. and Nissan, T. (2014). Interrelations between translation and general mrna degradation in yeast. *Wiley Interdisciplinary Reviews: RNA*, 5(6):747–763.
- [Johnson et al., 2006] Johnson, W. E., Li, C., and Rabinovic, A. (2006). Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127.
- [Karakach et al., 2010] Karakach, T. K., Flight, R. M., Douglas, S. E., and Wentzell, P. D. (2010). An introduction to dna microarrays for gene expression analysis. *Chemometrics and Intelligent Laboratory Systems*, 104(1):28–52. OMICS.
- [Katznelson and Katznelson, 2008] Katznelson, Y. and Katznelson, Y. R. (2008). *A (Terse) Introduction to Linear Algebra*. American Mathematical Society.
- [Kellis et al., 2015] Kellis, M. et al. (2015). Computational biology: Genomes, networks, evolution. mit course 6.047/6.878. License: Creative Commons BY-NC-SA.
- [Liesen and Mehrmann, 2015] Liesen, J. and Mehrmann, V. (2015). *Linear Algebra*. Springer International Publishing, Switzerland.
- [Love et al., 2014] Love, M. I., Huber, W., and Anders, S. (2014). Moderated estimation of fold change and dispersion for rna-seq data with deseq2. *Genome Biology*, 15(12):550.
- [Lund et al., 2012] Lund, S. P., Nettleton, D., McCarthy, D. J., and Smyth, G. K. (2012). Detecting differential expression in rna-sequence data using quasi-likelihood with shrunk dispersion estimates. *Statistical Applications in Genetics and Molecular Biology*, 11(5):Article 18.
- [McCarthy et al., 2012] McCarthy, D. J., Chen, Y., and Smyth, G. K. (2012). Differential expression analysis of multifactor rna-seq experiments with respect to biological variation. *Nucleic Acids Research*, 40(10):4288–4297.

Bibliography

- [Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- [Ritchie et al., 2015] Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., and Smyth, G. K. (2015). limma powers differential expression analyses for rna-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47–e47.
- [Robinson et al., 2010] Robinson, M. D., McCarthy, D. J., and Smyth, G. K. (2010). edgeR: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140.
- [Robinson and Oshlack, 2010] Robinson, M. D. and Oshlack, A. (2010). A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology*, 11(3):R25.
- [Robinson and Smyth, 2008] Robinson, M. D. and Smyth, G. K. (2008). Small-sample estimation of negative binomial dispersion, with applications to sage data. *Biostatistics*, 9(2):321–332.
- [Rosati et al., 2024] Rosati, D., Palmieri, M., Brunelli, G., Morrione, A., Iannelli, F., Frullanti, E., and Giordano, A. (2024). Differential gene expression analysis pipelines and bioinformatic tools for the identification of specific biomarkers: A review. *Comput. Struct. Biotechnol. J.*, 23:1154–1168.
- [Schönhage et al., 1994] Schönhage, A., Grotefeld, A. F. W., and Vetter, E. (1994). *Fast Algorithms—A Multitape Turing Machine Implementation*. BI Wissenschafts-Verlag.
- [Smyth, 2004] Smyth, G. K. (2004). Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Stat. Appl. Genet. Mol. Biol.*, 3(1):Article3.
- [Termanini, 2023] Termanini, R. (2023). Chapter 12 - storage and execution of dna contaminated code. In Termanini, R., editor, *Biomedical Defense Principles to Counter DNA Deep Hacking*, pages 239–263. Academic Press.
- [Torrenté et al., 2019] Torrenté, L. d., Zimmerman, S., Suzuki, M., Christopeit, M., Greal, J. M., and Mar, J. C. (2019). The shape of gene expression distributions matter: how incorporating distribution shape improves the interpretation of cancer transcriptomic data. *bioRxiv*.

Bibliography

- [Tran et al., 2020] Tran, H. T. N., Ang, K. S., Chevrier, M., Zhang, X., Lee, N. G. I., Chen, J., Poidinger, M., Chen, Y., Loh, A. P., Koeffler, H. P., Teh, B. T., Chan, J. K. Y., Yoshida, T., Koh, C. X., Tong, J. C., Loh, M., Su, I. M., Zeng, Z., Chng, W. J., Robson, P., Yang, L., Ng, H. H., and Chan, J. Y. K. (2020). A benchmark of batch-effect correction methods for single-cell rna sequencing data. *Genome Biology*, 21(1):12.
- [Urry et al., 2021] Urry, L., Cain, M., Wasserman, S., Minorsky, P., and Orr, R. (2021). *Campbell Biology*. Pearson.
- [Zappia et al., 2017] Zappia, L., Phipson, B., and Oshlack, A. (2017). Splatter: simulation of single-cell rna sequencing data. *Genome Biology*, 18(1):174.
- [Zhang and Jia, 2022] Zhang, H. and Jia, J. (2022). Elastic-net regularized high-dimensional negative binomial regression: Consistency and weak signal detection. *Statistica Sinica*.
- [Zhang et al., 2020] Zhang, Y., Parmigiani, G., and Johnson, W. E. (2020). Combat-seq: batch effect adjustment for rna-seq count data. *NAR Genomics and Bioinformatics*, 2(3):lqaa078.
- [Zhou et al., 2011] Zhou, Y. H., Xia, K., and Wright, F. A. (2011). A powerful and flexible approach to the analysis of RNA sequence count data. *Bioinformatics*, 27(19):2672–2678.
- [Zou and Hastie, 2005] Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.

A. Appendix - Linear and Generalized Linear Models

This section will cover the mathematical background, the estimation methods and algorithms for ordinary linear regression models as well as generalized linear models (GLM).

A.1. Linear regression models

Linear regression models are used to describe the relationship between a dependent/response variable and its corresponding explanatory variables. They assume that the response variables are consistent with constant variance and the mean of the responses is equal to the linear predictor.

A.1.1. Model assumptions

Let $x = (x_1, \dots, x_n)$ define the explanatory variables and $y = (y_1, \dots, y_n)$ the corresponding response. Suppose the response variable y is a linear function of the features:

$$y_i = \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \epsilon_i = x_i \beta + \epsilon_i$$

with $x_i = (x_{i1}, \dots, x_{ip})$, $\beta = (\beta_1, \dots, \beta_p)$. This can be summarized in the following matrix notation, using the $n \times p$ design matrix $X = (x_{ij})$ and $\epsilon = (\epsilon_1, \dots, \epsilon_n)$:

$$y = X\beta + \epsilon$$

The goal is to find such β coefficients that best approximate the model.

A.1.2. Coefficient estimation methods

A.1.2.1. Ordinary least squares (OLS)

The following method holds under the assumption that the errors follow a normal distribution $\epsilon \sim N(0, \sigma^2)$ and have constant variance (homoscedasticity). The ordinary least squares method estimates the coefficients β by minimizing the sum of squares [Davison, 2003]:

$$\hat{\beta} = \arg \min_{\beta} SS(\beta) = \arg \min_{\beta} \|y - X\beta\|_2^2 = \arg \min_{\beta} (y - X\beta)^T (y - X\beta) \quad (\text{A.1})$$

This can be solved by finding the derivative of the sum of squares and setting it to zero, equivalent to solving the normal equations [Davison, 2003]:

$$X^T(y - X\beta) = 0 \Rightarrow (X^T X)\beta = X^T y$$

This equation has a unique solution if the matrix $(X^T X)$ is of full rank and therefore invertible [Davison, 2003]:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

The variance of $\hat{\beta}$ can then be calculated as:

$$\text{Var}(\hat{\beta}|X) = \sigma^2 (X^T X)^{-1}$$

A.1.2.2. Weighted least squares (WLS)

Suppose the errors have unequal variances, governed by a known weight matrix $W = \text{diag}\{w_1, \dots, w_n\}$. They are given by $\text{var}(\epsilon_j) = \sigma^2/w_j$.

The homoscedasticity can be eliminated by multiplying each observation by these known weights. The equation then becomes of the same form as A.1, the estimators can therefore be calculated as:

$$\hat{\beta} = (X^T W X)^{-1} X^T W y \quad (\text{A.2})$$

A.2. Generalized linear models (GLM)

Generalized linear model (GLM) is a generalization of linear regression, relaxing the assumptions by allowing non-linear relationship between the response mean and the linear

A. Appendix - Linear and Generalized Linear Models

predictor, as well as non-normal density of the response. The relationship is defined using a link function $g(\mu)$, assumed to be the canonical function in linear regression: $\eta = X\beta = \mu = g(\mu)$.

A.2.1. Model assumptions

Using the notations from Section A.1.1 the following model is defined:

$$\eta = X\beta \quad \mu = g^{-1}(\eta),$$

with η being the linear predictor function and $g^{-1}(\eta)$ the link function. It is further assumed that the response comes from a distribution from the exponential family [Davison, 2003].

The expected value and variance are defined as:

$$\begin{aligned}\mathbb{E}(y|x) &= \mu = g^{-1}(X\beta) \\ \text{Var}(y|x) &= V(\mu),\end{aligned}$$

where $V(\mu)$ is the variance function that characterizes the mean–variance relationship.

For the Poisson and negative binomial distributions, a log-link function is commonly used:

$$X\beta = g(\mu) = \log \mu \Rightarrow \mu = e^{X\beta}$$

The variance function for the negative binomial is often expressed as:

$$V(\mu) = \mu + \phi\mu^2,$$

where ϕ is a dispersion parameter controlling the degree of overdispersion. The Poisson distribution is obtained as a limiting case of the negative binomial when $\phi = 0$.

A.2.2. Maximum likelihood estimation methods

The parameters $\beta = (\beta_1, \dots, \beta_p)$ are typically estimated using maximum likelihood methods (MLE). Given observed discrete values y with a probability function $f(y; \theta)$ with θ as the mean/ location parameter, the plausibility of different θ values can be assessed with the help of the likelihood function [Davison, 2003]:

$$L(\theta; y) = f(y; \theta) = \prod_{i=1}^n f(y_i; \theta)$$

A. Appendix - Linear and Generalized Linear Models

By evaluating the product of the probabilities of all observations under the model specified by θ , one can determine how likely a particular parameter setting is, given the data. In practice, it is often more convenient to calculate the log-likelihood, as it transforms the product into a sum, which is numerically more stable [Schönhage et al., 1994].

$$\ell(\theta; y) = \log L(\theta; y) = \log \prod_{i=1}^n f(y_i; \theta) = \sum_{i=1}^n \ell_i(\theta; y),$$

with $\ell_i(\theta)$ equal to $\log f(y_i; \theta)$.

Maximum likelihood estimation is based on finding the estimator of the parameter that maximizes the log-likelihood or equivalently that minimizes the negative log-likelihood cost function (The likelihood functions for the negative binomial distribution are defined in detail in Appendix C). For methods like IRLS, the negative log-likelihood is preferred because these algorithms are naturally set up to minimize a cost function instead of maximizing the likelihood directly.

$$\hat{\theta} = \arg \min_{\theta} -\ell(\theta; y) = \arg \min_{\theta} NLL(\theta; y)$$

A.2.2.1. Newton-Rhapson

The Newton-Rhapson method of estimation is a root-finding algorithm, operating using iterative steps to find an approximation for the root $f(r) = 0$ of a function f . Using the first order Taylor approximation of the function, the linear approximation around a starting guess x_0 for the root r can be obtained:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0).$$

Setting $f(x_1) = 0$ then yields a new approximation for the root. This creates the iterative process of approximating the real root of the function by using the roots of the tangent line approximations [Hilbe, 2011].

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

In practice, this method is frequently employed as an optimization technique, such as in the estimation of a negative binomial regression model. To identify the maximum of the log-likelihood function (or equivalently the minimum of the negative log-likelihood) the method is applied to the gradient of the objective function to iteratively locate the extremum [Hilbe, 2011]:

$$\beta^{n+1} = \beta^n - H^{-1} \left(\frac{\partial \ell(\beta^n; y)}{\partial \beta} \right),$$

A. Appendix - Linear and Generalized Linear Models

where $\frac{\partial \ell}{\partial \beta} = \left(\frac{\partial \ell}{\partial \beta_1} \dots \frac{\partial \ell}{\partial \beta_p} \right)$ is the gradient and H is the Hessian matrix of the cost function with:

$$H = \frac{\partial^2 \ell}{\partial \beta^2} = \begin{pmatrix} \frac{\partial^2 \ell}{\partial \beta_1^2} & \frac{\partial^2 \ell}{\partial \beta_1 \partial \beta_2} & \cdots & \frac{\partial^2 \ell}{\partial \beta_1 \partial \beta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \ell}{\partial \beta_n \partial \beta_1} & \frac{\partial^2 \ell}{\partial \beta_n \partial \beta_2} & \cdots & \frac{\partial^2 \ell}{\partial \beta_n^2} \end{pmatrix}.$$

The negative Hessian matrix is also known as the *observed information* or *observed Fisher information* in the model [Davison, 2003].

A.2.2.2. Fisher scoring

In large-scale experiments, computing the Hessian matrix can become computationally expensive, especially when evaluating it at every iteration of the Newton-Raphson method. The Fisher scoring algorithm addresses this issue by replacing the observed Hessian with the *expected Fisher information* matrix, given by [Davison, 2003]:

$$\mathcal{I}(\beta) = \mathbb{E}[-H] = \mathbb{E} \left[-\frac{\partial^2 \ell(\beta; y)}{\partial \beta^2} \right] = \mathbb{E} \left[\frac{\partial^2 NLL(\beta; y)}{\partial \beta^2} \right].$$

The form of the Newton-Raphson iteration steps that is relevant for this thesis is therefore the following, using $NLL(\beta) = -\ell(\beta)$:

$$\beta^{n+1} = \beta^n - (\mathcal{I}(\beta))^{-1} \left(\frac{\partial NLL(\beta^n; y)}{\partial \beta} \right) = \beta^n + (\mathcal{I}(\beta))^{-1} \left(\frac{\partial \ell(\beta^n; y)}{\partial \beta} \right).$$

A.2.2.3. Iteratively Reweighted Least Squares (IRLS)

In the early 1970s, Nelder and Wedderburn adapted the existing method of weighted linear regression to implement the Fisher Scoring algorithm for generalized linear models. By reformulating maximum likelihood estimation as a weighted least squares problem, they were able to reuse standard linear model fitting procedures. This approach provided a computationally efficient solution for estimating a broad class of models from the exponential family, without the need for developing new optimization routines for each case [Hardin and Hilbe, 2012].

The *IRLS* fitting algorithm is an iterative fitting procedure, that solves the following weighted linear least squares problem in each iteration to find the next β^{n+1} :

$$\beta^{n+1} = (X^T W^n X)^{-1} X^T W^n y.$$

A. Appendix - Linear and Generalized Linear Models

The procedure follows largely the linear regression methods described in Section A.1.

To illustrate how the *Fisher scoring* iterative form can be transformed into a form suitable for weighted linear regression, the negative binomial distribution will be used. This transformation can be generalized to all members of the exponential family of distributions, as outlined in [Hilbe, 2011][Methods of estimation].

Given the negative binomial log-likelihood function as derived in Appendix C and using the link function $\mu_j = e^{x_j\beta}$, assuming that the dispersion parameters α are estimated and fixed, this section will cover the derivation of the corresponding derivatives. This is done under the assumption that Y is a random variable with $Y \sim NB(\mu, \alpha)$ and $y_j = (y_{j1}, \dots, y_{jn})$ is the j -th row of the count matrix corresponding to the counts for one gene across different samples.

$$\begin{aligned} \ell(\beta; y) = & \sum_{j=1}^n \log \Gamma\left(y_j + \frac{1}{\alpha}\right) - \log \Gamma(y_j + 1) - \log \Gamma\left(\frac{1}{\alpha}\right) + y_j \log \alpha + y_j \log \mu_j - \\ & \left(y_j + \frac{1}{\alpha}\right) \log(1 + \alpha\mu_j) \end{aligned}$$

The sample specific row of the design matrix is denoted by $x_j = (x_{j1}, \dots, x_{jn})$. The gradient of $\mu_j = e^{x_j\beta}$ is given by:

$$\begin{aligned} \frac{\partial \mu_j}{\partial \beta} &= \left(\frac{\partial \mu_j}{\partial \beta_1}, \dots, \frac{\partial \mu_j}{\partial \beta_p} \right) \\ \frac{\partial \mu_j}{\partial \beta_k} &= \frac{\partial e^{x_j\beta}}{\partial \beta_k} = x_{jk} e^{x_j\beta} = x_{jk} \mu_j \end{aligned}$$

The first derivative with respect to β_k is derived accordingly as:

$$\begin{aligned} \frac{\partial \ell(\beta; y)}{\partial \beta_k} &= \frac{\partial}{\partial \beta_k} \sum_{j=1}^n y_j \log \mu_j - \left(y_j + \frac{1}{\alpha}\right) \log(1 + \alpha\mu_j) \\ &= \sum_{j=1}^n y_j x_{jk} \mu_j \frac{1}{\mu_j} - \left(y_j + \frac{1}{\alpha}\right) \frac{1}{1 + \alpha\mu_j} (\alpha x_{jk} \mu_j) \\ &= \sum_{j=1}^n \frac{y_j x_{jk} (1 + \alpha\mu_j) - (\alpha y_j + 1) x_{jk} \mu_j}{1 + \alpha\mu_j} \\ &= \sum_{j=1}^n \frac{y_j x_{jk} - x_{jk} \mu_j}{1 + \alpha\mu_j} = \sum_{j=1}^n \frac{x_{jk} (y_j - \mu_j)}{1 + \alpha\mu_j} \end{aligned}$$

The second derivative with respect to β_l , serving also as an element of the Hessian matrix

A. Appendix - Linear and Generalized Linear Models

can be calculated as follows:

$$\begin{aligned} H_{lk} &= \frac{\partial^2 \ell}{\partial \beta_l \partial \beta_k} = \frac{\partial \ell}{\partial \beta_l} \sum_{j=1}^n \frac{x_{jk}(y_j - \mu_j)}{1 + \alpha \mu_j} \\ &= \sum_{j=1}^n x_{jk} \frac{-x_{jl} \mu_j (1 + \alpha \mu_j) - (y_j - \mu_j) \alpha x_{jl} \mu_j}{(1 + \alpha \mu_j)^2} \end{aligned}$$

Since *IRLS* is built upon the *Fisher scoring algorithm*, the expected value of the negative Hessian matrix is to be used. The calculation can therefore be simplified accordingly by using the following fact: $\mathbb{E}[y_j] = \mu_j \Rightarrow \mathbb{E}[y_j - \mu_j] = 0$.

$$\begin{aligned} \mathcal{I}(\beta)_{lk} &= \mathbb{E}[-H_{lk}] = -\mathbb{E} \left[\sum_{j=1}^n x_{jk} \frac{-x_{jl} \mu_j (1 + \alpha \mu_j)}{(1 + \alpha \mu_j)^2} - \frac{(y_j - \mu_j) \alpha x_{jl} \mu_j}{(1 + \alpha \mu_j)^2} \right] \\ &= -\sum_{j=1}^n x_{jk} \left(\mathbb{E} \left[\frac{-x_{jl} \mu_j (1 + \alpha \mu_j)}{(1 + \alpha \mu_j)^2} \right] - \underbrace{\mathbb{E} \left[\frac{(y_j - \mu_j) \alpha x_{jl} \mu_j}{(1 + \alpha \mu_j)^2} \right]}_{=0} \right) \\ &= \sum_{j=1}^n x_{jk} x_{jl} \frac{\mu_j}{1 + \alpha \mu_j} \end{aligned}$$

Going back to the iterative form of the *Fisher scoring algorithm*, it can be reformulated as:

$$\mathcal{I}(\beta) \beta^{n+1} = \mathcal{I}(\beta) \beta^n + \left(\frac{\partial \ell(\beta^n; y)}{\partial \beta} \right)$$

Introduce further the diagonal weight matrix W^n with:

$$w_i^n = \frac{\mu_i}{1 + \alpha \mu_i} = \frac{e^{x_i \beta^n}}{1 + \alpha e^{x_i \beta^n}}$$

The *Fisher information matrix* can be written in terms of the design matrix X and the weight matrix W^n :

$$\mathcal{I}(\beta)_{lk} = X^T W^n X_{lk} = \sum_{j=1}^n x_{jk} x_{jl} \frac{\mu_j}{1 + \alpha \mu_j}.$$

By setting $z_j = x_j \beta^n + \frac{y_j - \mu_j}{\mu_j}$ the equation can be transformed into a form like (A.2),

A. Appendix - Linear and Generalized Linear Models

suitable for the *IRLS* algorithm:

$$\begin{aligned}
\mathcal{I}(\beta)\beta^{n+1} &= \mathcal{I}(\beta)\beta^n + \left(\frac{\partial \ell(\beta^n; y)}{\partial \beta} \right) \\
(X^T W^n X)\beta^{n+1} &= (X^T W^n X)\beta^n + \left(\sum_{j=1}^n \frac{x_{jk}(y_j - \mu_j)}{1 + \alpha \mu_j} \right)_k \\
(X^T W^n X)\beta^{n+1} &= (X^T W^n X)\beta^n + X^T W^n \left(\frac{y_j - \mu_j}{\mu_j} \right)_j \\
\beta^{n+1} &= (X^T W^n X)^{-1} X^T W^n \left(X\beta^n + \left(\frac{y_j - \mu_j}{\mu_j} \right)_j \right) \\
\beta^{n+1} &= (X^T W X)^{-1} X^T W z
\end{aligned}$$

A.3. Elastic Net Regularization

Elastic net is a regression method that combines LASSO and ridge regression to achieve more stable fitting. It merges the feature selection capability of LASSO with the continuous shrinkage of ridge regression, mitigating issues such as collinearity, limited interpretability and non-unique solutions. This section briefly introduces the methodology following [Zou and Hastie, 2005].

Given the model defined in Section A.1.1, the coefficients β are estimated by minimizing the penalized sum of squares:

$$\hat{\beta} = \arg \min_{\beta} L(\lambda_1, \lambda_2, \beta) = \|y - X\beta\|_2^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2,$$

where the l_q norm of β is defined as $\|\beta\|_q = (\sum_{i=1}^p |\beta_i|^q)^{1/q}$ for $q \geq 1$ with p being the number of factors. A value of $\lambda_2 = 0$ indicates LASSO regression, while $\lambda_1 = 0$ results in ridge regression.

The LASSO penalty acts as a feature selection method, assuming that the predictors are sparse. If there are more parameters than observations, i.e $p > n$, LASSO chooses at most n parameters and sets the others to zero. While effective in many cases, LASSO has some limitations. For example, when predictors are highly correlated, LASSO tends to arbitrarily select only one variable from a correlated group and which variable is chosen can vary between runs. This instability reduces the model's interpretability and reproducibility. As illustrated in the contour plot in Figure A.1 (adapted from [Emmert-Streib and Dehmer, 2019]), the solution space is biased toward boundary solutions where the optimal point occurs at coefficients that are exactly zero for one or more variables.

A. Appendix - Linear and Generalized Linear Models

In contrast, Ridge regression does not assume sparsity and instead shrinks certain coefficients without setting any exactly to zero. This approach better handles multicollinearity among predictors but does not address the interpretability issue that arises when dealing with a large number of features.

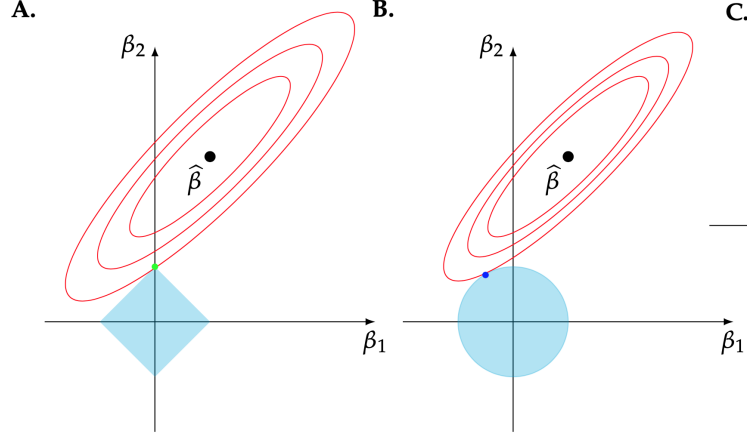


Figure A.1.: *Lasso (A) and Ridge (B) regression with two coefficients (β_1, β_2) . Red ellipses are loss contours, blue shapes are constraint regions. Dots mark the optimal solutions.*

This motivated the development of Elastic Net regularization, defined as a linear combination of the LASSO and Ridge penalties to overcome the limitations of both methods. Elastic Net is strictly convex, ensuring a unique solution. It performs automatic feature selection and continuous shrinkage while effectively selecting groups of correlated variables [Zou and Hastie, 2005]. Figure A.2 (adapted from [Emmert-Streib and Dehmer, 2019]) illustrates the geometric interpretation of the three penalties, with the Elastic Net constraint forming a diamond-shaped region.

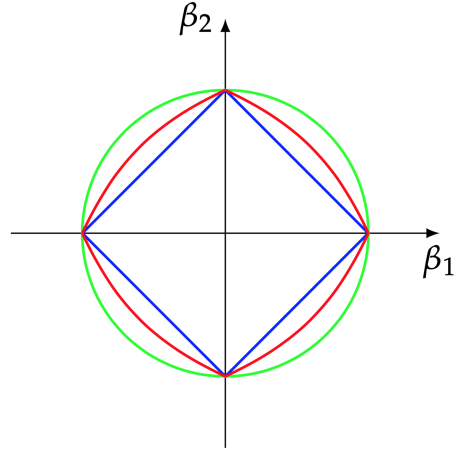


Figure A.2.: Contour plots of elastic net regularization (red) combining the L_2 -norm (green) of ridge regression and the L_1 -norm (blue) of LASSO.

B. Appendix - Empirical Bayes

Bayesian inference is a method for statistical inference, used for estimating the probability of a hypothesis or the likelihood that observed values come from a certain distribution. A central point is the use of an assumed fixed prior distribution, representing the initial assumptions on the data. After observing the data, the distribution probabilities are updated, giving a more accurate estimate.

Given data $y = (y_1, \dots, y_n)$, originating from a probability distribution $f(y|\theta)$, it is further assumed that the distribution parameters $\theta = (\theta_1, \dots, \theta_n)$ are unknown and distributed according to some prior distribution $\pi(\theta|\eta)$ with $\eta = (\eta_1, \dots, \eta_n)$ being a vector of hyperparameters.

The *Bayes Theorem* postulates that the prior distribution of θ can be updated using the knowledge of the observed data [Carlin and Louis, 2000, 2.1]. This is referred to as the posterior distribution, defined as:

$$p(\theta|y, \eta) = \frac{p(y, \theta|\eta)}{p(y|\eta)} = \frac{p(y|\theta, \eta)p(\theta|\eta)}{\int p(y, u|\eta) du} = \frac{f(y|\theta)\pi(\theta|\eta)}{\int f(y|u)\pi(u|\eta) du}$$

The term $\int f(y|u)\pi(u|\eta) du$ refers to the marginal distribution of y given the value of η [Carlin and Louis, 2000, 2.1].

Empirical Bayes methods in contrast to fully Bayesian analysis estimate the prior distribution using the observed data instead of fixing it in advance. The method of moments can be used to estimate the distribution parameters.

The EB estimator is then calculated as the posterior mean of the conditional distribution.

$$\theta^* = \mathbb{E}[\theta|y, \eta] \tag{B.1}$$

C. Appendix - Negative Binomial Distribution

The negative binomial distribution is a discrete distribution used to model overdispersed count data, particularly in biological applications such as RNA-seq experiments [Robinson and Smyth, 2008]. It can be derived through multiple formulations, the most common being as a Poisson-gamma mixture or as a member of the exponential family of distributions. Unlike the Poisson distribution, which assumes equal variance and mean, the negative binomial accounts for overdispersion by defining the variance function as $\mu + \alpha\mu^2 > \mu$ with α being a dispersion parameter, controlling the shape.

C.1. Model derivation

Let Y be a random variable such that $Y \sim NB(r, p)$. The probability mass function is given by [Barry, 2020]:

$$f(y; r, p) = \binom{y+r-1}{r-1} p^r (1-p)^y = \frac{\Gamma(y+r)}{\Gamma(r)\Gamma(y+1)} p^r (1-p)^y, \quad (\text{C.1})$$

where the gamma function Γ satisfies the identity $\Gamma(x) = (x-1)!$ and the binomial coefficient is defined as [Abramowitz and Stegun, 1972]:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}.$$

This formulation can be interpreted as the probability of observing y failures before the r -th success in a sequence of Bernoulli trials with success probability p .

Poisson-Gamma mixture

The negative binomial distribution can be derived as a compound distribution of a Poisson-distributed count variable with a gamma-distributed rate [Cameron and Trivedi, 1998].

C. Appendix - Negative Binomial Distribution

Let:

$$Y|t \sim \text{Pois}(t\lambda)$$

$$t \sim \text{gamma}\left(r, \frac{p}{1-p}\right)$$

It can be shown that marginalizing over the variable t yields the negative binomial distribution in the form of (C.1).

An alternative parametrization is given by [Cameron and Trivedi, 1998] by setting $r = \frac{1}{\alpha}$ and $p = \frac{1}{1+\alpha\mu}$:

$$f(y; \mu, \alpha) = \frac{\Gamma(y + \frac{1}{\alpha})}{\Gamma(\frac{1}{\alpha})\Gamma(y+1)} \left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^y, \quad (\text{C.2})$$

with μ being the mean and α the dispersion with the variance equal to $\mu + \alpha\mu^2$.

Exponential family

The negative binomial distribution with fixed number of failures r can also be converted into the exponential family form, essential for GLM-based inference [Hilbe, 2011].

$$f(y; r, p) = \exp\left(y \ln(1-p) + r \ln(p) + \ln\left(\frac{y+r-1}{r-1}\right)\right)$$

C.2. Maximum likelihood estimation

The likelihood function as defined in A.2.2 can be calculated for independent observations $(y_1, \dots, y_n)$ from a negative binomial distribution $NB(\mu, \alpha)$ as follows:

$$L(\mu, \alpha; y) = \prod_{j=1}^n f(y_j; \mu, \alpha) =$$

$$= \prod_{j=1}^n \frac{\Gamma(y_j + \frac{1}{\alpha})}{\Gamma(\frac{1}{\alpha})\Gamma(y_j+1)} \left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{y_j} \quad (\text{C.3})$$

Taking the logarithm yields the log-likelihood function:

$$\ell(\mu, \alpha) = \log L_g(\mu, \alpha) = \sum_{j=1}^n \log \Gamma\left(y_j + \frac{1}{\alpha}\right) - \log \Gamma(y_j + 1) - \log \Gamma\left(\frac{1}{\alpha}\right) +$$

$$y_j \log \alpha + y_j(\log \mu) - \left(y_j + \frac{1}{\alpha}\right) \log(1 + \alpha\mu) \quad (\text{C.4})$$

D. Appendix - Experiments Results

D.1. Gene count study

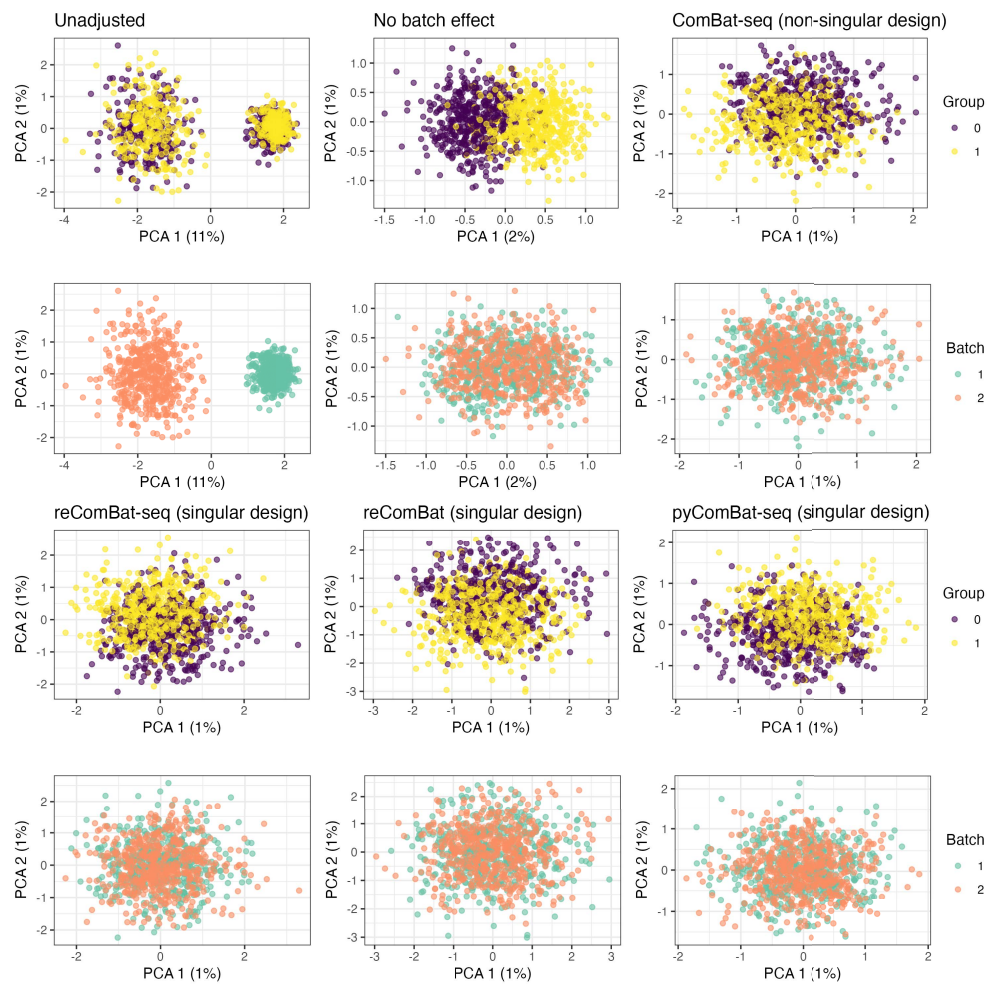


Figure D.1.: PCA plots of the simulated data with 128 genes, 1024 samples, 2 batches

D.2. Sample count study

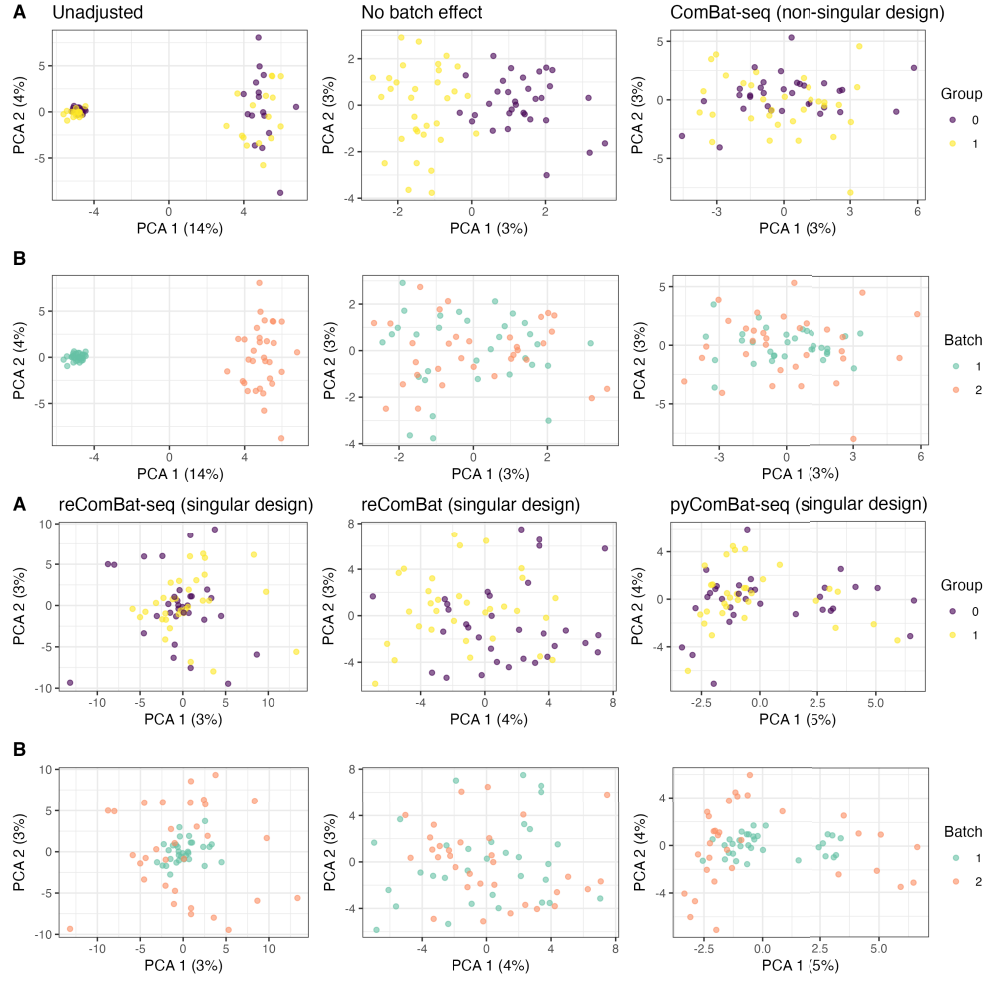


Figure D.2.: PCA plots of the simulated data with 5000 genes, 64 samples, 2 batches

D.3. Real data

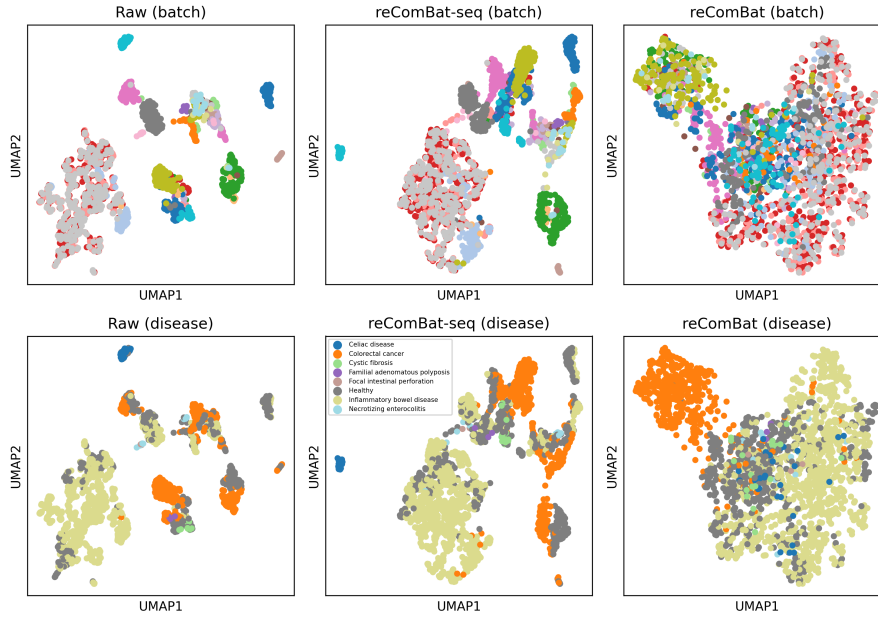


Figure D.3.: UMAP plots of the Bowel tissue data with 2014 samples, 43 batches

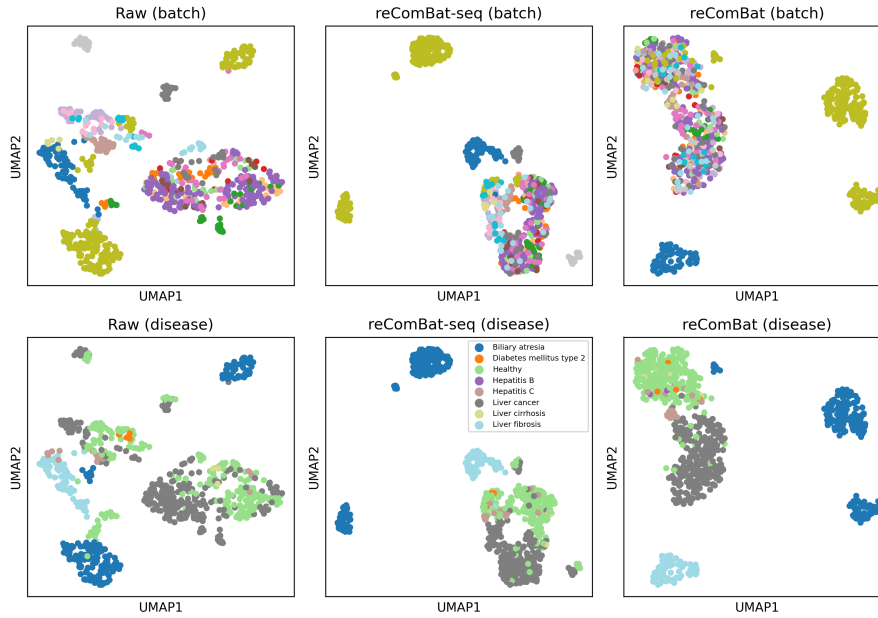


Figure D.4.: UMAP plots of the Liver tissue data with 764 samples, 31 batches

D. Appendix - Experiments Results

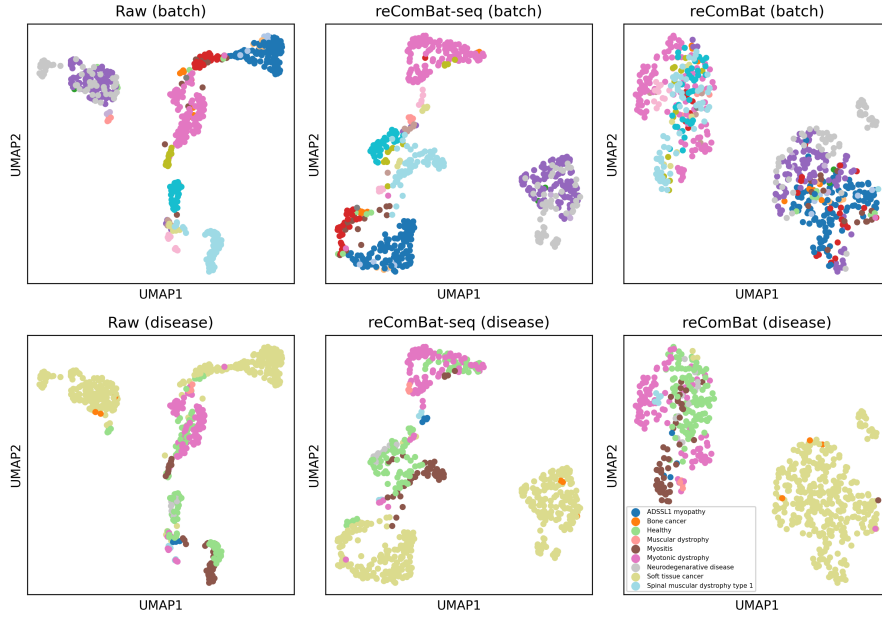


Figure D.5.: UMAP plots of the Skeletal muscle tissue data with 598 samples, 22 batches

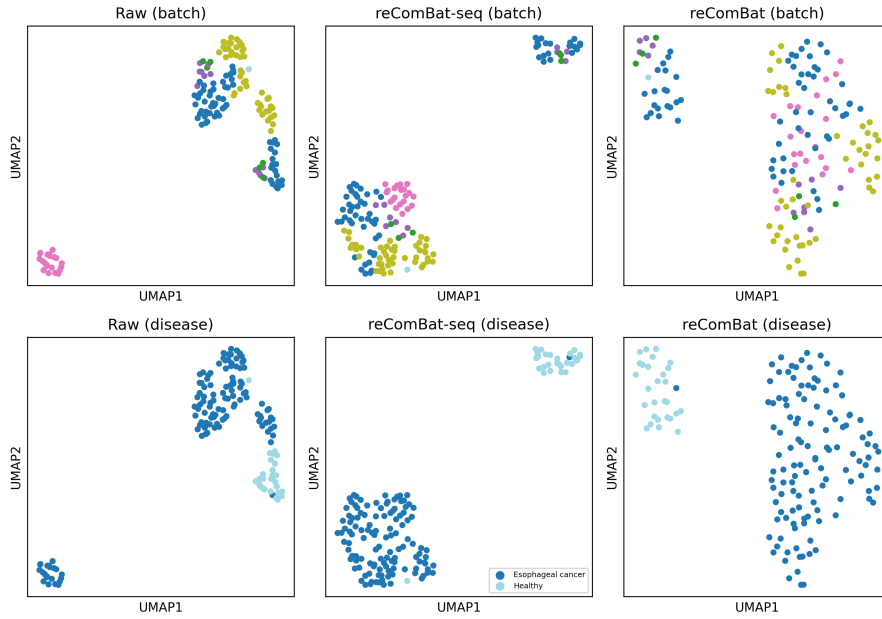


Figure D.6.: UMAP plots of the Esophagus tissue data with 149 samples, 6 batches